**Level-01 : Discussion of TinyURL like URL Shortening service : Introduction

Purpose:
This is the service prepared for handling the short url against the long url.
First step is to understand the requirement and goal of the system.

You should always clarify the requirement in very beginning of your discussion. It helps to start building the ladder between you and your audience.

So, after discussion you will finalize the requirement and goal of the system. Let's put it down in 03 different category (FR, NFR, Extended requirement)

Requirement Gathering and Analysis

Functional Requirement

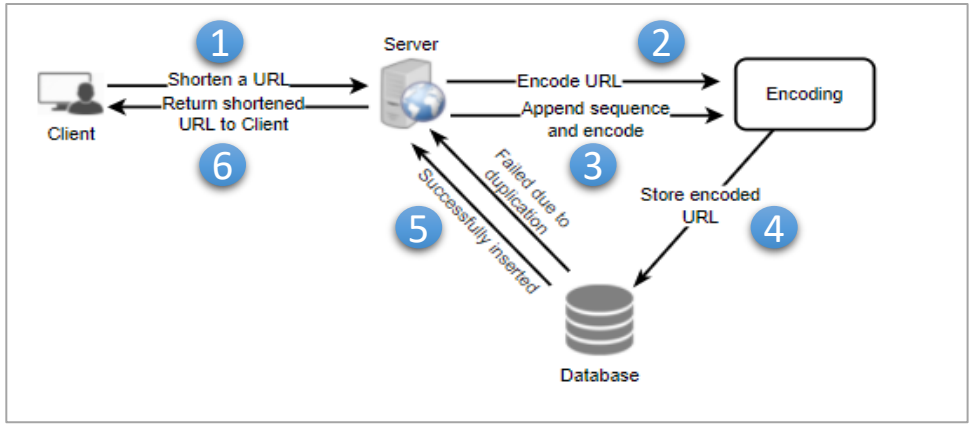
- ✓ System must generate short url for given long url and vice-versa
- ✓ System must be capable to redirect for short url to the original url
- ✓ Custom name possible for your short url as per user choice
- ✓ Link must be attached with validity timestamp(how long valid)

Non-Functional Requirement

- **High Availability –** Single most important NFR of system is availability, service should be available all the time.
- **Low Latency** – in re-direction of url (Short url -> Original url)
- **Security** - This generated link should be safe & secure

Extended Requirement

- **Analytics –** Build analytics on usage of system by user to attract and enhance their experience
- **Service** should expose as rest webservice api



Capacity Estimation & Constraints

- **Read heavy system:** If you analyse the requirement, you will able to understand that in this system, creating a short url is one time task but using that short url many-many times possible. So, this refers our assumption of read heavy system is valid.

As you mention the read heavy, but how to quantify this? And so let's assume that our system is in 100:1 read vs write ratio.

- **Traffic Estimate**:

For Ex,
1. 500 Million active request for URL shortening per month
2. So, using the R:W ratio, we are expecting the 100 * 500 Million read per month , which is approximately **50 Billion read request per month**
3. If I have to calculate the QPS (Query Per Second) for the system

It could be calculated like **50 Billion / (30 * 24 * 60 * 60) Equivalent to(=) 20 K per second**

- **Storage Estimates**:

You assume that whatever request coming to our system, it is getting created and also getting stored into persistent system, irrespective of timestamp for live is limited for shortened url. Meaning for system this data need to be persistent forever but for user to the system might have, some limited time for this shortened url only.

For Storage purpose let's assume that 500 Million URL getting created in a month time.
Also, assume that being an architect you have to gathered information that you might need planning for at least 5 years ahead storage requirement.

Than, lets calculate the storage required for the 5 years according to the calculation assumption for a month

It is approximately : 500 Million per month * 12 * 5 = 30 Billion Record (Storage Object)

One more assumption you have to make here is that let's assume that each storage object roughly required 500 bytes
So approximately our Storage Estimates is around, (500 * 30 Billion) = 15000 * 1000 * Million bytes = 15000 * 1000 * 10^6 bytes = 15000 * 1000 Mega Bytes = 15000 Giga Bytes = **15 Tera Bytes**

** Level 01 – Professional : Expect discussion at justifying your all decision with why?

Discussion of TinyURL like URL Shortening service – High Level System Estimation

➢ Bandwidth Estimates

For Write of New Record
As per our earlier assumption taken, 500 Millions shortening URL request received by our system.

If I deep dive and calculate for second here, it would be turned out to be approximately 200 new record per second bandwidth load on our network.

As per our earlier assumption for one record object size was 500 bytes, so if multiply this with new record per second, it turned out to be 500 * 200 = 10^5 bytes = 100 KB/s

For read requests,

Our assumption was that , read vs write ratio was 100,
So by this logic our read bandwidth requirement will be around 100 * 100KB/s = 10 MB/s

➢ Memory/Cache Estimates

Let's understand here the cache requirement. **Why you need it at first place?**

As an architect we realised that a good percentage of short URL are getting lots of request, because this URL seems to be some content from celebrity people.

So every time you will go to the database and fetch corresponding original URL wound's be a good idea and so CACHING of such record into CACHE(may be **Redis**) would be beneficial and performant.

Let's calculate the number now?
Meaning, how much memory planning is sufficient as per probabilistic calculation?
As per our observation it seems Pareto principle 20% of the URL is generating 80% traffic is the fact and so in that case you should focus to provide cache for that 20% to improve the overall experience of your system user.

As per our calculation, we observed earlier that 20K request per second is the futuristic load for our system.

So, 20 * 1000 * 60 * 60 * 24 per day requirement. It comes, 172 GB (approx. = .02 TB) per day

➢ Bandwidth Estimates

So, overall high level estimates for the system

Traffic Estimate	20 K / s
Storage Estimate for 05 Years(~)	15 TB
Bandwidth – Incoming	100 KB / s
Bandwidth - Outgoing	10 MB / s
Memory for Cache	172 GB per day

So, the above calculation is just futuristic and based on assumption. So, please safely assume that if the assumption changed, your other estimation will also got changed.

So, until this point you and your audience got locked over the system requirement about the features and NFR of the system. So always good idea to define the system api of your system at higher level to understand the broader system api with your audience.

Let's focus on System API now,
Primarily we need two api , one for creation and other for deletion.
So, for now let's stick to these two primary api's.

createURL(api_dev_key, original_url, custom_alias = none, user_name = none, expire_date = none)

deleteURL(api_dev_key, url_key)

Let's define more detailed point for these api's in next page…

System design : High Level - System api, Data Model

API for MFP

createURL(api_dev_key, original_url, custom_alias=None, user_name=None, expire_date=None)

Parameters

- ✓ Api_dev_key (string): this is the key assigned to registered account.
- ✓ original_url (string) : This is the original URL for which short URL is getting created
- ✓ custom_alias (string) : optional custom key for the url
- ✓ user_name(string) : optional username to be used in encoding
- ✓ Expire_date(string) : optional expiration date for the shortening url (how long it will be maintained in cache

Return (string):
If successful than return the short url otherwise return the error code

deleteURL(api_dev_key, url_key)
Parameters
If successful return deleted shorturl otherwise return error code

How to prevent the API abuse in your system. Or otherwise, **do you see the reverse proxy or security handling required** before it hits your api server?

DB Design

Database design
@Note : you must define the data model in early stage of your design discussion, because if you go late and if you miss something than there is no chance to go back and correct it. So, it is always advisable to discuss the data schema with audience as far as you can see for now and get some level of consent with your audience up to initial agreement.
Logical steps for getting kind of data required into your system.
Observe the required feature and NFR into the system

1. We have to store billion of records
2. Each record is small in size for managing shortURL vs origionalURL
3. There is no relationship in records except the user who is creating shortURL
4. This system observed as read heavy operation system

So, from above observation, there are two Table Schema I can visualize here

URL	
PK	**Hash: varchar(16)**
	OriginalURL: varchar
	CreationDate: datetime
	ExpirationDate: datetime
	UserID: int

User	
PK	**UserID: int**
	Name: varchar
	Email: varchar
	CreationDate: datetime
	LastLogin: datetime

[NoSQL DB could be choice] By looking the above schema, I can sense here that relational database is not as much required, instead seems scale is driving factor here and so seems that some NoSQL like database will be considered more appropriate in this situation.

I can see some example of DB here, but most of them have similar feature so choosing one of them is ok. For ex: Cassandra, DynamoDB, or Azure Cosmos DB, etc

System design : High Level - System design blocks 1/5

Basic System Design

Problem: How to generate short and unique key for a given URL.

Let's try to understand the exact requirement through one example
https://tinyurl.com/rxcsyr3r in this example last 8 character can be considered as the shortURL or Unique key expected from your system.

How to generate these 8 characters based unique key or shortURL?
There are multiple approach possible to generate these shortURL or unique key.
Let's discuss the Approaches one by one and their pros and cons…

Approach 01 : Encoding Actual URL

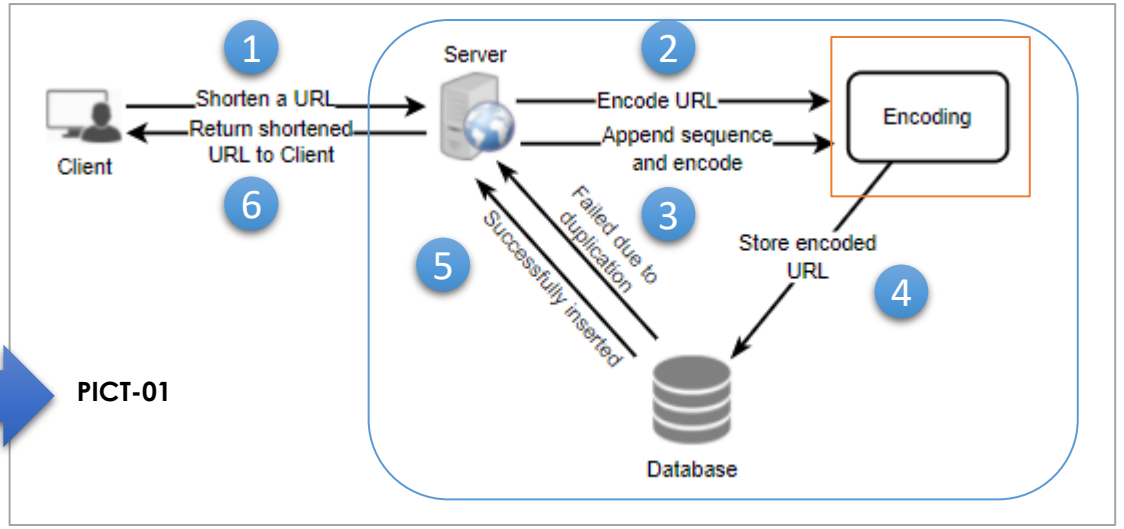
So Pict-01 is our initial basic design (**Approach 01**) where we have shown the basic flow, about how system is sequentially interacting for simple use-case like "User or client requested for the short url and in return server is returning the short url after the successful completion all intermediate steps"

Here in interaction diagram (PICT-01) showing the System have majorly 03 components
1. Server (API Server)
2. Encoding Component Server
3. Database (DB Server for Storage and persistency)

Steps to produce shortURL or unique key
1. Compute an unique hash using standard Hash function like MD5 or SHA256
2. Now next thing is to encode this hash value for display, and for this encoding could be **Base64** [A~Z, a ~ z, 0 ~ 9, '+','/']
3. Next thing is to understand that how big of this shortURL length would be suffice ?

To understand this let's calculate the unique value of key's possible with different length
So, according to the requirement you can choose the length of your shortURL. Please note that small url is easy to remember but also less number of unique key and that would get exhausted very fast as your system user grows.

Len = 6	64^6 = 69 billion
Len = 7	64^7 = 4 trillion
Len = 8	64^8 = 281 trillion



PICT-01

Let's assume that we are planning to select 6 length because it is solving our problem for now.

If we use MD5 (https://en.wikipedia.org/wiki/MD5) as our hash function, it will produce 128-bit hash value.

After base64 encoding, we'll get a string having more than 21 characters (since each base64 character encodes 6 bits { @note 2^6 = 64 so produce all these 64 character, 6 bits are sufficient}of the hash value).

Calculation is like 128 bit/6 bit >= 22 character (but need more than 21 character)

Now, since as per our decision, we assume that our string length of short url is only 6 in length.

So, then our problem here is how to choose 06 character out of 21 character?
So, let's assume that you also have such challenge, and you must handle this with your approach.

**Let's continue our discussion in next page…

System design : High Level - System design blocks 2/5

Contd from previous page
If I select first 06 character out of 21 character, what will be consequences might be.
So, think the scenario where you have two cases when there overall encoded 21 character might not be the same but their first 06 character are same. So, they are making duplicate claim but, they can't. This is your system problem. So, your decision to choosing first 06 character as a key won't be good option.

Here as an architect, you might look solution around, and solution might be picking some other character with some logical thought or steps from remaining character. Yes, so this might work. But it has still some problem, let's discuss that now.

What are the different issues with our solution?

1. If multiple users enter the same URL, they can get the same shortened URL, which is not acceptable
2. Suppose if only difference is in encoding value of two shortURL

So, these above problem could be solved if we do the following,

A) Append the unique increasing number along with every URL request
B) Append the userID along with every URL request

These approach will solve the problem at certain extreme, but it has also limitation. So, we can't have full proof solution. To achieve 100% accuracy, we only must write logic in such a way that you keep repeating some steps until find the unique key.

For that we must introduce some database where we can check the existing shortURL key generated until now and on finding new one that will be pushed to database for next duplicity check.

Since you introduced new component to check the duplicity so you might encounter here some issue while scaling the system in distributed case because database interaction is not that super performant.

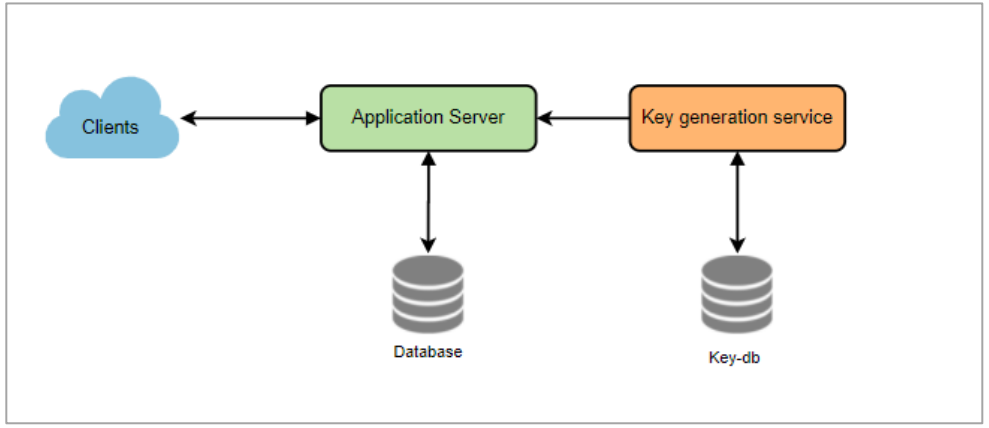
But those discussion , I am leaving now, and we will discuss the same again if time permits.

Approach 02 : Generating Key Offline

We can have a standalone Key Generation Service (KGS) that generates random six-letter strings beforehand and stores them in a database (let's call it key-DB).

Whenever we want to shorten a URL, we will take one of the already-generated keys and use it. This approach will make things quite simple and fast.

In this case, duplicity is not in the question because KGS(Key Generator Service) will make sure that all key inserted into key-DB will be unique.



System design : High Level - System design blocks 3/5

Can concurrency cause problems?
If there are multiple servers reading keys concurrently, we might get a scenario where two or more servers try to read the same key from the database. **How can we solve this concurrency problem?**

Step 01 : Create Two table , one for used key and one for Unused key. By doing this, first good thing happen in terms of two call possible on two different table at a tome. Initially everything is inside Unused key table and as soon as we shared key to some request, this key should be removed from Unused table and put into Used table.

Step 02 : In your architecture decision also think to put some key in cache to handle key allocation, which will improve your system performance. This will require steps if your latency have some NFR constraints.

Step 03: KGS seems Single Point of Failure right now, so I would suggest to consider some standby replica of KGS.

Since you must adopt offline Key generation so in this **case you should must plan the Key-DB size?**
I believe that with length 6 of key , we can produce 64^6 (using Base 64 Encoding) these many unique keys. Which is roughly 68.7 Billion

And so total size required will be 6*68.7 Billion bytes = 412 GB (roughly)

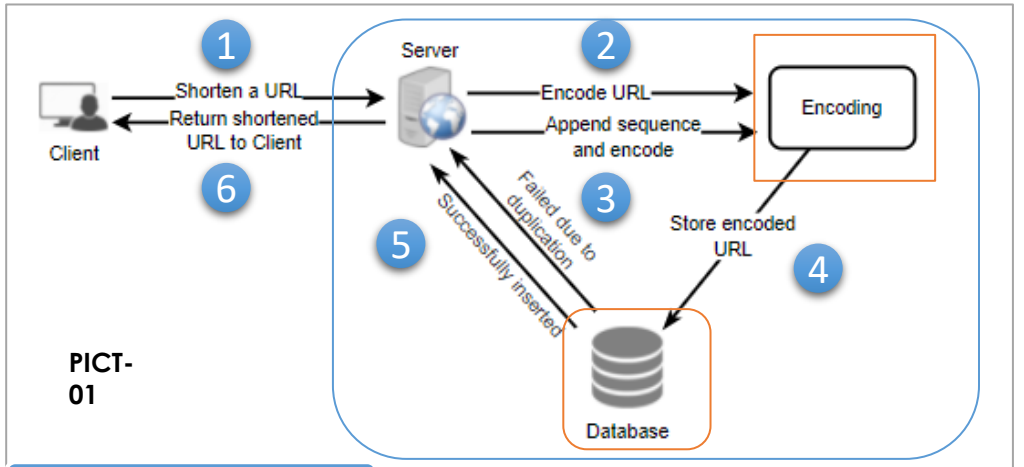
Step 04: **Isn't KGS a single point of failure?** Yes, it is. To solve this, we can have a standby replica of KGS. Whenever the primary server dies, the standby server can take over to generate and provide keys.

Step 05: **Can each app server cache some keys from key-DB?** Yes, this can surely speed things up. Although, in this case, if the application server dies before consuming all the keys, we will end up losing those keys. This can be acceptable since we have 68B unique six-letter keys.

Step 06: **How would we perform a key lookup?** We can look up the key in our database to get the full URL. If it's present in the DB, issue an "HTTP 302 Redirect" status back to the browser, passing the stored URL in the "Location" field of the request. If that key is not present in our system, issue an "HTTP 404 Not Found" status or redirect the user back to the homepage.

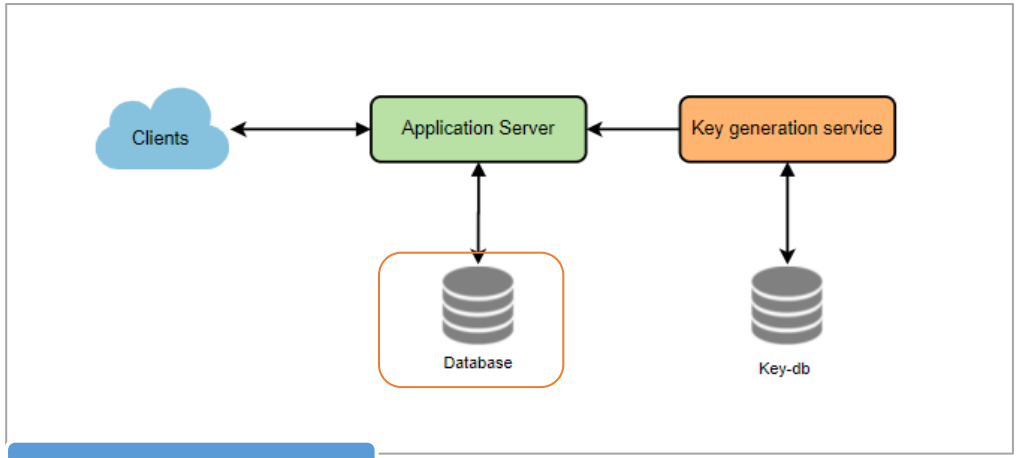
Step 07 : **Should we impose size limits on custom aliases?** Our service supports custom aliases. Users can pick any 'key' they like, but providing a custom alias is not mandatory. However, it is reasonable (and often desirable) to impose a size limit on a custom alias to ensure we have a consistent URL database.

Summary



PICT-01

Encoding URL Approach



Generating Key Offline

System design : High Level - System design blocks 4/5

Can concurrency cause problems?

If there are multiple servers reading keys concurrently, we might get a scenario where two or more servers try to read the same key from the database. **How can we solve this concurrency problem?**

Data Partition and Replication?

To scale out our DB, we need to partition it so that it can store information about billions of URLs.

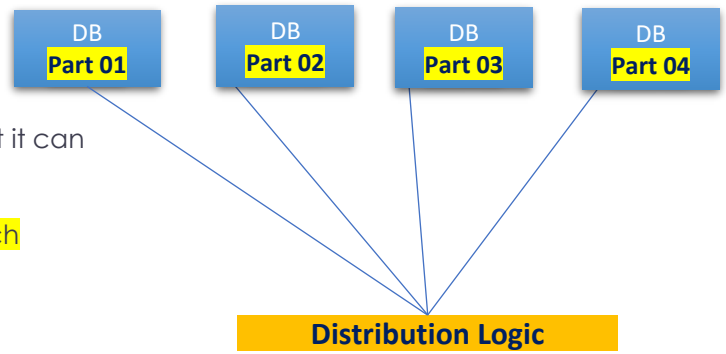
Therefore, we need to develop a partitioning scheme that would divide and store our data into different DB servers.

Ok, so let's think about , how to partitioning the table here in our case
Majorly we are dealing here for ShortURL, OrigionalURL and UserID

a. Range Based Partitioning:
In this approach, we are planning to divide the first character of our short URL into range and assign range to DB server. Meaning here, suppose your first letter of my short URL fall in A~D character than as per range division it will go Part 01 DB server, and so on…

Conceptual Diagram

We can even combine certain less frequently occurring letters into one database partition.



The main problem with this approach is that it can lead to unbalanced DB servers.

So, we should look for some better approach

b. Hash based partitioning

In this scheme, we take a hash of the object we are storing. We then calculate which partition to use based upon the hash. In our case, we can take the hash of the 'key' or the short link to determine the partition in which we store the data object.

And one more point I would like to mention here that we might have to use consistent hashing in this case to minimize the impact of certain server failure or addition for scale up.

So, this much information seems enough for partitioning and than every partition should be backed with replication to maintain the resiliency into our system.

Cache
In this system we already observed that there are some of the ShortURL which is hitting more and more than others because these shortURL is for some celebrity video

How much cache memory?
So, 20 * 1000 * 60 * 60 * 24 per day requirement. It comes, 172 GB (approx. = .02 TB) per day

Apart from size, we also have to plan the cache eviction policy here.

Which cache eviction policy would best fit our needs?
Least Recently Used (LRU) can be a reasonable policy for our system.

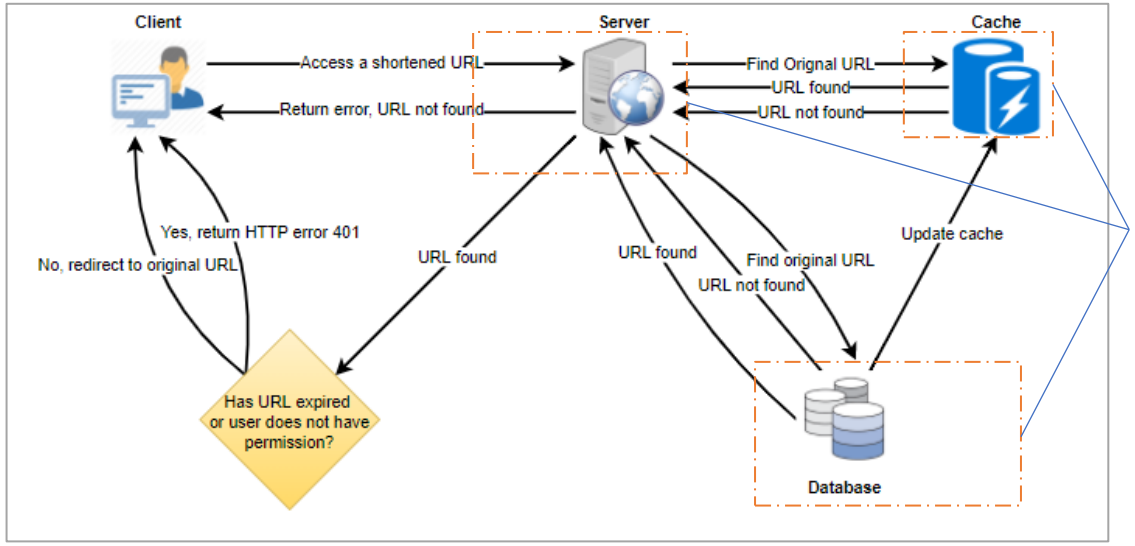
To further increase the efficiency, we can replicate our caching servers to distribute the load between them.

Contd on next page….

System design : High Level - System design blocks 5/5

Contd from last page for Cache discussion

How can each cache replica be updated?
Whenever there is a cache miss, our servers would be hitting a backend database. Whenever this happens, we can update the cache and pass the new entry to all the cache replicas. Each replica can update its cache by adding the new entry. If a replica already has that entry, it can simply ignore it.



Load Balancer

We can add a Load balancing layer at three places in our system:
1.Between Clients and Application servers
2.Between Application Servers and database servers
3.Between Application Servers and Cache servers

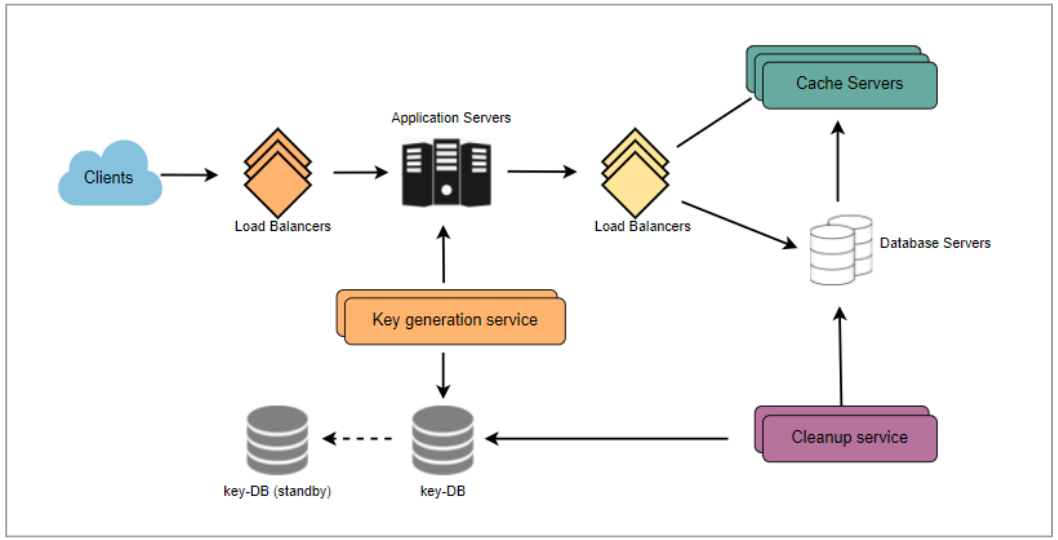
Infect load balancer also have different strategy, but I am not going much in detail here to save this discussion for another article. One of the good strategy for load balancer might be **weighted round robin load balancer** , where we will collect the load of server by pinging them for load enquiry.

DB Purging or DB Cleanup

If you observe this system, you will clearly find that many DB entry will go out to dead entry because our generated shortURL have attached time span. So, with this time span and so many dead record in DB table don't help system without having proper DB purging and DB cleanup approach.

We should create one DB Cleanup service and it will clean the dead links from table with lazy cleanup approach, but we should make sure that any deadlink is not getting return to the user of our system.

After considering all these components , services , LB, cluster replication our system will look like …



Apart from above system, I observe that few more service like some monitoring of system usage analytics and some security module for handling the API call brutality, Middle man attack on shortURL etc…may also as add value in this system. I would like to discuss that in my next topic from where you can pick and add in this article also. For Now I am ending this article here only. Keep Learning and Keep Growing…