**2. Example system design interview answer**

To illustrate the framework discussed above, we’ve laid out an example answer to a real system design question that was asked in an [Amazon TPM interview](https://igotanoffer.com/blogs/tech/amazon-technical-program-manager-interview), according to data from Glassdoor.

**Question**

“Design a database for a tiny URL implementation.”

**Ask clarifying questions**

***Clarify the scope of the question***

**Candidate:** “Okay, I’m going to restate the question first, to make sure that I understand what you’re asking me. As I understand a tiny URL system, a user will input a URL and the system will return a short ‘pointer’ URL to the full URL. It would be something like *‘https://tiny.url/shortCode*’. When someone visits the short URL, they will be redirected to the full URL. The service would generate *‘shortCode’* as a unique reference to the URL. Am I correct that I need to design the persistent storage component of a tiny URL system?”

**Interviewer:** “Yes, that’s correct.”

**Candidate:** “Great. To clarify, we'll need to design the data schema, database architecture, and choose a database technology to use.”

**Interviewer:** “Yes that’s correct”

**Candidate:** “Do we need to specify a particular brand or instance of database engine, or can we specify the technology type, e.g. relational, key-value, document, etc?”

**Interviewer:** “Let's keep it at the level of technology type, we don't need to specify the vendor or engine.”

***Specify your assumptions***

**Candidate:** “Great, thank you. I'm assuming this system would be for a large number of users and the links would need to be available for an extended period of time, or perhaps indefinitely? In other words, a simple in memory or local file store would not be suitable for this system”

**Interviewer**: “Yes, definitely.”

**Design high-level**

***Calculate metrics***

**Candidate:** “Great. How many users are we catering for? Or in other words, how many URLs would we be generating each second, and how many URLs would we be serving each second? I'm assuming that it would be highly asymmetric, in that there would be many more URLs served each second than URLs generated?”

**Interviewer:** “Yes, it would be  asymmetric. We would need to cater for a high volume of traffic, as it would be used as the default URL shortener for a popular social network, which has about 300 million daily active users.”

**Candidate:** “I see. Okay, let me try some rough calculations. Let's say 20%, or 60 million, users are frequent original posters (assuming that most social users consume or share existing posts). If they post something with a URL, say, once per day, then we would need to create 60 million URLs per day. *60 million / (24 hours \* 3600 seconds in an hour) = 695* URLs created each second, if equally spaced across the day.”

**Interview:** “That's probably high for current users and usage, as not everything posted is a link, but the user base is growing, so it will be a good estimate for the next few years.”

**Candidate:** “Cool, we'll go with those numbers then. For URLs served, I'm assuming there would be a spread of popularity. Some shortened URLs could have zero clicks, while a few may go viral and get tens or hundreds of thousands, or even millions of clicks.”

**Interviewer:** “Sounds like a reasonable assumption.”

**Candidate:** “All right, let's assume that on average a URL would be read maybe 100 times, heavily weighted to the most popular URL. So we can say there are roughly 100 times more reads per second than writes.”

**Interviewer:** “Reads are definitely higher than writes, but that sounds a bit low as the ratio.”

**Candidate:** “Let's increase it by an order of magnitude, say, 1000 times?”

**Interviewer:** “Okay, sounds good.”

**Candidate:** “We will be generating a lot of URLs! Would we need to keep them forever, or should they expire after some time?”

**Interviewer:** “Hmm, I think we can expire them after some time.”

**Candidate:** “Okay, we'll keep track of when a URL is created, so we can remove it.”

**Interviewer**: “Cool”

***Identify high-level components or services***

**Candidate:** “Now that we know the scale that we are designing for, a single server clearly will not meet the load or availability requirements.”

**Interviewer:** “That’s true”

**Candidate:** “Ok, so I’m thinking we’ll need multiple API or web servers, and we’ll also need multiple database servers, to create a distributed system. This way we’ll be able to meet the high demand and availability requirements”

**Interviewer:** “Yes, a distributed system sounds like the way to go here”

**Candidate:** “We’ve established that a few of the links will get the most traffic. This seems like a good candidate for a caching layer, so we can reduce the load on our databases and make the most used links quicker to access.”

**Interviewer:** “That's a good idea.”

***Design the database***

**Candidate**: “OK, so now I’d like to determine what the data schema should look like”

**Interviewer:** “Go ahead”

**Candidate:** At the minimum, we’ll be storing the full URL, so we can redirect to it, and the short code, which must be unique per URL. Since we also look up links by the short code, it could also be the primary key or identifier for the record.

**Interviewer**: Yes, the short code would be unique. Is that the only data to store?

**Candidate:** Earlier we also mentioned that links should expire, so we’d need to store a time stamp. Perhaps either the date the link should expire, or the date it was created.

**Interviewer**: Is there a difference there?

**Candidate**: If we store the date created, we’ll be able to change expiry policy parameters a bit easier, and enable more flexibility there. It would also be simple to report on tiny url’s created per time frame, without back calculating. If we store the expiry date though, its means we won’t need to calculate it from the created date, and that could speed up purging old records. We could store both as well.

**Interviewer:** Ok, sounds good to store both, enabling a bit of flexibility.

**Candidate:** Cool, we could also store the user that created the link, if the service records that. Do we have a user Id to store, or would it be anonymous?

**Interview:** It would be users of the social network, so they would have a user Id.

**Candidate**: Ok great. As far as I see it, we only really have 1 table or document, or value object to store. It would look a bit like this:

Link Record

—————

shortCode

fullUrl

createdDate

expiryDate

userId

**Candidate:** “We could have a bit of a problem with introducing a distributed system now though”

**Interviewer**: “What's that?”

**Candidate:** “Well, we need to generate a unique *‘shortCode*’ for each of the URLs right?”

**Interviewer:** “That's right.”

**Candidate:** “On a single database server architecture, that’s simple – we'd just let the database generate an ID for us. But we will be running multiple servers, or partitions, and if each server is creating an ID, we will be bound to get collisions.”

**Interviewer:** “That's true.”

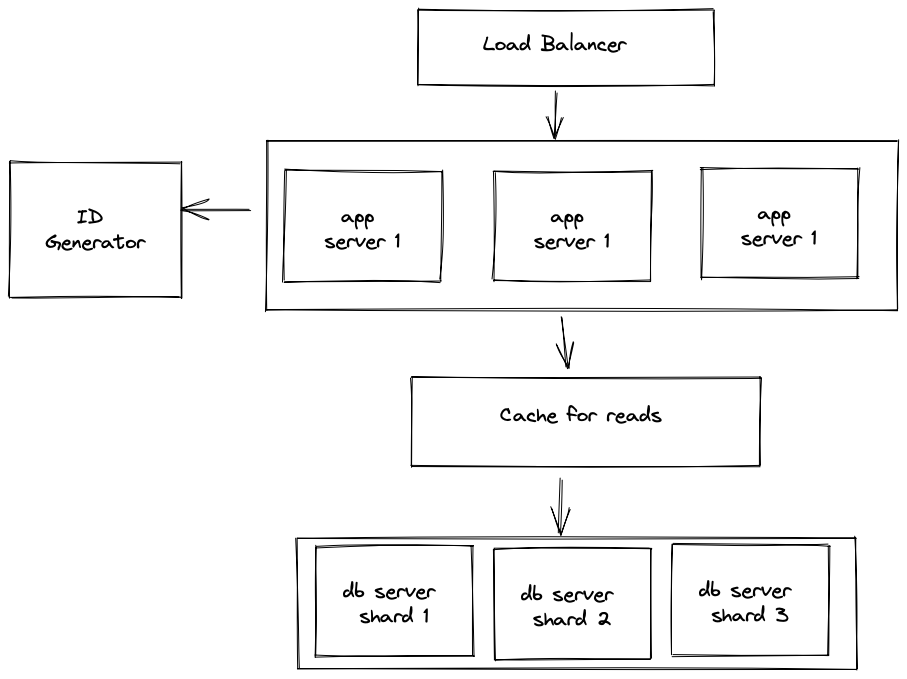
**Candidate:** “So, we'll need some other scheme to come up with an ID for the *‘shortCode*’. We could make a central, shared ID generator that each server can call when creating a new short code. That way we can guarantee that the generated short code will be unique.”

**Interviewer**: I guess that would solve the collision problem.

**Drill-down on your design**

***Draw an architecture diagram***

**Candidate:** “Ok, so I think this is the basics of what the system would look like with the major components:”



**Interviewer** “I see you've added in a load-balancer?”

**Candidate:** “Yes, it's not part of the data design, but I thought I'd add it in for completeness.”

***Identify bottlenecks***

**Candidate:** “Looking at the diagram, I guess a potential problem is that now the whole system relies on the central ID generator being up all the time. If it fails, then we won't be able to create new short codes, although reading would still work.”

**Interviewer**: Yes, that sounds like it could be a problem. Is a potential failure of the central ID generator the only issue with it?

**Candidate:** I guess another problem could be that since there is only 1 ID generator, the speed and throughput of the system is limited by it. It would have to be very fast if it were to manage the amount of traffic, particularly in peak times.

**Interviewer:** “That’s certainly a problem”

**Candidate:** “I think we could fix it though. We could make a few more ID generators, and the API servers could ‘round robin’ between them to get a new number. Each of the servers would be able to serve from an assigned range of numbers, so that they don't return the same numbers. So ID generator 1 would serve from say 0-50 billion, generator 2 from (50 billion +1) to 100 billion, and so on. Then if one ID generator goes down, the servers can still create short codes, just from a different range.”

**Interviewer**: Making the ID generator effectively distributed as well.

**Candidate**: Yes, exactly.

***Dive into a specific component***

**Candidate:** “If you’re OK, I’d like to dive deeper into this ID generation function. How does that sound to you?”

**Interviewer:** “Sounds interesting, continue.”

C**andidate:** “Well, ff we extend the concept of predefined ranges, instead of returning a single ID, we could return a list of numbers to the application servers. That way things are sped up, as the application servers do not need to call the ID generators each call, which also makes it even more reliable. They can assign a *shortCode* from the list of numbers, and only when they run out do they need to call one of the ID generator servers again.”

**Interviewer:** “So extra reliability and speed at the same time?”

**Candidate:** “Yeah! I've not used it, but I've seen systems like ZooKeeper that I understand can manage this distributed key generating scheme. Then we can use existing, reliable software for this part, instead of writing it from scratch, especially since it is a core function of the system.”

**Interviewer:** “Sounds promising”

**Candidate: “**Another detail of the ID generator, is what format does the ID take.”

**Interviewer**: “Can you expand on that?”

**Candidate:** Well, we’ve discussed that the ID generators would generate numbers within ranges, in other words generate integers. If we simply return the integer number, it soon could become quite a large number. Our tiny URL would soon become a very long URL, the more the service is used. If we re-encoded that ID as a number in a number system that uses letters and numbers and is also case sensitive, which could be a base62 representation, then we would be able to constrain the length of the *‘shortCode*’ to be, well, short.”

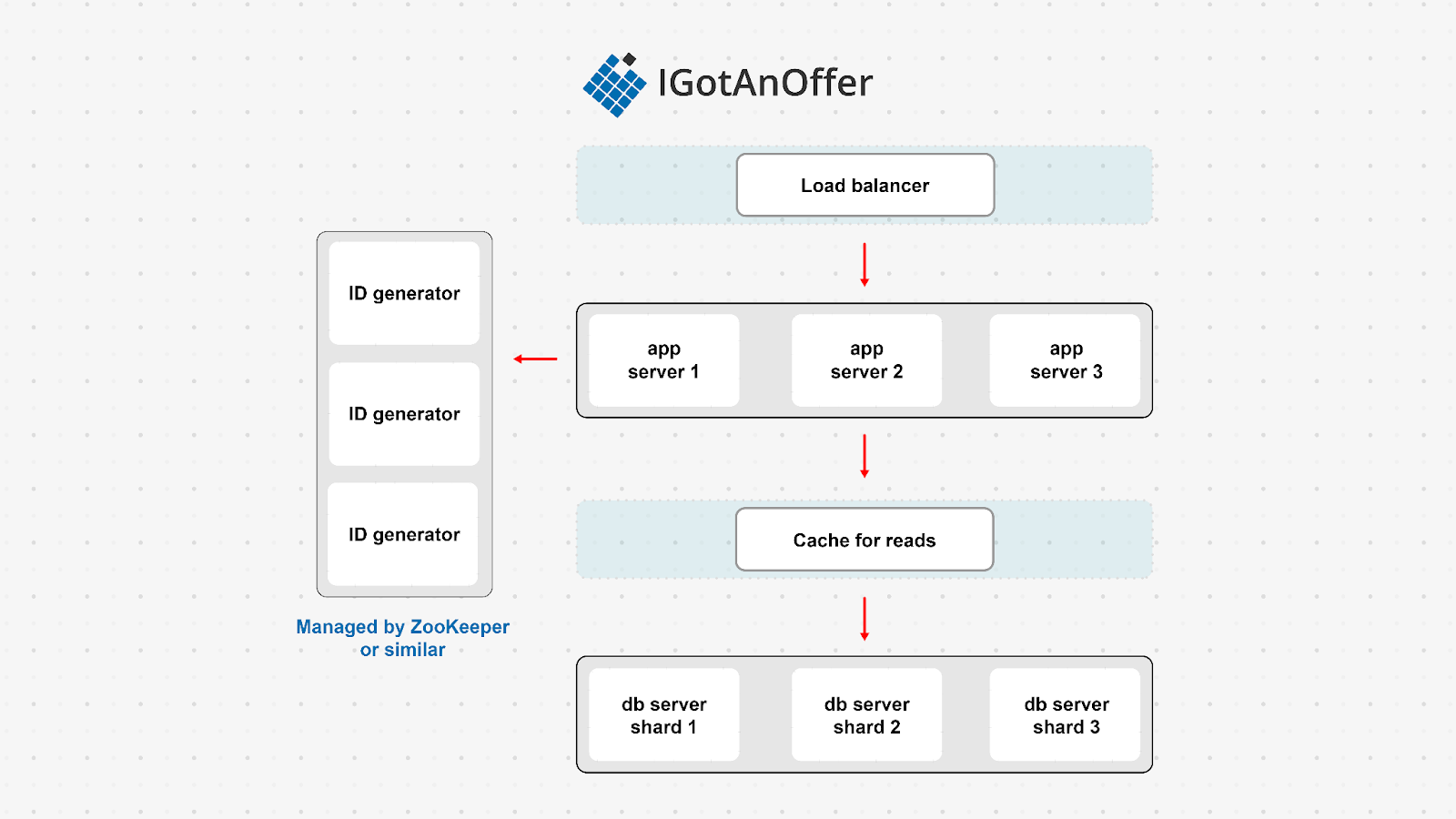
**Interviewer:** “Ah, so after a year of generating short codes, we'd be at ID *‘60million\*365 = 21 900 000 000*’, which is 11 characters long. But if we represented it in a higher base, like base62, it would be something like *‘nU6aqs’*, only 6 characters, making it a lot shorter.”

**Candidate:** “Yeah, exactly that.

**Bring it all together**

***Refine the architecture***

**Candidate:** “Ok, so here is the revised architecture with the the distributed ID generation part”



**Interviewer**: “Seems to be coming together”

**Candidate:** “Yes, though one thing we haven't settled on yet is the database technology. I'm thinking that since the schema is quite simple, and the link records are independent (i.e. they’re not related to each other), we are not constrained to traditional relational databases. Various NoSQL solutions have good support for sharding and horizontal scaling. I'm just thinking about whether a key-value store or a document-oriented database would be the better option.”

**Interviewer:** “Sounds logical. What would drive the consideration between the two technologies?”

**Candidate:** “I think it depends on how much information we are storing for each link, and how that information is used. In a key-value store, it is not usually straightforward to query on anything other than the key. We have a few other fields that might be useful to query on, especially the userId and expiry dates.”

**Interviewer:** “I think it's reasonable to assume that there would queries at least occasionally beyond just the key.”

**Candidate:** “Okay, in that case, I think we should go with a document store, as it will provide that flexibility.”

**Interviewer**: “Makes sense.”

**Candidate:** “I think we just need to check that the design satisfies our initial objectives now.”

**Interviewer**: “Okay.”

**Candidate:** “The main requirement was that it generated a unique short URL given an arbitrary one. It also needs to scale to handle a large amount of traffic, perhaps up to about 600 writes per second, and 100-1000 times more reads. It should also be highly available. I think the design meets those requirements. It generates a short URL using a base62 representation of the record ID. It handles scale by making use of database sharding, a distributed key generation system, and a cache for the most popular links. It also handles availability by having multiple servers, or redundancy, for each key component.”

**Interviewer**: “Good – it sounds like the only unknown part for you in the system is the lower level details of ZooKeeper?”

**Candidate:** “That's right, I haven't used it or configured it, but I would try to use ZooKeeper or something similar before writing a custom solution there. It sounds like a common problem that it solves, so I would think there are existing systems we could leverage.”

***Questions for the interviewer***

**Interviewer**: “Okay, thanks. I think we're almost out of time for this section, so do you have any other questions or concerns, whether for this design or something else.”

**Candidate:** “Yeah, one question I have is if this kind of blue sky design is a usual part of the job, or is it more weighted to maintaining existing solutions? ”

**Interviewer**: “Well, at the scale that we operate, even seemingly small features can require a significant amount of engineering. As you mentioned earlier, a solution for tiny url system at low scale and availability requirements would simply be a lookup table, perhaps backed up to a file. So yes, there are often new ground up solutions needed, and it is a large part of the work in this job.”

*—————————————————————————————————————————--*

*This part will be tricky. I’d like to have a unique example answer here, but I don’t think I have the technical knowledge/background to make a convincing enough example for system design, as opposed to behavioral or prioritization.*