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

### Step 1: Requirements clarifications

Let’s expand this with an actual example of designing 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 the user’s timeline?
* Will tweets contain photos and videos?
* Are we focusing on the backend only or are we developing the front-end too?
* Will users be able to search tweets?
* Do we need to display hot trending topics?
* Will there be any push notification for new (or important) tweets?

### Step 2: Back-of-the-envelope estimation

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, number of timeline generations per sec., etc.)?
* How much storage will we need? We will have different storage requirements if users can have photos and videos in their tweets.
* What network bandwidth usage are we expecting? This will be crucial in deciding how we will manage traffic and balance load between servers

### Step 3: System interface definition

Define what APIs are expected from the system. This will not only establish the exact contract expected from the system but will also ensure if we haven’t gotten any requirements wrong. Some examples of APIs for our Twitter-like service will 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 4: Defining data model

in the early part of the interview will clarify how data will flow between different components of the system. Later, it will guide for data partitioning and management.

The candidate should be able to identify various entities of the system, how they will interact with each other, and different aspects of data management like storage, transportation, encryption, etc.

**User:** UserID, Name, Email, DoB, CreationData, LastLogin, etc.  
**Tweet:** TweetID, Content, TweetLocation, NumberOfLikes, TimeStamp, etc.  
**UserFollowo:** UserdID1, UserID2  
**FavoriteTweets:** UserID, TweetID, TimeStamp

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

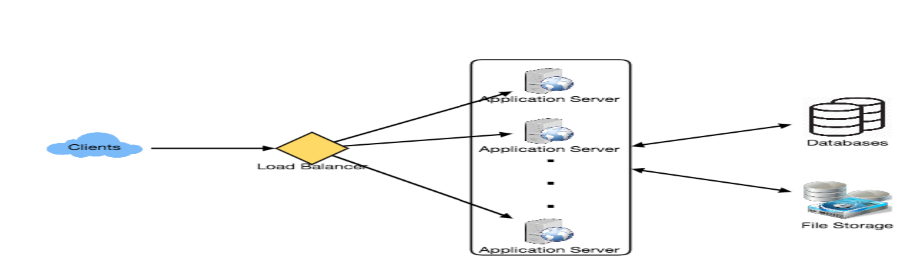
### Step 5: High-level design

Draw a block diagram with 5-6 boxes representing the core components of our system.

If we’re assuming that we will 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 will also need a distributed file storage system for storing photos and videos.

For Twitter, at a high-level, we will need multiple application servers to serve all the read/write requests with load balancers in front of them for traffic distributions.



### Step 6: Detailed design

Dig deeper into two or three major components; interviewer’s feedback should always guide us to what parts of the system need further discussion.

We should be able to present different approaches, their pros and cons, and explain why we will prefer one approach on the other.

**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 have enough replicas of the data so that if we lose a few servers, we can still serve our users?
* Similarly, do we have 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 degrades?

**In short, preparation and being organized during the interview are the keys to be successful in system design interviews. The steps mentioned above should guide you to remain on track and cover all the different aspects while designing a system.**

# Designing a URL Shortening service like TinyURL

### 1. Why do we need URL shortening?

Users are redirected to the original URL when they hit these short links. Short links save a lot of space when displayed, printed, messaged, or tweeted. Additionally, users are less likely to mistype shorter URLs.