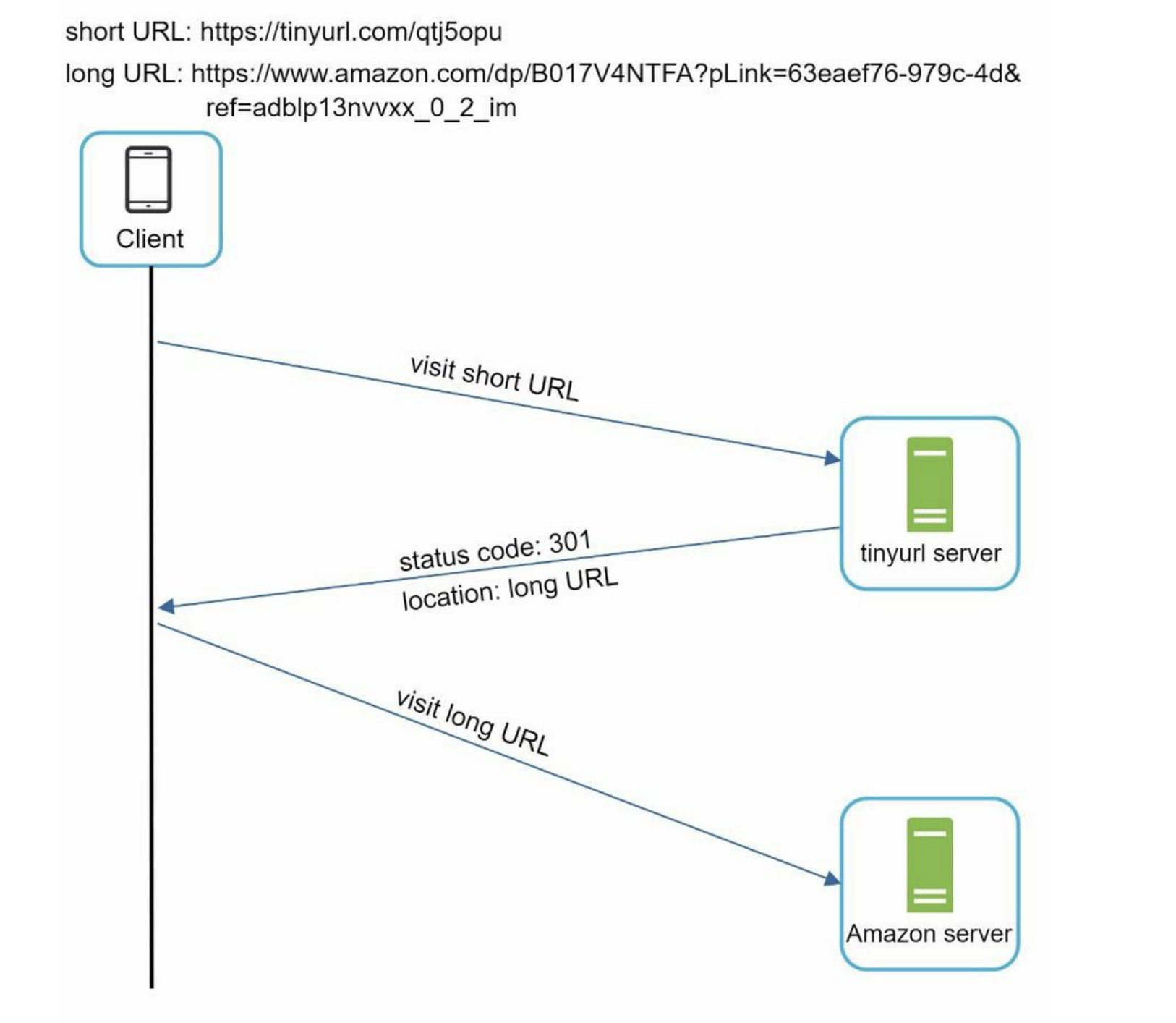
**URL SHORTNER (tiny url)**

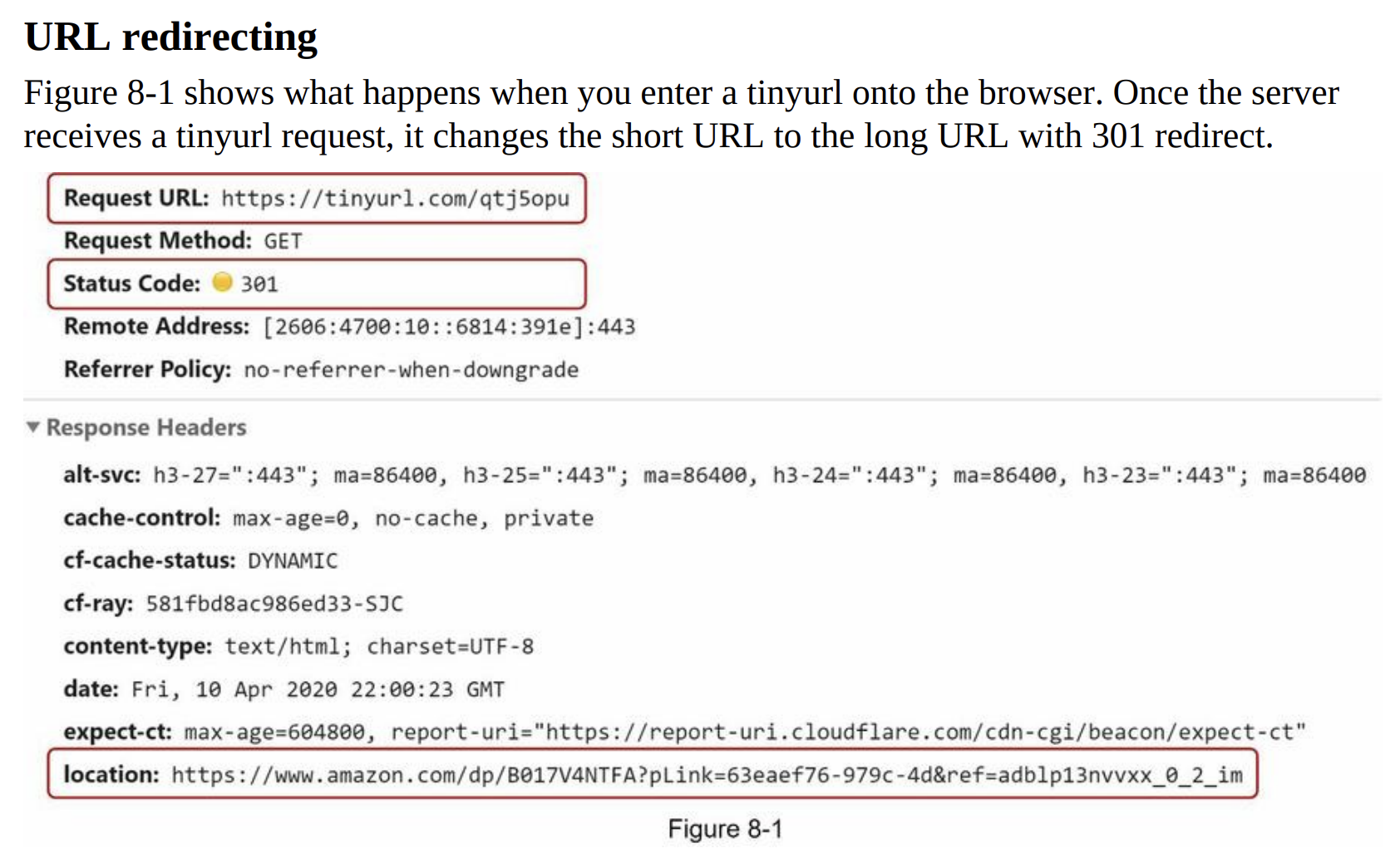
<https://www.systeminterview.com/q=chatsystem&c=loggedin&v=v3&l=long> is the original URL. Your service creates an alias with shorter length: <https://tinyurl.com/y7keocwj>

If you click the alias, it redirects you to the original URL.



301 redirect. A 301 redirect shows that the requested URL is “permanently” moved to the long URL. Since it is permanently redirected, the browser caches the response, and subsequent requests for the same URL will not be sent to the URL shortening service. Instead, requests are redirected to the long URL server directly.

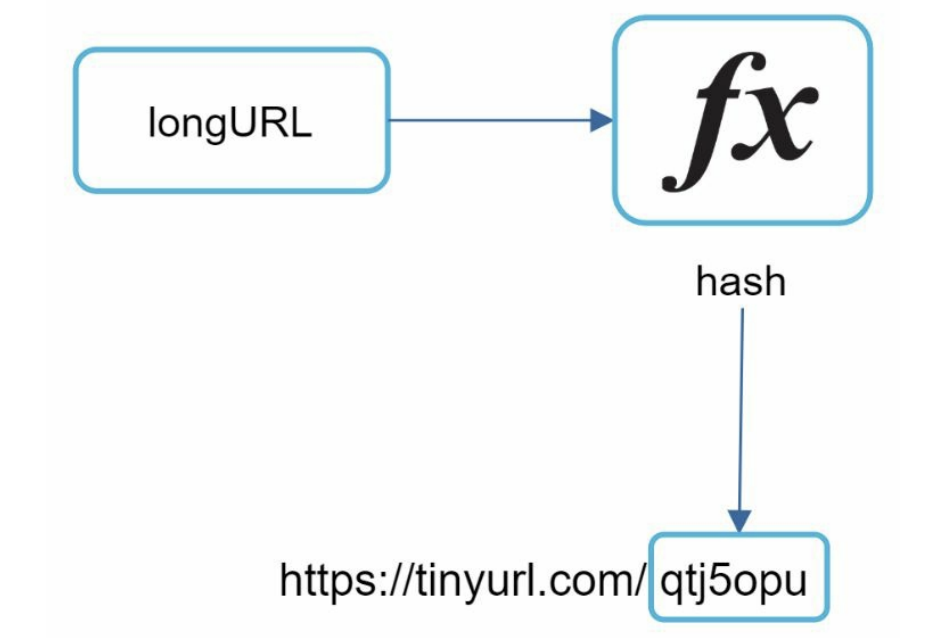
302 redirect. A 302 redirect means that the URL is “temporarily” moved to the long URL, meaning that subsequent requests for the same URL will be sent to the URL shortening service first. Then, they are redirected to the long URL server.



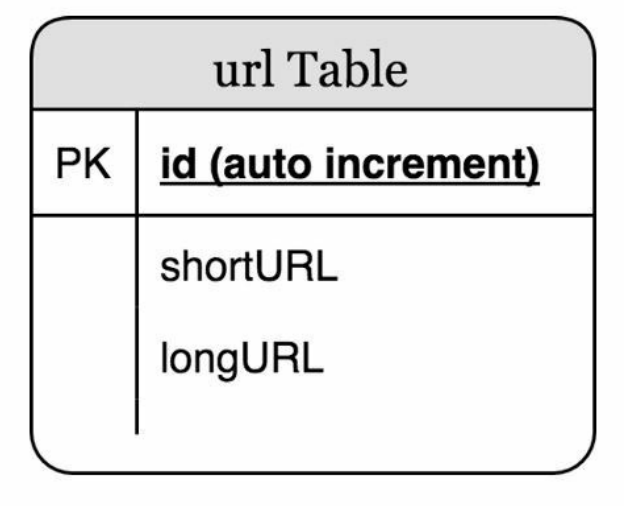
**URL Shortening**

The most intuitive way to implement URL redirecting is to use hash tables. Assuming the hash table stores pairs <shortURL, LongURL>, URL redirecting can be implemented by the following:

* Get longURL: longURL = hashTable.get(shortURL)
* Once you get the longURL, perform the URL redirect.

****

**Better to use Relation model to store key-value pairs**

****

**Hash value length**

The hashValue consists of characters from [0-9, a-z, A-Z], containing 10 + 26 + 26 = 62 possible characters. To figure out the length of hashValue, find the smallest n such that 62^n ≥ 365 billion.

The system must support up to 365 billion URLs based on the back of the envelope estimation.

When n = 7, 62 ^ n = ~3.5 trillion, 3.5 trillion is more than enough to hold 365 billion URLs, so the length of hashValue is 7

Basic two types of hash functions for a URL shortener:

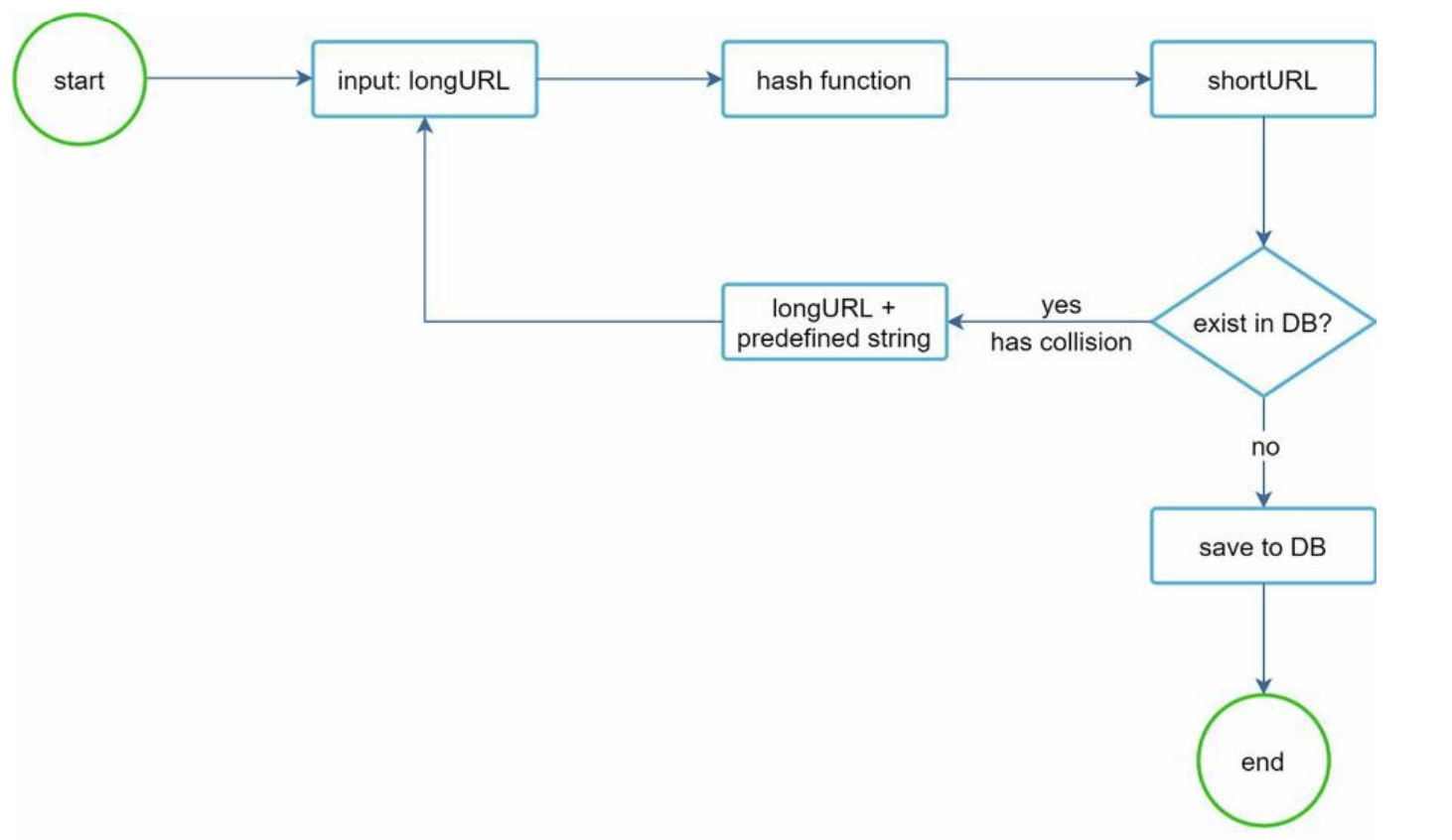
1. Hash + collision resolution
2. base 62 conversion

**Hash + collision resistant**

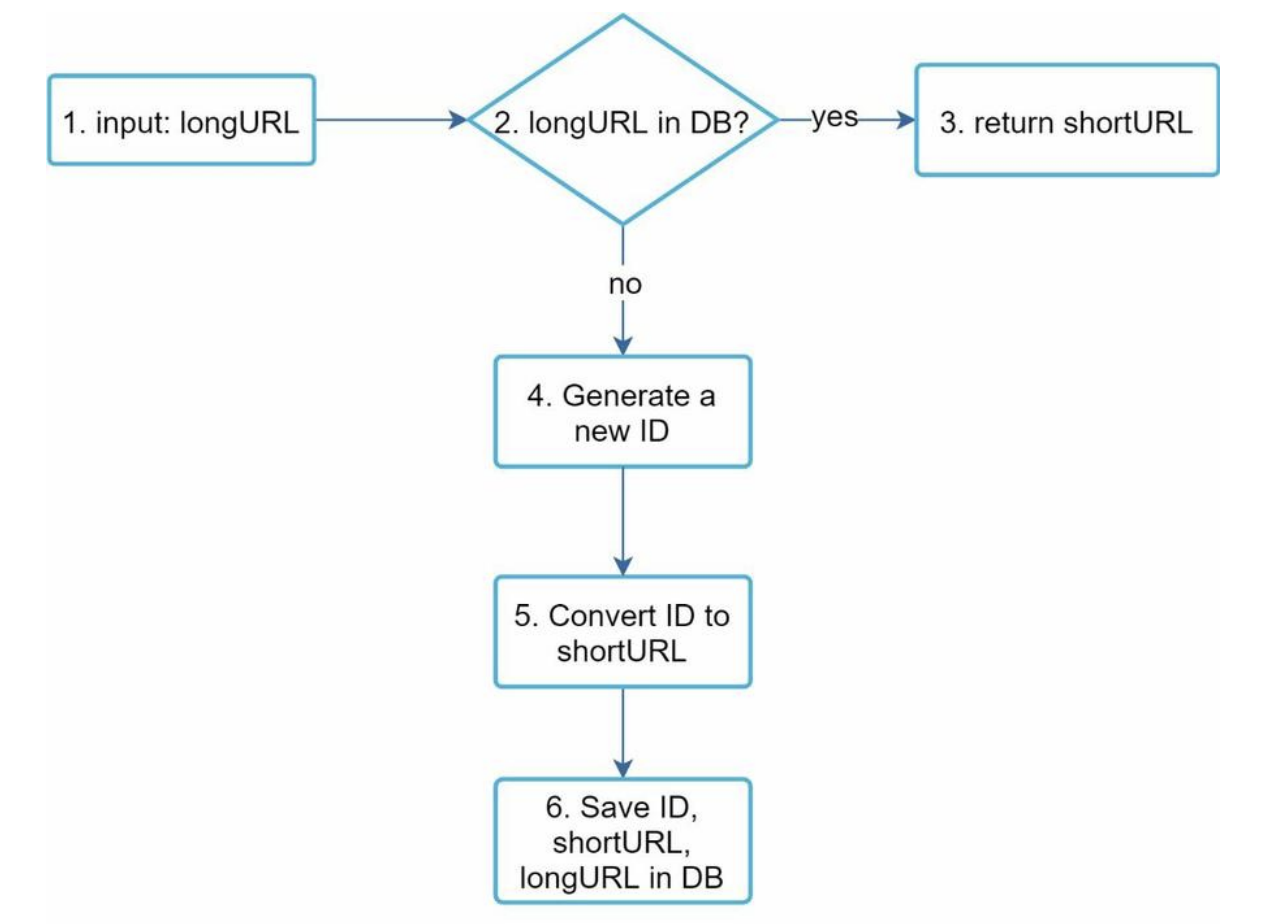
To shorten a long URL, we should implement a hash function that hashes a long URL to a 7- character string. A straightforward solution is to use well-known hash functions like CRC32, MD5, or SHA-1

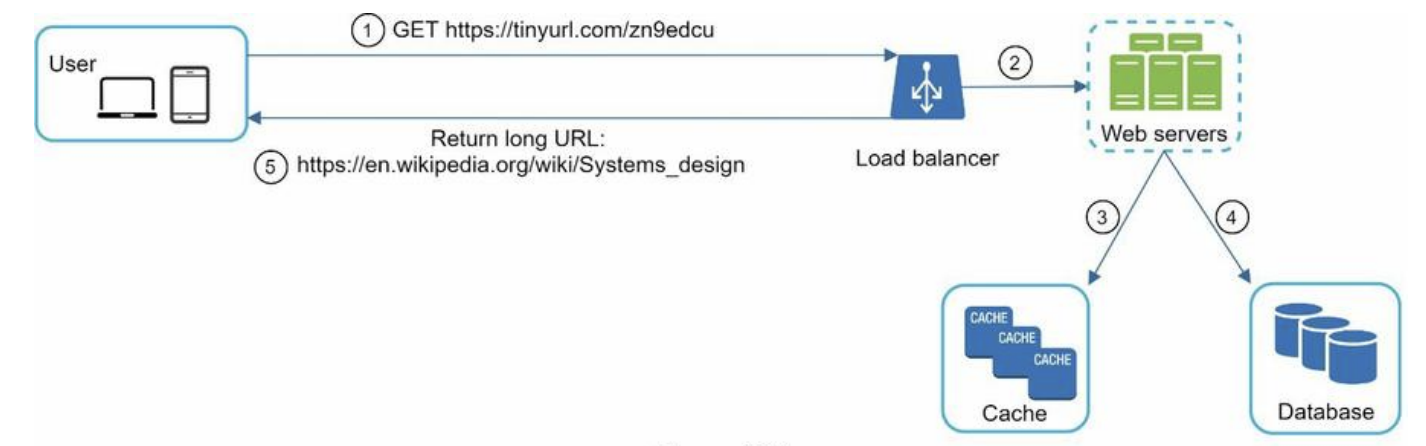
The first approach is to collect the first 7 characters of a hash value; however, this method can lead to hash collisions. To resolve hash collisions, we can recursively append a new predefined string until no more collision is discovered

This method can eliminate collision; however, it is expensive to query the database to check if a shortURL exists for every request. A technique called bloom filters [2] can improve performance. A bloom filter is a space-efficient probabilistic technique to test if an element is a member of a set.

****

**URL shortening deep dive**

****

**HLD  
  
**

The flow of URL redirecting is summarized as follows:

1. A user clicks a short URL link: <https://tinyurl.com/zn9edcu>

2. The load balancer forwards the request to web servers.

3. If a shortURL is already in the cache, return the longURL directly.

4. If a shortURL is not in the cache, fetch the longURL from the database. If it is not in the

database, it is likely a user entered an invalid shortURL.

5. The longURL is returned to the user.

**Can also think about :**

* Rate limiter: A potential security problem we could face is that malicious users send an overwhelmingly large number of URL shortening requests. Rate limiter helps to filter out requests based on IP address or other filtering rules
* Web server scaling: Since the web tier is stateless, it is easy to scale the web tier by adding or removing web servers
* Database scaling: Database replication and sharding are common techniques
* Analytics: Data is increasingly important for business success. Integrating an analytics solution to the URL shortener could help to answer important questions like how many people click on a link