URL Shortner (bit.ly, TinyURL, etc) – System Design

Last Updated : 20 Jan, 2025

The need for an efficient and concise URL management system has become a big matter in this technical age. URL shortening services, such as bit.ly, and TinyURL, play a massive role in transforming lengthy web addresses into shorter, shareable links. As the demand for such services grows, it has become vital to understand the System Design of URL shorteners and master the art of designing a scalable and reliable URL-shortening system.

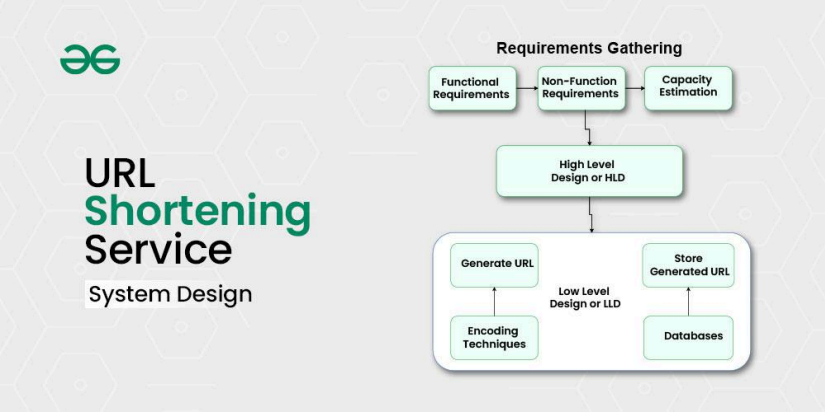


Table of Content

What is a URL Shortening service?

A URL shortening service takes a long, complex web address and converts it into a shorter, more manageable link. This shorter URL redirects users to the original destination, making sharing links easier and cleaner—especially on platforms with character limits, like Twitter. Common examples include services like Bit.ly and TinyURL, which create concise links that are easy to remember and track.

How Would You Design a URL Shortener Service Like TinyURL?

URL shortening services like bit.ly or TinyURL are very popular to generate shorter aliases for long URLs. You need to design this kind of web service where if a user gives a long URL then the service returns a short URL and if the user gives a short URL then it returns the original long URL.

For example, shortening the given URL through TinyURL:

https://www.geeksforgeeks.org/system-design-interview-bootcamp-guide/

We get the result given below:

http://bit.ly/3uQqImU

Requirements for URL Shortner Service System Design

1. Functional requirements

- Given a long URL, the service should generate a shorter and unique alias for it.
- When the user hits a short link, the service should redirect to the original link.
- Links will expire after a standard default time span.

2. Non-Functional requirements

start failing.
- URL redirection should happen in real-time with minimal latency.
- Shortened links should not be predictable.

Capacity estimation for System Design of URL Shortner

Let's assume our service has 30M new URL shortenings per month. Let's assume we store every URL shortening request (and associated shortened link) for **5 years**. For this period the service will generate about 1.8 B records.

30 million * 5 years * 12 months = 1.8B

Note: Let's consider we are using 7 characters to generate a short URL. These characters are a combination of 62 characters [A-Z, a-z, 0-9] something like **http://ad.com/abXdef2**.

Data Capacity Modeling

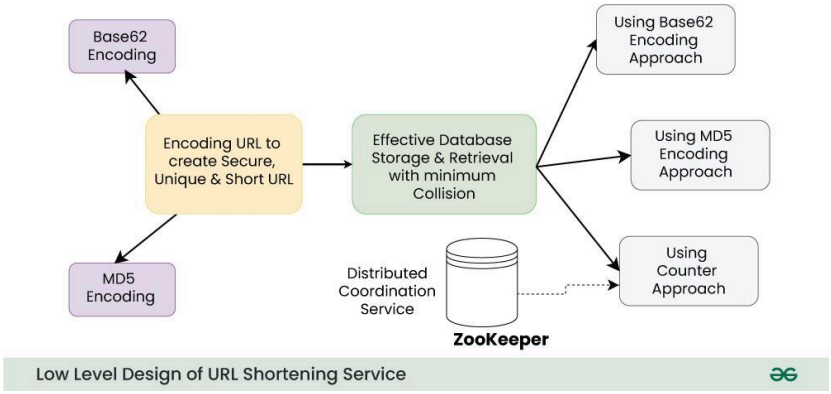
Discuss the data capacity model to estimate the storage of the system. We need to understand how much data we might have to insert into our system. Think about the different columns or attributes that will be stored in our database and calculate the storage of data for five years. Let's make the assumption given below for different attributes.

- Consider the average long URL size of 2KB ie for 2048 characters.
- Short URL size: 17 Bytes for 17 characters
- created_at- 7 bytes
- expiration_length_in_minutes -7 bytes

The above calculation will give a total of 2.031KB per shortened URL

Note: We need to think about the reads and writes that will happen on our system for this amount of data. This will decide what kind of database (RDBMS or NoSQL) we need to use.

Low-Level Design for System Design of URL Shortner



URL Encoding Techniques to create Shortened URL

To convert a long URL into a unique short URL we can use some hashing techniques like Base62 or MD5. We will discuss both approaches.

1. Base62 Encoding

- Base62 encoder allows us to use the combination of characters and numbers which contains A-Z, a-z, 0–9 total(26 + 26 + 10 = 62).
- So for 7 characters short URL, we can serve 62^7 ~= **3500 billion URLs** which is quite enough in comparison to base10 (base10 only contains numbers 0-9 so you will get only 10M combinations).
- We can generate a random number for the given long URL and convert it to base62 and use the hash as a short URL id.

If we use base62 making the assumption that the service is generating 1000 tiny URLs/sec then it will take 110 years to exhaust

Python JavaScript

```python
def to_base_62(deci):
    s = '012345689abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
    hash_str = ''
    while deci > 0:
        hash_str = s[deci % 62] + hash_str
        deci /= 62
    return hash_str


print to_base_62(999)
```

2. MD5 Encoding

MD5 also gives base62 output but the MD5 hash gives a lengthy output which is more than 7 characters.

- MD5 hash generates 128-bit long output so out of 128 bits we will take 43 bits to generate a tiny URL of 7 characters.
- MD5 can create a lot of collisions. For two or many different long URL inputs we may get the same unique id for a short URL and that could cause data corruption.
- So we need to perform some checks to ensure that this unique id doesn't exist in the database already.

Efficient Database Storage & Retrieval of TinyURL

Let's discuss the mapping of a long URL into a short URL in our database:

1. Using Base62 Encoding

Assume we generate the Tiny URL using base62 encoding then we need to perform the steps given below:

- The tiny URL should be unique so firstly check the existence of this tiny URL in the database (doing get(tiny) on DB). If it's already present there for some other long URL then generate a new short URL.

*This technique works with one server very well but if there will be multiple servers then this technique will create a **race condition** .*

- *When multiple servers will work together, there will be a possibility that they all can generate the same unique id or the same tiny URL for different long URLs.*
- *Even after checking the database, they will be allowed to insert the same tiny URLs simultaneously in the database and this may end up corrupting the data.*

2. Using MD5 Approach

- Encode the long URL using the MD5 approach and take only the first 7 chars to generate TinyURL.
- The first 7 characters could be the same for different long URLs so check the DB (as we have discussed in Technique 1) to verify that TinyURL is not used already.

This approach saves some space in the database but how?

- If two users want to generate a tiny URL for the same long URL then the first technique will generate two random numbers and it requires two rows in the database.
- In the second technique, both the longer URL will have the same MD5 so it will have the same first 43 bits.
- This means we will get some deduping and we will end up with saving some space since we only need to store one row instead of two rows in the database.

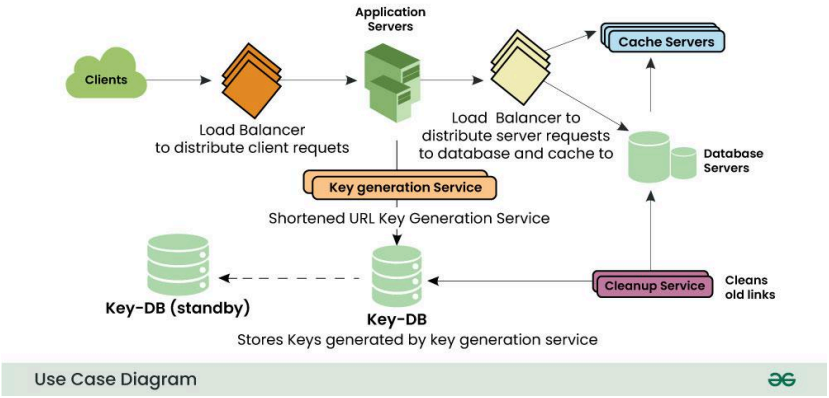
3. Using Counter Approach

Using a counter is a good decision for a scalable solution because

Single server approach:

- A single host or server (say database) will be responsible for maintaining the counter.
- When the worker host receives a request it talks to the counter host, which returns a unique number and increments the counter. When the next request comes the counter host again returns the unique number and this goes on.
- Every worker host gets a unique number which is used to generate TinyURL.

High-level Design of a URL-Shortening Service



Use Case Diagram

- **User Interface/Clients:**
    - The user interface allows users to enter a long URL and receive a shortened link. This could be a simple web form or a RESTful API.
- **Application Server:**
    - The application server receives the long URL from the user interface and generates a unique, shorter alias or key for the URL. It then stores the alias and the original URL in a database. The application server also tracks click events on the shortened links.

instances of the application server. We can add a Load balancing layer at three places in our service:

    - Between Clients and Application servers
    - Between Application Servers and database servers
    - Between Application Servers and Cache servers

- **Database:**
    - The database stores the alias or key and the original URL. The database should be scalable to handle a large number of URLs and clicks. We can use NoSQL databases such as MongoDB or Cassandra, which can handle large amounts of data and can scale horizontally.
    - As soon as a key is used, it should be marked in the database to ensure it doesn't get used again. 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
- **Caching**:
    - Since reading from the database can be slow and resource-intensive, we can add a caching layer to speed up read operations. We can use in-memory caches like Redis or Memcached to store the most frequently accessed URLs.
- **Cleanup Service:**
    - This service helps in cleaning the old data from the databases
- **Redirection:**
    - When a user clicks on a shortened link, the application server looks up the original URL from the database using the alias or key. It then redirects the user to the original URL using HTTP 301 status code, which is a permanent redirect.

includes the number of clicks, the referrer, the browser, and the device used to access the link.

- **Security:**
    - The service should be designed to prevent malicious users from generating short links to phishing or malware sites. It should also protect against DDoS attacks and brute force attacks. We can use firewalls, rate-limiting, and authentication mechanisms to ensure the security of the service.

Database Design

Let us explore some of the choices for System Design of Databases of URL Shortner:

- We can use RDBMS which uses ACID properties but you will be facing the scalability issue with relational databases.
- Now if you think you can use **sharding** and resolve the **scalability** issue in RDBMS then that will increase the complexity of the system.
- There are 30M active users so there will be conversions and a lot of Short URL resolution and redirections.
- Read and write will be heavy for these 30M users so scaling the RDBMS using shard will increase the complexity of the design.

> *You may have to use **consistent hashing** to balance the traffic and DB queries in the case of RDBMS and which is a complicated process. So to handle this amount of huge traffic on our system relational databases are not fit and also it won't be a good decision to scale the RDBMS.*

So let's take a look at NoSQL Database:

requirement.

- NoSQL can easily handle the 30M of active users and it is easy to scale. We just need to keep adding the nodes when we want to expand the storage.

Caching and Load Balancing in URL Shortening service

In a URL shortening service, caching and load balancing are essential for managing high demand and optimizing response times. The service could greatly benefit from read-through caching or write-through caching mechanisms.

- A read-through cache automatically loads data into the cache when a miss occurs
- While a write-through cache updates the cache whenever the database is updated.

In this scenario, a read-through cache would be especially useful since shortened URLs are likely accessed multiple times. Redis or Memcached would be good choices for the caching layer due to their speed and support for frequently accessed data.

For load balancing, algorithms like Round Robin or Least Connections are effective choices. Round Robin evenly distributes incoming traffic across servers, making it easy to implement and scalable. However, for services with variable request times, Least Connections can be better, as it allocates requests based on the server's current load.

Conclusion

Overall, a URL shortening service is a simple but useful application that can be built using a variety of technologies and architectures. By following the above architecture, we can build a scalable, reliable, and secure URL-

Next Article

Load Balancer - System Design Interview Question

Similar Reads

Most Commonly Asked System Design Interview Problems/Questions

This System Design Interview Guide will provide the most commonly asked system design interview questions and equip you with the knowledge and techniques needed to design, build, and scale your...

2 min read

Grokking Modern System Design Interview Guide

{ "header": { "title": "Grokking Modern System Design Interview Guide", "description": "This System Design Interview Guide will provide the latest system design interview questions and equip you with...

3 min read

How to Crack System Design Interview Round?

In the System Design Interview round, You will have to give a clear explanation about designing large scalable distributed systems to the interviewer. This round may be challenging and complex for you...

9 min read

System Design Interview Questions and Answers [2024]

In the hiring procedure, system design interviews play a significant role for many tech businesses, particularly those that develop large, reliable software systems. In order to satisfy requirements like...

7 min read

---

5 Common System Design Concepts for Interview Preparation

In the software engineering interview process system design round has become a standard part of the interview. The main purpose of this round is to check the ability of a candidate to build a complex and...

12 min read

6 min read

Chat Applications Design Problems

Designing Facebook Messenger | System Design Interview

We are going to build a real-time Facebook messaging app, that can support millions of users. All the chat applications like Facebook Messenger, WhatsApp, Discord, etc. can be built using the same…

9 min read

Media Storage and Streaming Design Problems

System Design Netflix | A Complete Architecture

Designing Netflix is a quite common question of system design rounds in interviews. In the world of streaming services, Netflix stands as a monopoly, captivating millions of viewers worldwide with its…

15+ min read

Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:
K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

Company
About Us
Legal
Privacy Policy
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain

Data Science & ML
Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy