System Design 1

Designing a URL Shortening service

What to be designed?

1. URL shortening is used to create shorter aliases for long URLs. Shorter URLs take up less space and can save mistakes through typos in longer URLs.
2. Play around with shortening of URLs on tinyurl.com to understand the use case of shortened URLs.
3. URL shortening is used for optimizing links across devices, tracking individual links to analyze audience and campaign performance, and hiding affiliated original URLs.

Feature Requirements

Primary Features:

1. Generate a shorter and unique alias for a long URL. Shortened URL should be short enough to be easily copied and pasted into applications. Users should able to pick a custom short link for their URL.
2. When users access a short link, our service should redirect them to the original link.
3. Shortened URL Links will expire after a standard default timespan. Users should be able to specify the expiration time.

Secondary Requirements:

1. URL redirection should happen in real-time with minimal latency.
2. System should be highly available. If our service is down, all the URL redirections should not be failing.
3. Security: Shortened URLs should not be predictable.

Extended Features: Analytics; e.g., how many times a redirection happened? Our service should also be accessible through REST APIs by other services.

TO DO : Follow the below framweork to design System. Mandatory upload of hand drawn photos of design.

Capacity Estimation and Constraints

What will your estimations of scale? As a system design student you should be able to make intelligent assumptions based on real life systems.

System APIs

Database Design

Basic System Design and Algorithm

Data Partitioning and Replication

Caching

Load Balancing

Handling Edge cases in given system

Functional Requirement

Non Functional Requirement

Capacity Estimation

Application is Read heavy system. Assuming there will be 100 reads per write to the application.

Traffic

1 short URL = 100 bytes.

1 Long URL = 200 Bytes

Over all there will be 300 Bytes to be processed.

Assume 100M new writes per month

This get us to (100 * 100) M read request.

QPS → Converting Month to Sec (Query Per Second) = 100M /(30*24*3600) == 150 url/sec

Read Request will be 10,000 M which will brings us to 15K URL/sec

Storage

Assuming we will have on average 200 + 100 Bytes of Short and long URLs. For each day we have 50K req.

Storage required for each month we 100M request with each storing 300Bytes

(300 bytes* 100M Users * 30 Days* 12 Month) Year * 5 Years = 15 TB storage.

Memory

Cache Memory to be stored with 80:20 rule. We can cache the 20% daily request to be cached which is frequently being used for better performance.

On Average we are getting 15K request per second. This will be 15K * 24 * 3600 = 2 Billion

20% of 2 Billion req = 0.2 * 2 * 500 Bytes = 140 GB.

Bandwidth

Application should be able to handle Incoming data and outgoing data seamlessly.

Incoming Data = 500 Bytes/URL and there are 200 New URL/Sec

There will be 500 * 200 is the data transfer happening per second == 100 KB/sec.

System API

There are 2 main APIs

CreateURL(api_key, userID, longURLString) : String shortURL

DeletURL(userID, shortURL) : void

Data Base Design

USER and URL table

USER Table will hold all the profile details.

- USER_ID
- USER_NAME
- PROFILE_NAME
- DOB

URL Table –

- API_KEY_ID
- USER_ID
- URL_STRING
- SHORT_URL
- EXIRATION_DATE

System Design

System has to be highly available – Maintain Load Balancer at each level

i.e Between Client Request to Server,

Application Server to DB Server

For Better performance in accessing frequently visited URL maintain a cache server which hold 20% of daily frequently accessed request.

Eviction Policy

Remove the frequently least accessed URLs every 24 Hours through a manual process.

Replication Strategy

As the application is read heavy there must be a read replication which gets the data from DB server