## **Requirements**

**Functional Requirements:**

1. Take a long URL (For example <https://www.youtube.com/watch?v=9dPDLa5PDTI&list=PLDnNz0AN_2ca8Yei_5JM8a54gDqBulHn7&index=4>) and convert it to short URL (<https://tinyurl.com/y7rgks9l>)
2. Redirect a given TinyURL to the associated long URL.
3. Create a custom alias for the URL.
4. Default or custom expiration date (which can be set by the user).
5. The ability of the user to delete URLs.

**Non Functional Requirements:**

1. It should be highly available.
2. Redirection as well as URL generation must happen with low latency.
3. It should be reliable.
4. It must be secure.

## **Capacity Estimation**

**Assumptions:**

|  |  |
| --- | --- |
| Number of users per month | 100 million |
| Number of URL’s generated by one user per month | 10 URL/s month |
| The ratio of URL reads/ writes | 50: 1 (Read heavy system) |
| Default Expiration time | 2 years from the created date |
| Long URL Size | 300 bytes |

**Traffic:**   
  
Write queries - (100 million \* 10 (URLs generated per user per month) ) / (24 \* 60 \* 60)   
~ **Approx ( 400 queries/second)**

Read queries - 50 \* write queries = 50 \* 400 qps = **20,000 queries / second**

**Disk Space:**

100 million \* 10 URL’s \* 12 months \* 2 years \* 300 bytes = 7200000000000 bytes ~   
= **7.2 TB (terabytes)**

**Bandwidth:**

Incoming Bandwidth (writes) = Number of users per month \* Number of URL’s generated by one user per month to seconds \* URL size ~= 100 million \* 10 URLs \* 300 ~= 120,000 bytes / second ~= **0.1 MB / second**

Outgoing Bandwidth (Reads) = 50 \* 0.1 MB / second = **5 MB / second**

**Memory:**

20% of the total queries will be in the cache.

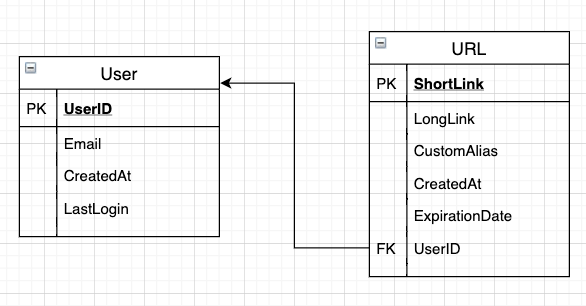
The cache will be periodically updated per day. So we have 400 queries per second. Each query is of size 300 bytes. So,

Cache size ~= 0.2 \* 400 \* 300 \* 24 \* 60 \* 60 = **10.37 GB** (We can keep additional 1.4X of this as buffer to account for the hot days)

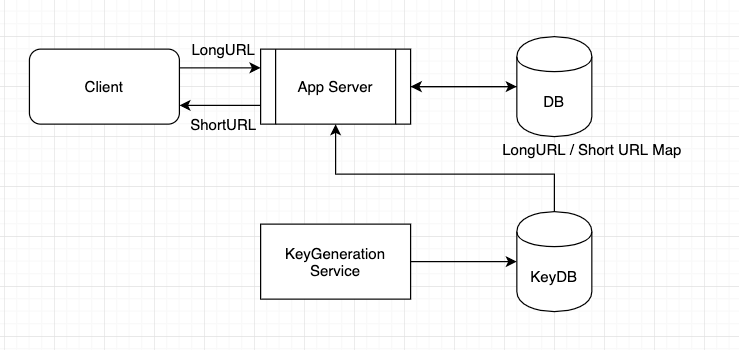
## **API’s**

1. createUrl(api\_devkey, longUrl, customAlias?, customExpirationDate?)
2. deleteUrl(api\_devkey, shortUrl)
3. redirectUrl(shortUrl)

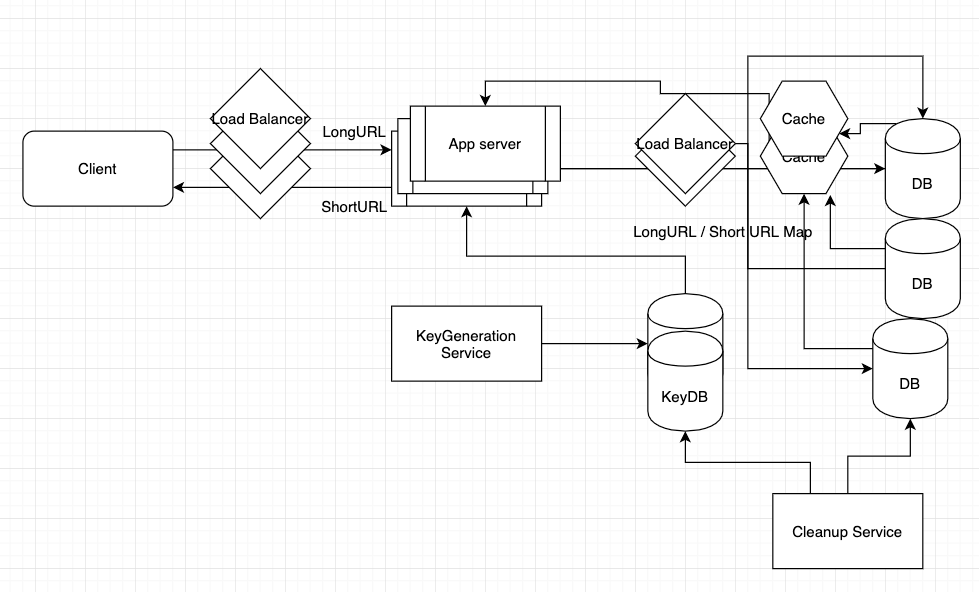
## **Data Modelling**

****

## **High-Level Design**



**Component Design**

****

KeyGeneration Service: This will help generate the random key and stores it in the KeyDB and also in the cache for fast access.   
  
Load balancers use the Weighted Round robin algorithm to distribute the load to the DB’s and to the cache.

Database stores the LongURL to short URL mapping which helps with redirection.

We have multiple DB’s, Key DB’s and cache for redundancy so that there is not a single point of failure.

Cleanup service runs periodically when the servers are not much active to delete expired URL records from the DB.

When a key from the KeyDB is used, that key is moved to the Used table so that next time, when the key is generated next time, we can ensure that it is unique.

## Telemetry and Security

We can keep track of hot URLs (keeping a count of how often every URL is accessed) and hot users (which the user actively generates or redirects to URLs). We can also keep track of how the system is used during hot days like Black Friday or other seasonal events to understand how the system works under stress. These data can be used to further improve capacity on certain days and scale up and down depending on the needs.

We can access specifiers to the URLs (Private, Public). In the event that the URL is private, the user can specify which other users, other than himself, have access to it with the help of their friend's UserIDs. This will improve the privacy and security of the system. We can also have a service to check if the long URL has “https”, at the start. If not we can mark the URL as unsafe and warn the user who tries to redirect.   
  
Public means that anyone can access and redirect those URLs.