System Design

3 Principle :->

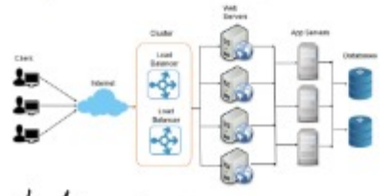
① Single responsibility → Each component present in S.D should have single job to perform.

② No single point of failure → The system should not have any component whose failure will result in the failure of entire System.

③ No bottleneck principle → Since we will design scalable System should not have performance bottleneck. So ideally we scale horizontally to handell large amount of processing.

Here L.B, webserver, appserver Database have singe Responsibility

Here if one webserver gd failed then still system will work



traffic will be distributed to avoid bottleneck.

5 Step Guide for System Design

① Requirement Analysis (functional + Non functional Requirement) 2

② Api Design (name of Api + parameter) + return (with datatype)

③ Design Data Model (Define Table and Columns) ④

High level Design (Component and arrow flow, servers, db, L.B)

⑤ Scale the Design (to handell Large processing) 0

optional ⑥ Back-of-the-envelope Calculation

Design Tiny URL

① Requirement Analysis ③ Data Model design ⑤ Scale

② Api design ④ HLD ⑥ envelop calculation

① Function Requirement :→
→ Given URL we should get unique random short URL → CreateShortLink(originalLink, userName) ← Storing
→ Given URL should able to redirected to original URL → getUrl(shortLink) → String
→ Users should able to create Custome Link. customeURL ← variable that will be provided by user
→ Users should able to choose expiration date. deleteURL(shortLink, username) → void

Non functional Requirement :→
→ Highly available → Scalable
→ Minimum latency → users should able to know how many times link accessed

② API Design :→

① long CreateUser(email, username) to create user that take email and username.
② string CreateShortLink(originalURL = URL, CustomURL = None, ExpirationDate = None, developerId = None)
to creat shortlink using originalURL
③ String getOriginalLink(ShortLink) → to get the original link from short link.
④ void DeleteURL(shortlink, username) → to delete already created short link
⑤ int getHitCount(Shortlink, username) → get the no's of access of the short link.

③ Data Model :→

← Structured

User : userId(PK), username, email, creationdate

ShortURL : originalURL, userId, shortLink(PK), expirationDate, hitCount

← Structured

operation (insert, fetch)

Which kind of DB use (DB selection)

→ SQL (ACID, structured) slow if data is huge

→ Nosql (highly available, large amount of data can be stored)
- Column (structured) ✓ (limited Query) ✓
- Document DB (unstructured) ✗ (more query) ✗

So we will use Clem DB [Cassandra]
HBase
MariaDB

④ HLD ; The main the problem to convert long URL to short url

What are the character involved in short url

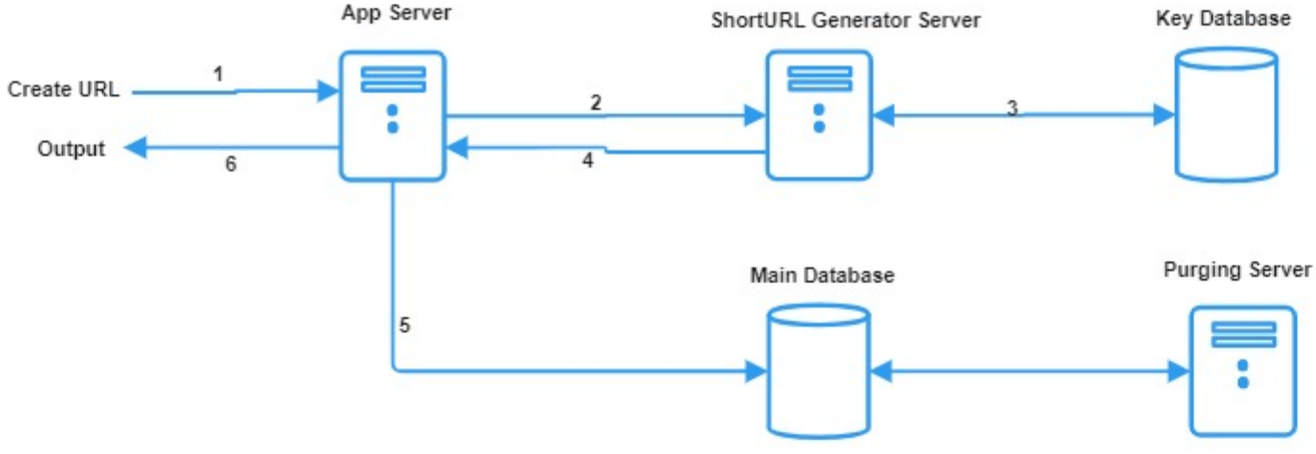
→ character Encoding Base 64 [A-Z][a-z][0-9][+/] =
26 26 10 2
64

→ way to generate short URL.

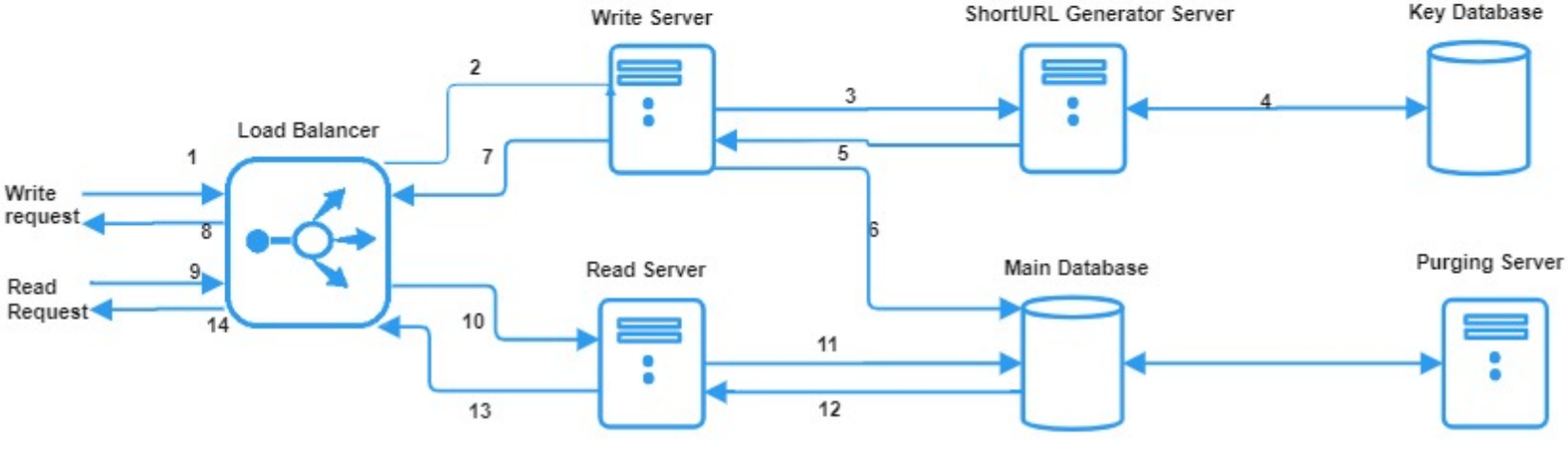
① generate random string using Base64 and consider it as a short URL.
There could be too much collision X

② to avoid collision we can use incremental X [a, b, i, n, i, z, aa] but we need randomness here it predictible

③ Create hash of (URL + key) using MD5, SHA256 ✓ as it fulfill functional requirement
key: random or incremental
MD5: older
SHA256: Minimum collision



System Design



Single Responsibility System Design

Back-of-the-envelope calculations

we have 100 write/second and each request is 10KB

Perday → 100 * 60 * 60 * 24 = 36000 x 24 = 360K x 24 ⇒ 400K x 20 ⇒ 8000K B write/sec

Peryear ⇒ 8000K x 365 ≈ 8000K x 400 = 3200000K ⇒ 3200M ⇒ 3.2B

Disk:

= Peryear ⇒ 3.2B x 10KB ⇒ 32B KB ⇒ 32 TB

Write x (7/10) = 32 GB ⇒ x = (32 x 10)/7 ⇒ 50 TB

Read

Consider read is 100 times from write

So read ⇒ 100 x 100 read/s

⇒ 10K r/s ⇒ 16 x 10 MB/s ⇒ 100 MB/s

if our one server able to handle 10 MB/s then (100 MB/s)/10 = 10 server for Read

= 1 for write

Cache :→ let say 20% is responsible for 80% of data

80GB x 20/100 ⇒ 16 GB cache needed

Database Selection based on application

MD5, functionality
SHA256 functionality