TinyURL

Saturday, 1 November 2025 10:43 AM

Tiny URL

bit.ly / 6-8
goo.gl / 6-8 letters
→ long url

Shortens long url to small 6-8 letter urls

Why Tiny URL is needed?

→ character limits in posting content.
→ cosmetic effects.

Functional Requirements

→ Given a URL, service should be able to generate a short and unique short url.
→ Given the short url, the user should be redirected to the original url.
→ Users should be able to choose a custom alias ✓
→ Links should be expired after sometime. User should be able to choose expiry time. ✓

Disadvantages

url → https://www.getsdeready.com/------
short → bit.ly/x12zB

→ not able to decide if link is secure.
→ brand value

bit.ly/GTSDR

goo.gl

Non-functional requirements → CAP, Latency, Reliability, Durability | Randomness

→ Availability

→ Latency → System should have very low latency.

→ short urls should not be predictable

Extended requirement / PI

→ Analytics
↳ metrics on redirections happening on a url.

MVP → Minimum viable product

Capacity Estimations

Assume traffic

→ TPS
→ Storage
→ Bandwidth
→ Memory

Assumptions: we will have 500M url shortenings per month

Application will have 100:1 Read/write ratio.

POST → 500M /month
GET → 500M × 100 /month

TPS :- $\frac{500 \times 10^6}{(30 \times 24 \times 60 \times 60)}$ = ~200 URL /s] write TPS
days, hours, min, sec.

Read TPS ⇒ write TPS × 100
200 × 100 = 20,000 URL /s

Storage :- Let's assume storage for 5 years.
500M / month × 12 months × 5 years.
500M × 12 × 5
$500 \times 10^6 \times 12 \times 5$ = 30B urls

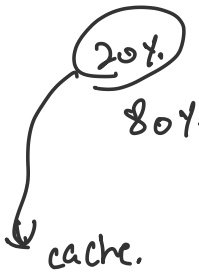
Ball park estimate :-
↳ Rough level estimate
storage for one url ⇒ 500 bytes.
30B urls ⇒ $30 \times 10^9 \times 500$
$\Rightarrow 15 \times 10^3 \times 10^9$
$\Rightarrow 15 \times 10^{12} \Rightarrow$ 15 TB

Bandwidth :-
No. of request × size of request.
• write BW per second :-
$200 \times 500 = 10 \times 10^4 = 100$ kBps
read BW
write BW × 100 = 100k × 100
= 10 MBps

Memory estimate :- Cache

we will be caching the hot URLs

Pareto's principle :- 80-20 rule



read traffic ≈ 20k /s

reads / day = 20k × 3600 × 24

= 1.7 bn

Space for caching 20% of 1.7 bn

0.2 × 1.7 × 500 bytes (size of one request)

≈ 170 GB

Since there are many duplicate requests, the actual caching space will be less than 170 GB.

Rough

write TPS ⇒ 200 /s

⇒ 200 × 3600 × 24

= 17280000

≈ 17M

cache ⇒ 17M × 500

⇒ 85 000 ⇒ 8.5 GB

avg expiry
↳ 15 days 8.5 × 15 ≈ 127 GB

System APIs

→ POST create URL
- → user_id
- → auth_token / auth_key
- → original_url
- → custom alias ⇒ optional
- → expiry ⇒ optional (default value)

returns → (string) → short link

→ GET get URL (short link)

return → original url

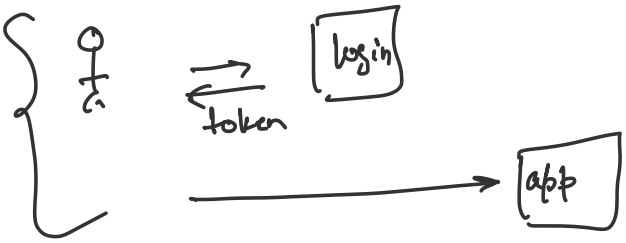
(HTTP 302) → Browser redirects to the url

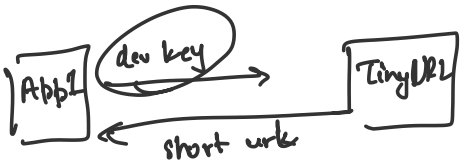
Admin
→ DELETE delete URL
- → short link
- → auth_token

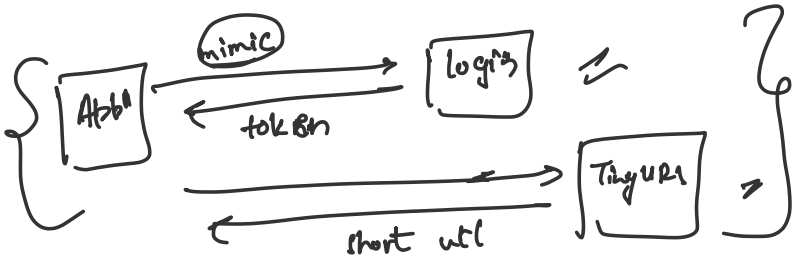
Auth tokens :-

→ ① Token → can be sent as a header
- → normally refreshed at login
- → used when user has to interact at app,

(2) Devkeys → - can be sent as header or body
→ are generated only once at account creation
→ are used when integrating with backend APIs







Database design

(i) 30B records
(ii) Style record size is very small (< 1k)
(iii) short url → original url (GET)
(iv) 100:1 read write ratio (Service is read heavy)

URL
PK : shortlink (20)
original URL (2000)

User
PK : userID (16)
name (50)

creation Time (16)
Expiration Time (16)
→ created By (16) FK
status (15)

email (200)
contact (20)
creation Date (16)
last login At (16)
hashed password. (3

SQL vs NoSQL

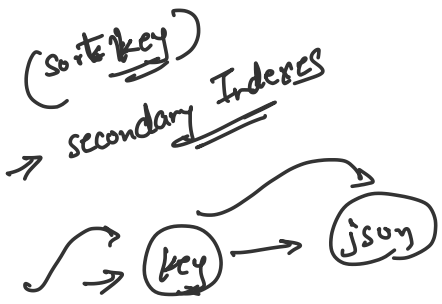
No SQL is better
↳ key-value based NoSQL

SQL uses Indexes for structuring Primary keys
↳ use a (B-Trees) (BST)
↓
has a best case time complexity of OClogN

→ For a key-value based NoSQL
↳ Time Complexity will be O(1)

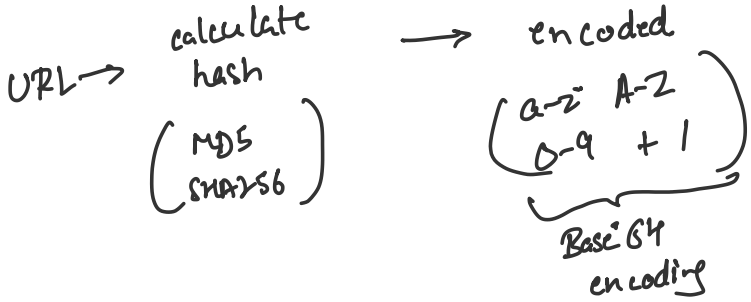
→ For normal functions, no joins are reqd blw the two tables. Hence relational DB is not mandatory to use.

Examples DynamoDB, Cassandra



H Algorithm

Cas Encoding original URL



128/6

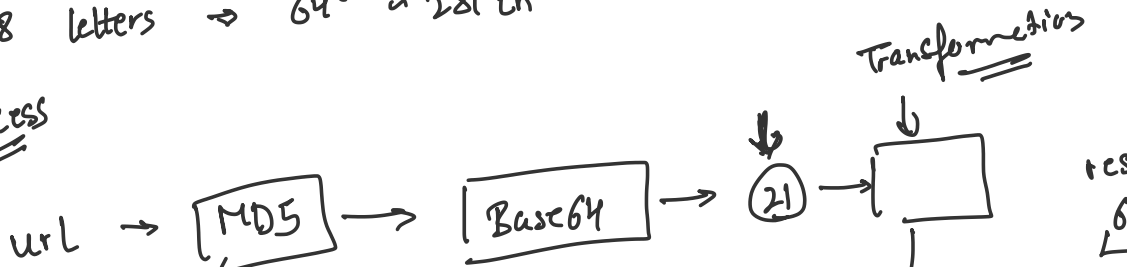
original url → MD5 → 128-bit hash value → Base 64
- 6 bits → 1 character
- 21 characters

64 characters
a-z A-Z
0-9 + /

6 letters → 64^6 ≈ 68.7 bn

8 letters → 64^8 ≈ 281 bn

Process



lose 13-15

duplication

Problems

→ same short link will be generated for same

→ if the same url is encoded, it will be treated as a different URL.

= ? & → %3F

url → encoded

↓ ↓

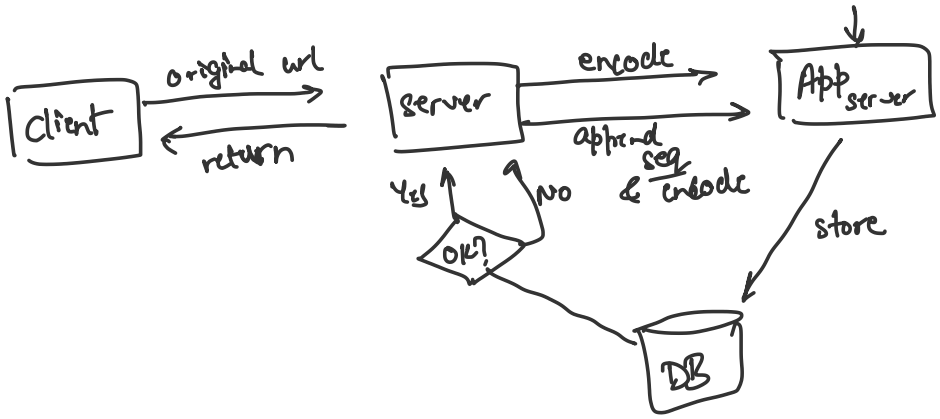
MD5 MD5

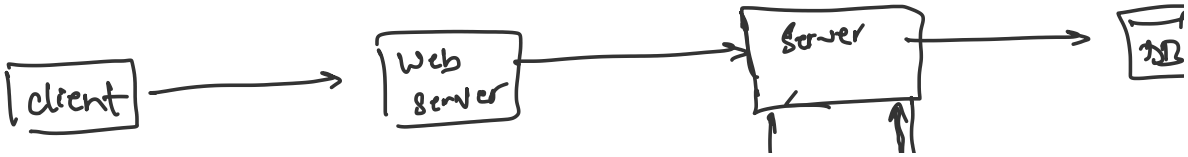
workarounds

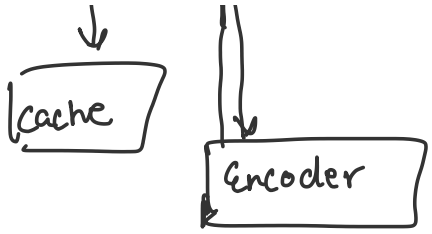
① → append an increasing sequence to original url before calculating hash.

→ [url → append seq → hash → encode → ....

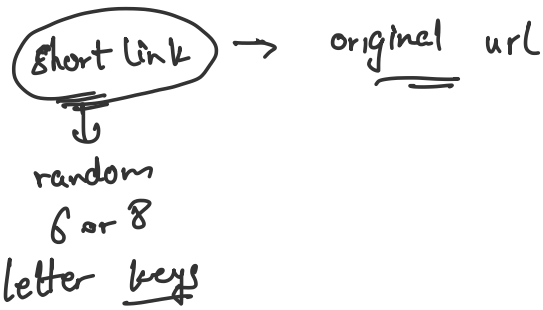
② → append user-id to the original







(b)



Generating keys offline

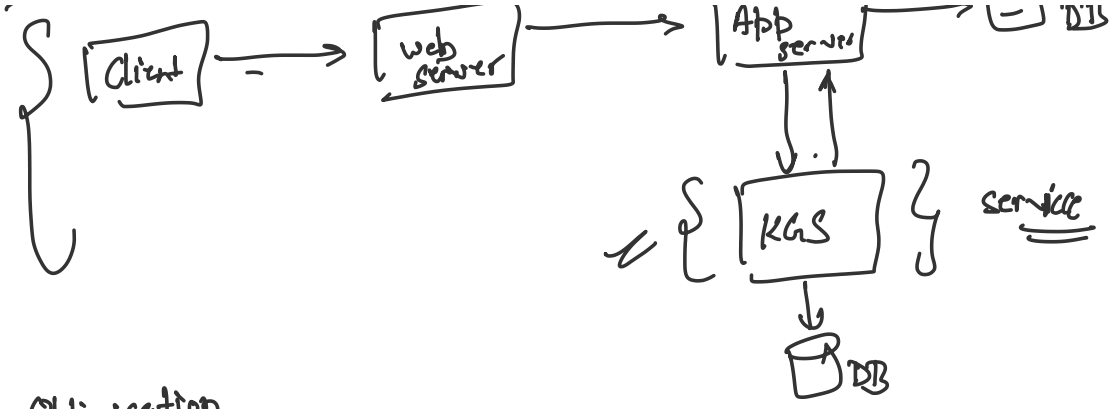
In this approach, we create random 6 or 8 letter keys.
when shortening, we can just assign one key as the short link for the url.

Considering, we are using Base 64 (a-z A-Z 0-9
using 6 letters, we have 68.7 bn

Key Generation service

↳ generate random 6- letter short key

Flow



Optimisation

App server can get a batch of short keys inst
of one key.

→ KGS can mark a key as used in its DB t
prevent providing duplicate keys.

Q. what if KGS fails?

→ we can run KGS as a Primary-secondary (Master-slave) apt

Q. We are loading 100 keys at once. What main server fails before utilising all the keys??

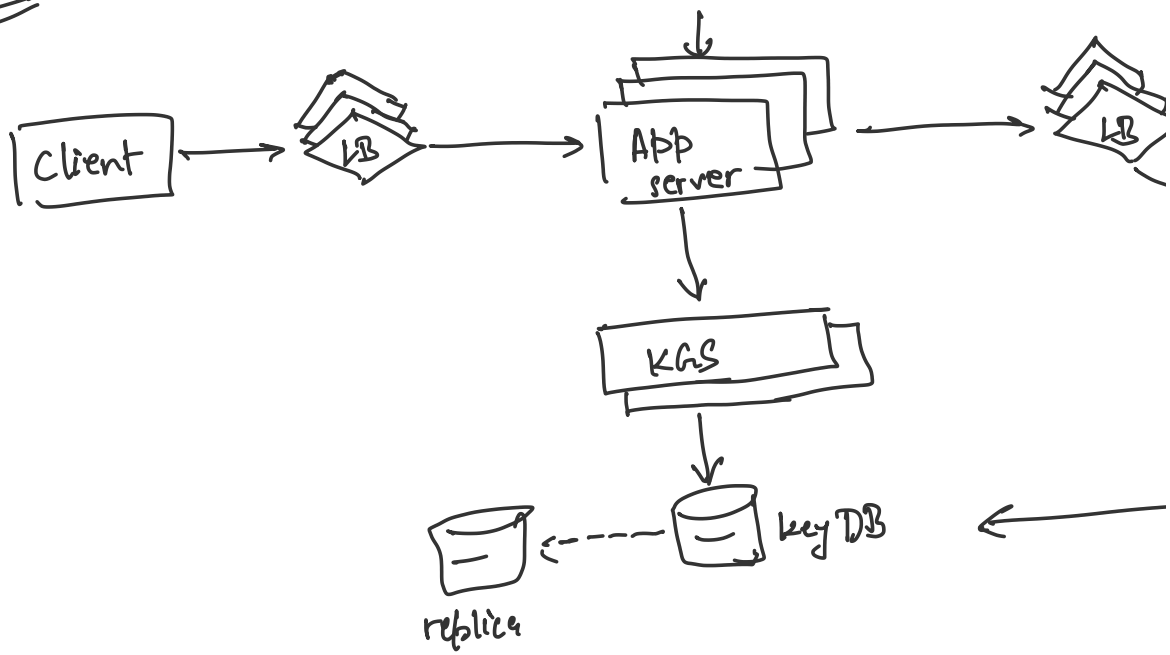
→ Keys will be wasted in such a scenario c
okay considering we have 68.7 bn (6-letter)
281 tn (8-letters)

→ Another approach is to write a job to free u
unused keys in key DB by matching it with
DB.

Q. Storage reqd for key DB ?

→ 6 characters * 68.7 bn ⇒ 412 GB

HLD



Data partitioning
Cache
Cleanup
Distribution

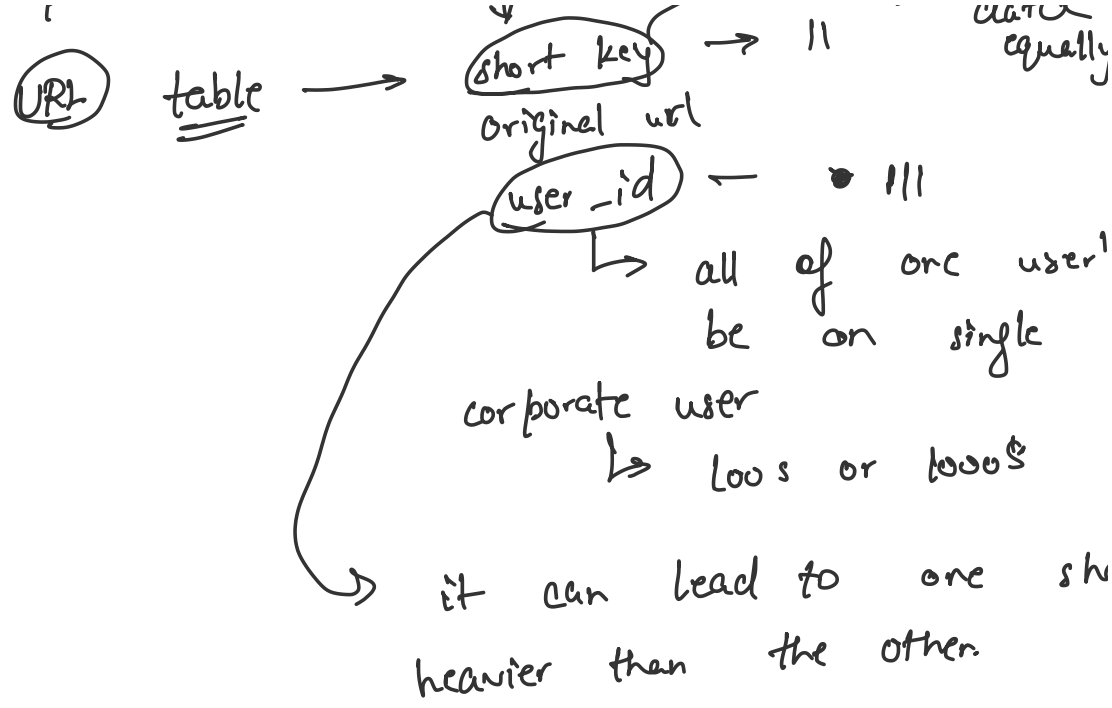
Data Partitioning

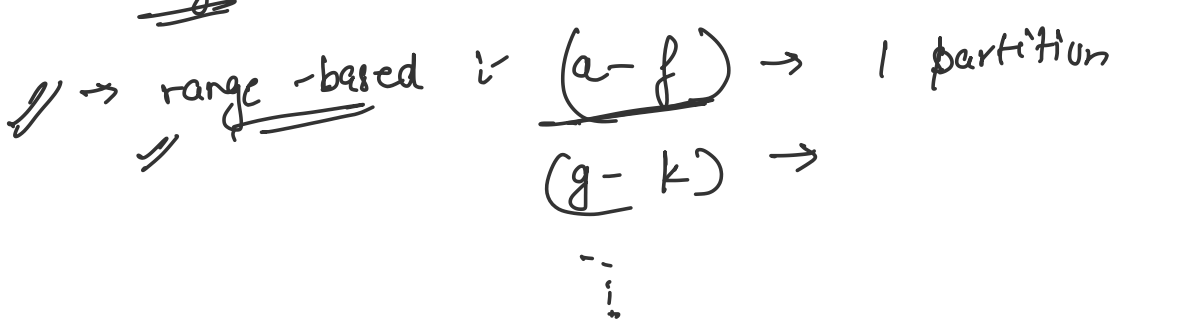
15 TB → cannot be stored in one ser

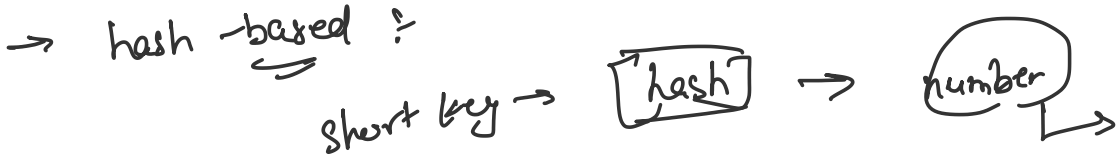
we need to divide our data in m

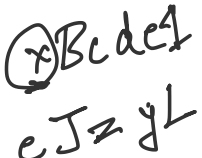
partitions

1.









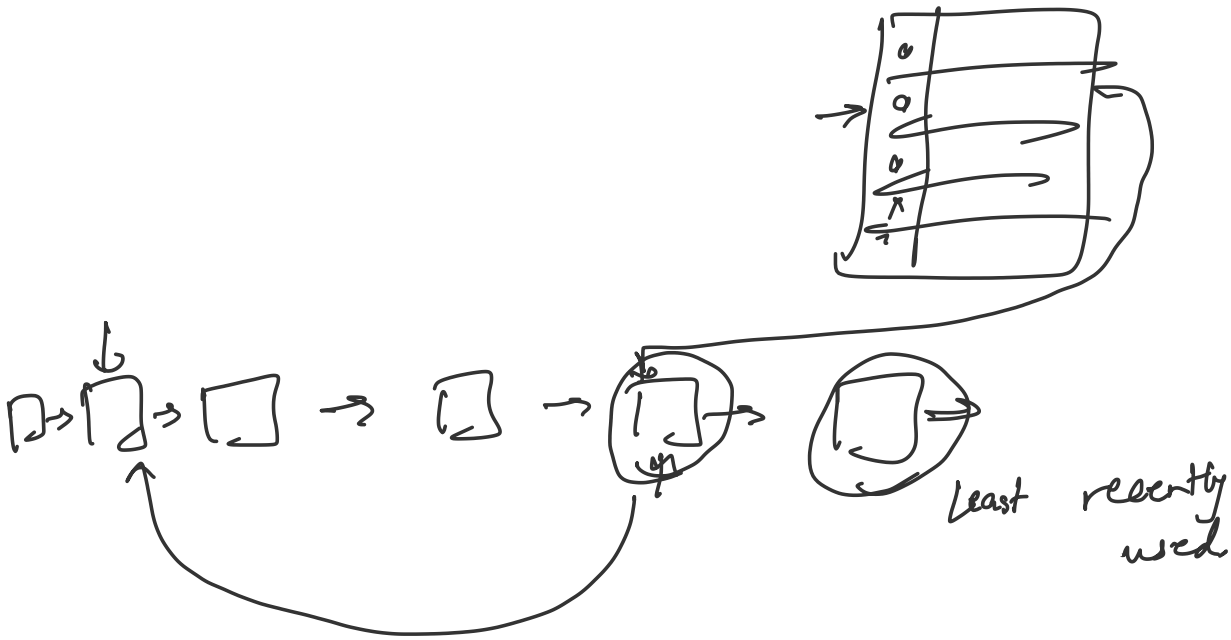
Cache

- Cache Eviction :-
  - Hot urls are to be stored

- Cache
  - urls tend to be used less with time.

⇒ LRU can be a good approach.

⇒ expiry → on expiry, url can be removed cache.



Cleanup

- cleanup or archival in Database.
  - main table
  - key DB

Q. Should url entries be deleted after ex

→ we can mark it archived, not comp
delete.

Q When can we delete the date?
→ we can define some period post expir
data can be deleted

DB → object (S3) → AWS glacier → log files. (HDD)

create a different lighter service runn
regular interval. (off-business hours)