Tiny URL System Design

One of the TinyURLs application is BitLy

From the name, we can assume we will get a long url and that will be converted into a short url. But don't assume.

What's TinyURL?
- You will provide the short URL for a given long URL
- Example: say we have a long url: https://docs.udemy.com/docs/java ==> Convert it into a shorter url: https://tinyurl.com/4859846
Whenever user hits the tinyurl, it will be converted back to the original long url

Functional requirements:
- Whenever you provide a long url, our system should be able to process it and give a short url (long url => short url)
- Whenever user clicks on short url, they should get redirected to the long url (short url => long url)
- Do we need any user information or do we need to store user info?
User1 is creating short url. Do we need their name, ip? If yes, it will take up space in database. The core focus of this application is url conversion so we wont go into details of user details or authentication etc. Answer is No, we are not storing User personal information.
- How do we Update url?
long url1: abc ==> converted to 01
long url2: abd ==> converted to 02 new entry only
- When we are creating any url, do we have any expiry date on it?
    - 1 year or 10 years

Functionality:
1. getShortURL(String longUrl)
2. getShortURL(longUrl, expiryTime, userData)
3. getLongURL(shortUrl)

Non-functional requirements:
- Availability is a must, we cant have low availability
- Low latency, that means we cannot have complex code
- High durability, that means the data stored in the system must be durable

Back of the envelope estimation:
it shows how your system is going to look like. When creating a system, we need to know how many users do we have.
- How many users?
- What is the query per second (QPS)? 1000 requests / second
- Time period: time period the url will be stored in our system is 10 years
1000 requests / second * 60 minutes * 60 seconds * 24 hours / day * 365 days / year * 10 years = 3,15,36,00,00,000 requests
These are the number of short urls we are going to generate. Approximately 320 billion is the number of urls you are going to generate in the next 10 years from now.

longUrl (https://docs.coursera.com/docs/java)
shortUrl (https://tinyurl.com/----------)
What are the valid characters that can replace the '----------'?
Base62: 0-9 (10), a-z (26), A-Z(26) = 10+26+26 = 62
that means only alpha-numeric characters
Base64: 0-9, a-z, A-Z, -, _ = 10+26+26+2 = 64

Two basic ways of handling the replacement

What should be the length of '----------' then?
Not simply 3 or 10 characters.
If length is 1, then with Base62 we can have only up to 62 urls, which is not enough
if length is 2, then with Base2, we can have up to 62 ^ 2 = 68,644
if length is 3, then 62 ^ 3 = 2,38,328 not enough
for 4, 62 ^ 4 = 1,47,76,336
for 5, 62 ^ 5 = 91,61,32,832
for 6, 62 ^ 6 = 56,80,02,35,584
for 7, 62 ^ 7 = 35,21,61,46,06,208 (3.5 Trillion)

We need to have 7 characters length: (https://tinyurl.com/-------)

Data storage size:
How much data we need to store?
LongUrl size is assumed to be 100 bytes, however, ask interviewer if thats ok

LongUrl (100 bytes)	ShortUrl (7 bytes)	Expiration time (8 bytes)

To store one single record, we need minimum 115 bytes. For 1000 records / second = 115,000 bytes / second.
For 320 B url records, 115 * 320 billion = 37 TB space required for 10 years

Request per second = 1000

Data storage type:
Which database should we go to?
ShortUrls are not bound to any particular user. Say User1 created a shortUrl for longUrl: abc and if User2 requests the shortUrl for the same long Url: abc, we can return the same shortUrl to User2 also.

Do we need a Relational database or Non-relational database (SQL or NoSQL)?

Is the nature of data relational? Do we have to manage multiple tables to maintain relational constraints? In this case, we don't have to manage multiple tables. So NoSQL is ok

Issue	SQL	NoSQL
Relational data - No		Yes
ACID - No		Yes
Analytics - No		Yes
Complex queries - No		Yes

Do we need to maintain ACID properties? No. we need durability of data but do we need all other properties also? No. Is Atomicity our primary goal?
Do we need to perform any Analytics? Not needed because we are not storing user data
Do we need to handle any complex queries? No, we will simply pass shortUrl and get longUrl and vice-versa. Ability to handle complex queries is the major reason to go with SQL because in SQL we can handle complex queries. In NoSQL, complex queries becomes the bottleneck sometimes.

So for the above-mentioned reasons, we are picking NoSQL

In NoSQL also, we have different kinds of databases: Graph, Document, Columnar, Key-value database.
For tiny url, we can pick key-value. For Key-value, we have picked DynamoDB.
We can look into the features of DynamoDB and convince the decision
if you don't know anything about DynamoDB then you may go with MySQL

Logic part:

We know we get some longUrl ===> Algorithm ===> shortUrl
We will pick Base62: 0-9, a-z, A-Z
longUrl ==> Hash function ==> Encoded value of 7 characters length, which we determined earlier
This 7 characters say, az37578, will be stored in the database and server as shortUrl ID.

Limitations:
Lets say the Hash function is generating: az129679ab and az129678ab for two different longUrls, both are 10 characters long but we pick only first 7 characters and we end up with the same values (az12967) for both longUrls. This is called as 'Url collision'.

What's the solution for url collision?
- enhance the Hash logic to improve collision
Add some value at end of url like /k=k but they look bad, we got to remove them later – not a good solution.
We know we are going to use 62 characters, so create a counter say it starts with 1. for 1, we will calculate the Base62(1), we will put that as shortUrl
Base8 ==> 0, 1, 2, 3, 4, 5, 6, 7, (Octal)
Base10 ==> 0,1,2,3,4,5,6,7,8,9
Base16 ==> 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F (Hexadecimal)

We will create a Base62(counter unique value), so the Hash value will be unique also for shortUrl.

LongUrl ==> Counter value ==> Algorithm ==> Database

Say if counter value is 101 then Base62(101) will go into database

1. getCounter()
2. Create Base62(counter value)
3. Save the value in databases

By doing this, all our shortUrls will be unique and we have solved collision problem. The disadvantage is we have to handle the counter separately.

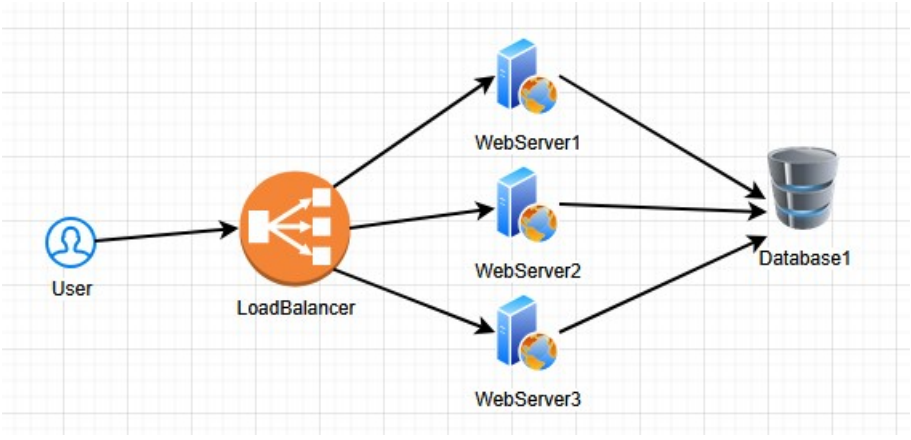
Distributed environment:

Lets say the current environment has only one server. Can 1 server handle 1000 rps? It will add latency plus it has high risk of going down.

In Non-functional requirements we mentioned high Availability and one server is not an option

Non-functional requirements:
- Availability is a must, we cant have low availability
- Low latency, that means we cannot have complex code
- High durability, that means the data stored in the system must be durable

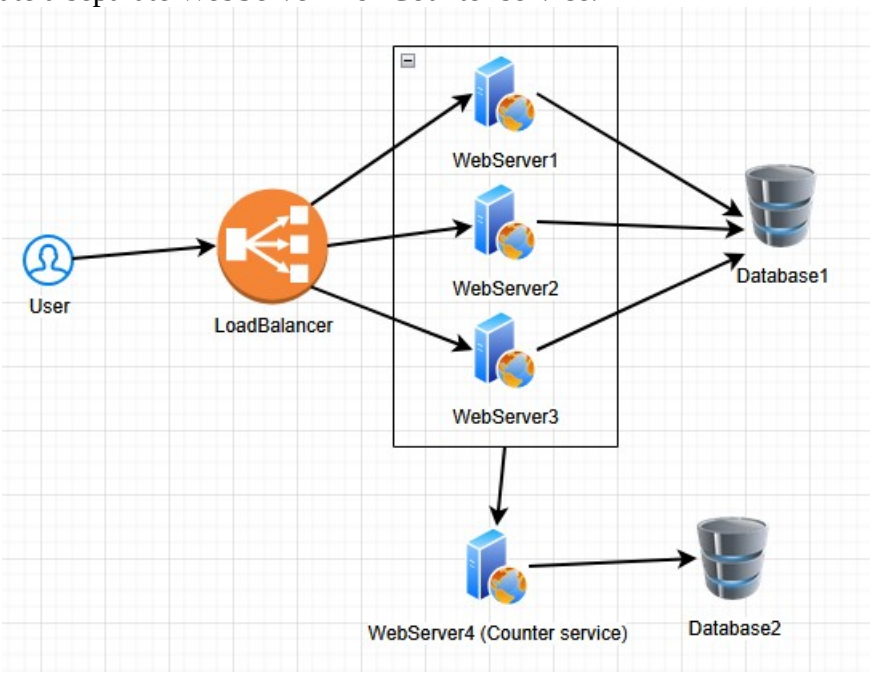
This is the normal structure of Url encoder



Lets say we are using EC2, whenever a server restarts, will it impact our counter?
How will you handle the counter in the multi-server environment?
WebServer1 can handle a counter range 0 to 10,000 so they work independently.
WebServer2 can handle a counter range 10,001 to 20,000
WebServer2 can handle a counter range 20,001 to 30,000

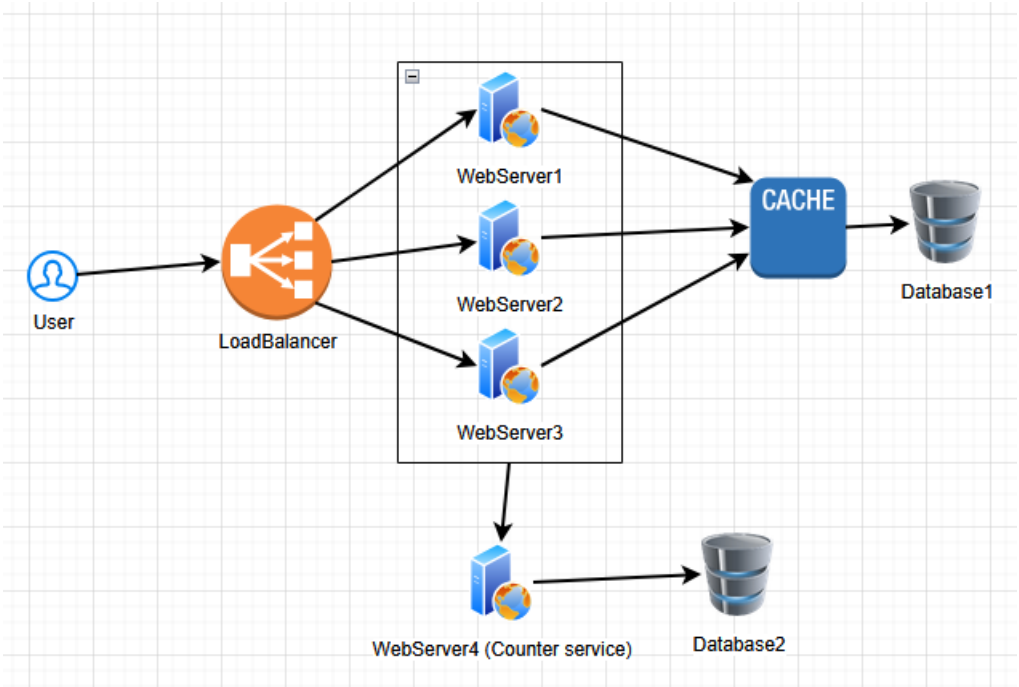
Lets say, WebServer2 has almost consumed their counter range, in that case, it will give a call to getCounterRange() and it will provide new range to WebServer2 (30,001 to 40,000).

We could also create a separate WebServer4 for Counter service.



Availability part is solved by adding multiple servers but Latency part is still an issue.

Cache solves Latency issue as it will decrease response time



We have solved Availability and Latency problems

We could have different solutions also for the same problems