7 Step Model for cracking the HLD Interview Round

Average Time of HLD Round: 45 minutes - 1 hour

Following are the seven steps that you should follow to approach your system design interview:

1. **Scope of the system**

Before starting with the interview, the interviewee should clarify the scope of the system. We should begin the interview by asking questions to determine the scope of the problem. The majority of design questions are open-ended, with no single correct answer. As a result, it's vital to clear up any uncertainties early in the interview. Candidates that make an effort to define the system's end goals properly have a better chance of succeeding.

The following points should be clearly defined in this step of the interview process:

- Functional Requirements of the system
- Non-Functional Requirements of the system

  Some important non functional requirements to clarify are around:
  - Expected latency of response
  - Prefer consistency of the system or availability
- Design Goals

Sample Questions:

Users

1. Who will make use of it?
2. What would be the process of using the application?

Constraints

1. Identifying traffic and data processing restrictions at scale is the primary goal.
2. System scale (number of requests per second, categories of requests, data written per second, data read per second, expected growth period (Also around expected growth in say next 5 years. This is so that you can design something that horizontally scales for lets say 20x user base in future))
3. Multi-threading, read/write oriented, and other system needs.

For Example:

During an interview for designing Facebook, you can ask:

- Who can create a post? (Users, Sample answer: All authenticated and authorised users)
- Who all can read and interact with the post? (Users, Sample answer: Depends on whether the account is public or private)
- Can a user like and comment on a post if they like it? (Sample answer: Yes)
- What are the modes of interaction on a post? (Sample answer: Comment, like, share, follow)
- What would be the average number of daily active users (Sample answer: 50 million users post every day)
- Are we designing client/server interaction, backend architecture, or both (Sample answer: we want to understand client/server interaction, but we'll concentrate on scaling the backend).

---

2. System Interface Definition

You're halfway there if you've gathered all of the requirements and identified the system's APIs.

Define which APIs the system should provide. This will not only establish the exact contract required from the system, but it will also guarantee that no requirements have been overlooked.

Example:

For our Instragam service, here are some examples:

- createPost(user_id, post_id, image_url, user_location, timestamp, …)
- generateTimeline(user_id, current_time)
- recordUserPostLikes(user_id, post_id, timestamp, …)

Tip: Also mention the expected output of the APIs (and preferably, also the data type of all arguments and returned value)

---

3. Capacity Estimation

Estimating the scope of the system you are going to develop is always a good idea. Later on, when you're working on scaling, partitioning, load balancing, and caching, this will come in handy.

Sample Questions:

- What is the system's expected scale? (e.g., number of new posts every day, number of post views, how many timeline generations per sec., etc.)
- How much space would we require? Whether or whether users can include photographs and videos in their posts will determine this. (Tip: When calculating the db size, do multiply the result by 2 or 3 in order to account for data replication.)

- What kind of network bandwidth use do we anticipate? This would be critical in determining how we would manage traffic and load balancer among servers.

---

4. Defining the data model

Defining the data model early on will help to understand how data will move between system components. It will later assist you in better data splitting and administration. The candidate should be able to recognise various system entities, how they will interact with one another, and multiple aspects of data management such as storage, transmission, encryption, and so on.

For Example:

For our Instragram service, here are some entities:

- User: UserID, Name, Email, DoB, CreationData, LastLogin, etc.
- Post: PostID, Content, PostLocation, NumberOfLikes, TimeStamp, etc.
- UserFollows: UserdID1, UserID2
- FavoritePost: UserID, PostID, TimeStamp

Sample Points to discuss in the interview:

- Which would be the most optimal database schema to be used in this case?
- Is NoSQL, such as Cassandra, the most excellent fit for our purposes, or should we go with a MySQL-like solution?
- To store photographs and videos, what form of blob storage should we use?
- Usually it's also good to mention about your indices for fast search queries (both for SQL and NoSQL DBs) and which column would you choose as index in different DBs

5. High-level architecture design (Abstract design)

Create a block diagram depicting the system's essential components. You should determine how many components are required to solve the problem from beginning to conclusion.

During an interview:

Draw the main components and their connections, but don't go into too much detail. You can include:

A. Application service Layer
B. Make a list of the different services that are necessary for your system.
C. Data storage layer
D. Scalability
    a. Web Server (load balancer)
    b. Queues(For any async processing and background jobs)
    c. service (service partition)
    d. database (master/slave database cluster)
    e. cache systems

For Example:

While designing Facebook, you can discuss:

At a high level, numerous application servers would be required to serve all read/write requests for Facebook-like service, with load balancers in front of them to distribute traffic. We can set up different servers to handle reads and writes if we anticipate a high volume of read traffic (as opposed to write traffic). We'll need an efficient database on the backend to hold all of the posts and support many reads.

We'd also need a distributed file storage system for photographs (and videos), as well as a search index and infrastructure to allow for post searching.

---

6. Component Design

Expand on 2–3 components; the interviewer's input should always direct you to the aspects of the system she wants you to explain in greater detail. You should describe various approaches, their benefits and drawbacks, and why you would choose one over the other. Also, remember that there is no one-size-fits-all solution; the only thing that matters is that you weigh the benefits and drawbacks of various solutions while keeping system limits in mind.

During an interview, you need to discuss:

- For each of the components, specialised APIs, which would be necessary.
- Designing a database schema.

For example:

While designing Facebook, you can think and discuss:

- How should we split our data to distribute it across various databases since we'll be storing a large amount of data? Should we aim to keep all of a user's data in the same database? What problems may it cause? (Tip: Good thing is to discuss about both horizontal sharding strategy and vertical sharding, and decide which one to go ahead with based on the pros and cons of each)
- How would we deal with high-traffic users, such as celebrities with millions of fans?
- Should we strive to store our data in an optimal way to scan the most current (and relevant) posts, as the user's timeline will contain the most recent (and appropriate) posts?

- To speed things up, how much and at which layer should we introduce cache?
- What parts of the system require more load balancing?

---

7. Understanding Bottlenecks

Try to cover as many bottlenecks as possible, as well as potential solutions to the problems.

Sample questions to discuss in an interview:

- To accommodate user requests, your system may require a load balancer and many machines. Perhaps the data is so large that you need to spread it across numerous devices. What are some of the negative consequences of doing so?
- Is the database running slowly and in need of some in-memory caching?
- Also what type of cache would help here based on invalidation strategy?
- Discuss about different ways of replication, and how it can vary in stateless and stateful services.
- Is our system susceptible to a single point of failure? So, what are we going to do about it?
- Do we have enough data copies to continue to serve our users even if a few servers fail?
- Do we have enough copies of different services running, for example, that a few failures won't force the entire system to shut down?
- How do we keep track of our service's performance? Do we be notified if one of our essential components fails or if its performance deteriorates? (Tip: Monitoring and Alerting is an underrated aspect of system design. This is definitely worth mentioning in the interview.)