# URL shortener

## Problem and design scope

Qu: url shortener example
ans: https://www.systeminterview.com/q=chatsystem&c=loggedin&v=v3&l=long is the original URL.

output: [https://tinyurl.com/y7keocwj](https://tinyurl.com/y7keocwj)

Qu: traffic volume?
100 mn url generated per day

qu: what characters are allowed?
can be alphanumeric

qu: can shorten URL be deleted or updated?
for simplicity, lets assume - not deleted or updated

### Requirements and back of envelope estimation

***Functional requirements:***

URL shortening: given URL - return much shorter URL
URL redirecting given shorter URL = redirect to original URL
high availability, scalability, and fault tolerance.

***back of the envelope estimation***

write operation = 100 mn url generate per day
- write per second = 100 mn / 24/ 3600
- read operation: assume read is to write = 1160*10 = 11600
- assume service run for 10 years = 365 billion * 100 bytes = 36.5 TB

## Step 2: Propose high-level design and get buy-in

***API endpoints***

1. URL shortening - to create new URL, client send POST request - with parameter: original long URL
   - POST api/v1/data/shorten
   - request shortURL

2. GET api/v1/shortURL
   return longURL for HTTP redirection

***URL redirecting***

- enter tiny url into browser
- server receive tiny url request - change short URL to long URL with 301 redirect.

NOTE: <u>301 redirect</u> - shows requested URL is "permanently" moved to long URL.

- since permanently redirected - browser cache response - subsequent request for same URL not sent to URL shortening service, instead directly redirected to long URL server directly.

- if want to reduce server load, 301 redirect makes sense only the first request of same URL is sent to URL shortening servers.

<u>302 redirect</u> - url temporarily moved to long url - subsequent request for same URL will be sent to URL shortening service first - then are redirected to long URL server.

- if analytics is important - it is a better choice as it track click rate and source of click more easily.

-most intuitive way to implement URL redirect is to use hash table - Get longURL:longURL = hashTable.get(shortURL)

-once get longURL - perform URL redirect.

URL shortening:

- each longURl must be hashed to one hashValue
- each hashValue can be mapped back to longURL

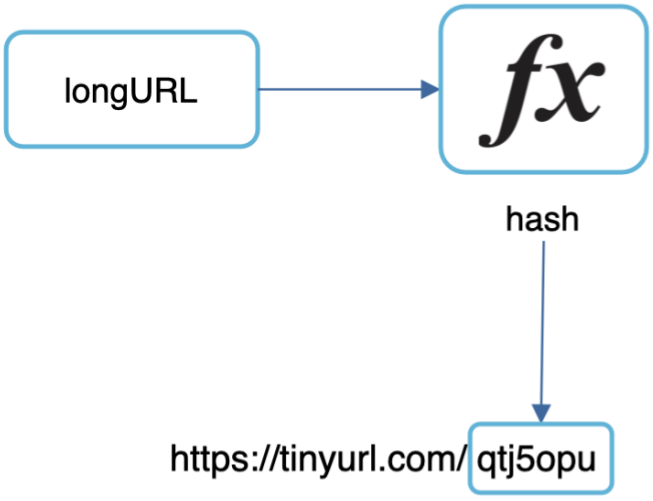


Figure 3

## Step 3: Design deep dive

***Data model***

- in HLD, everything is stored in hash table (not feasible for real world)
- better option - store <shortURL, longURL> mapping in relational database
- hash function used to hash long URL to short URL called hashValue. Length = 10 + 26 + 26 = 62 possible characters - possible combination = 62^n >= 365 billion
- implement hash function that hash long URL to 7 character string.
- some well known function. = CRC32, MD5, or SHA-1
- collect first 7 characters of hash value –

*but this method can lead to hash collision*

- <u>**solution**</u> - recursively append new predefined string until no more collision is discovered. -> this will eliminate collision but expensive query the database to check if shortURL exist for every request.
- bloom filter - space efficient probabilistic technique to test if element is member of set.

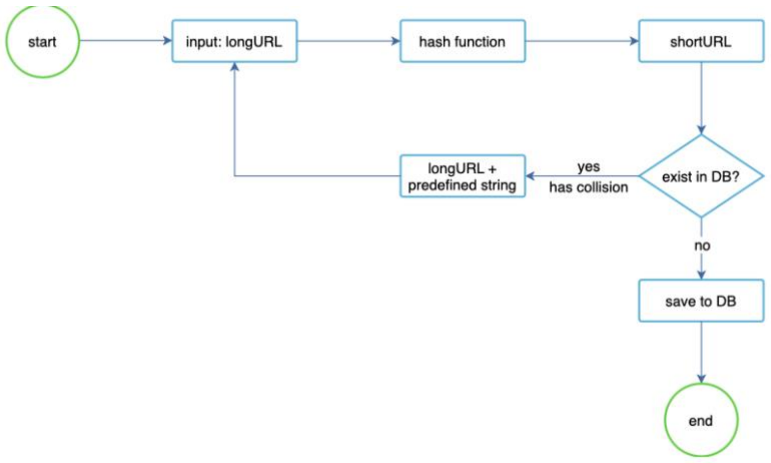


Figure 5

**Base 62 conversion:** convert the number into its base 62 format

| Hash + collision resolution | Base 62 conversion |
|---|---|
| Fix short url length | Short url length is not fixed. It goes up with ID |
| Does not need a unique ID generator | Option depends on unique ID generator |
| Collision is possible and need to resolve | Collision is not possible because unique ID |
| Not possible to figure out next available short URL because doesn't depend on ID | Easy to figure out what is next available short URL if id increment by 1 for new entry - can be a security concern. |

***URL shortening deep dive***

1. input = longURL
2. System check if longURL is in db
3. If present - fetch shortURL from sb and return to client
4. If not - generate new unique ID (pk) by ID generator - 5. convert ID to shortURL with base 62 conversion
5. create new db row with ID, shortURL, and longURl

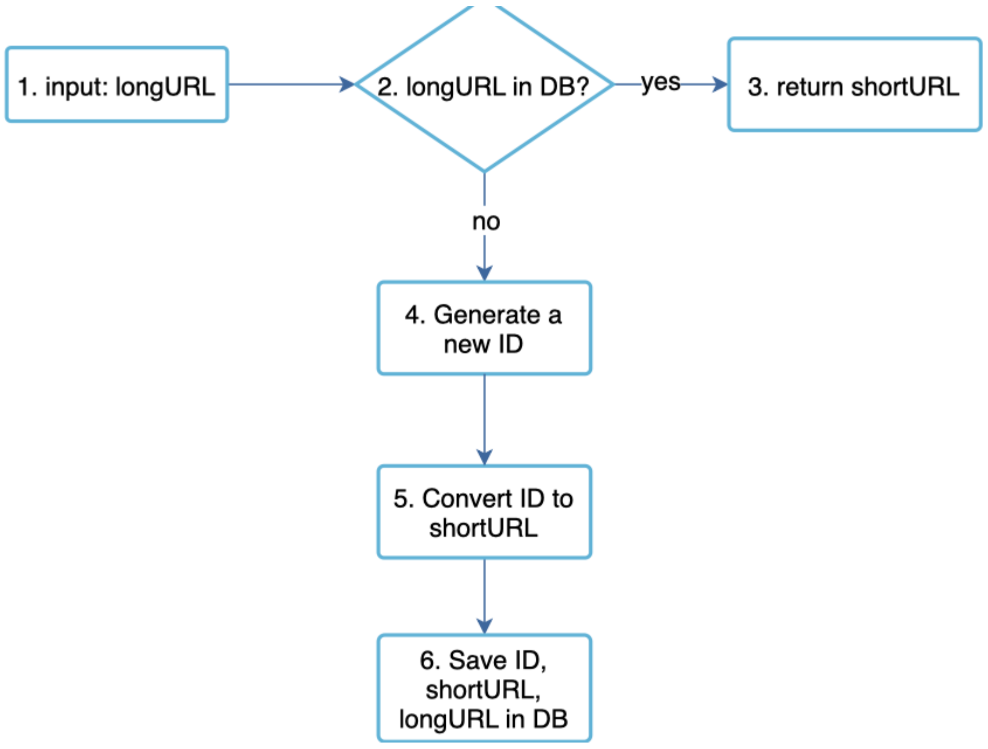
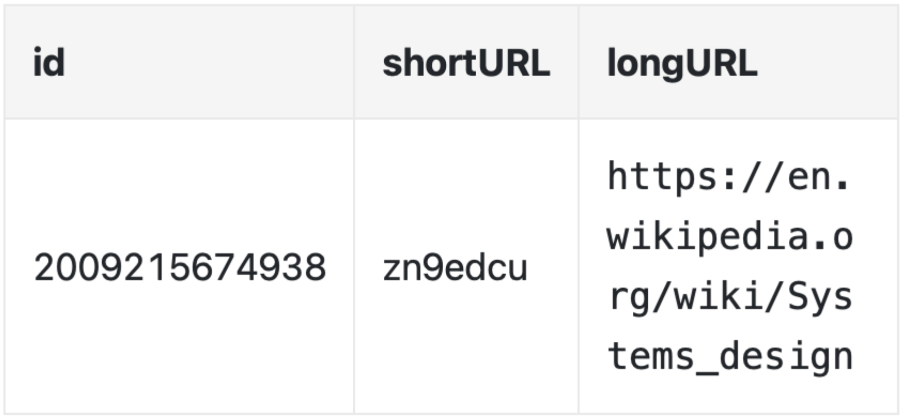


Figure 7



| id | shortURL | longURL |
|---|---|---|
| 2009215674938 | zn9edcu | `https://en.wikipedia.org/wiki/Systems_design` |

**URL redirect deep dive**

1. User click short url
2. Load balancer forward request to web server
3. If short url in cache - return long url directly
5. If not - fetch long from db - if not in db - user likely enter invalid short url
6. Longer return to user

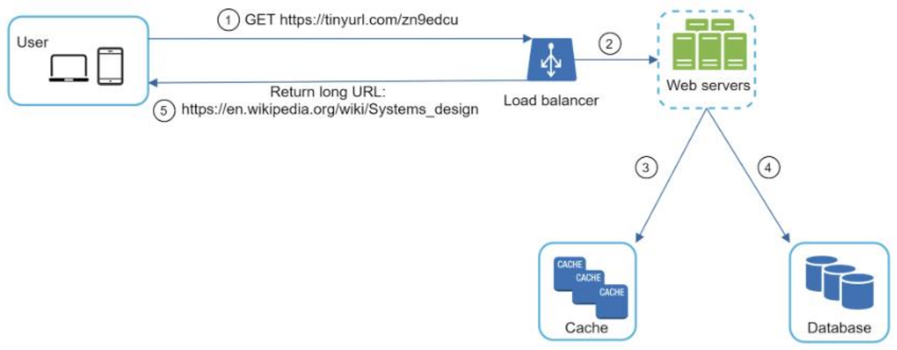

User
1 GET https://tinyurl.com/zn9edcu
Return long URL:
5 https://en.wikipedia.org/wiki/Systems_design
Load balancer
2
Web servers
3
4
CACHE
CACHE
CACHE
Cache
Database

Figure 8

**Wrap up**

1. Rate limiter - filter out request based on IP address or other filtering rules
2. Web server scaling - since web tier = stateless - easy to scale by add/remove server
3. Db scale = db replication and sharding
4. Analytics = integration result - how many click on link? When do they click link?
5. Availability, consistency and reliability