educative Ask a Question

Evaluation of TinyURL's Design

Let's evaluate the short URL service design based on its non-functional requirements.

We'll cover the following

Reviewing the requirements

The last stage of a system design is to evaluate it as per the non-functional requirements mentioned initially. Let’s look at each metric one by one.

Availability

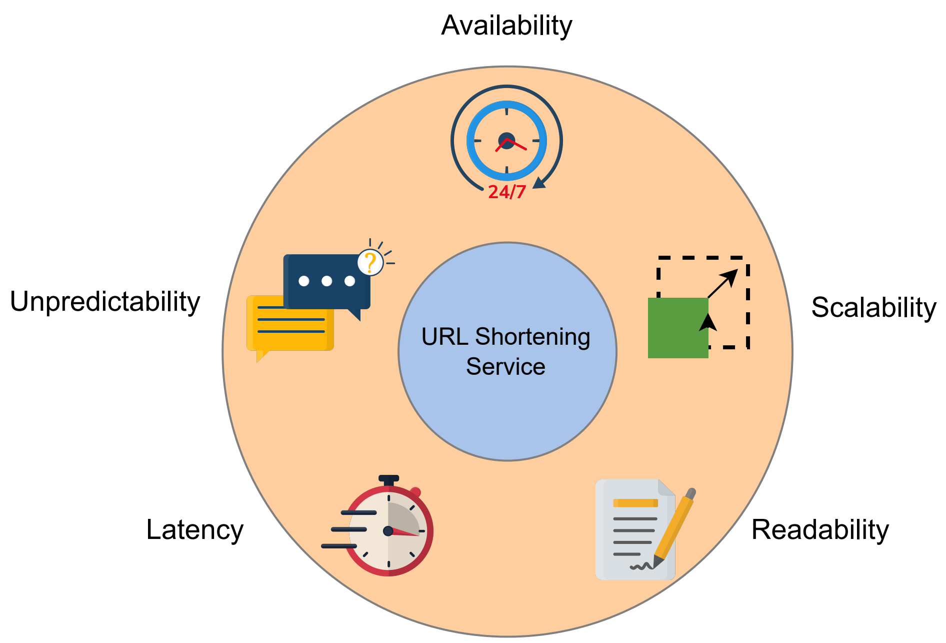
We need high availability for users generating new short URLs and redirecting them based on the existing short URLs.

Most of our building blocks, like databases, caches, and application servers have built-in replication that ensures availability and fault tolerance. The short URL generation system will not impact the availability either, as it depends on an easily replicable database of available and used unique IDs.

To handle disasters, we can perform frequent backups of the storage and application servers, preferably twice a day, as we can't risk losing URLs data. We can use the Amazon S3 storage service for backups, as it facilitates cross-zonal replicating and restoration as well. In the worst-case scenario, we might lose 3.3 Million (with 6.6 Million daily requests assumed) newly generated short URLs that are not backed up on that specific day.

Our design uses **global server load balancing (GSLB)** to handle our system traffic. It ensures intelligent request distribution among different global servers, especially in the case of on-site failures.

We also apply a limit on the requests from clients to secure the intrinsic points of failures. To protect the system against DoS attacks, we use rate limiters between the client and web servers to limit each user's resource allocation. This will ensure a good and smooth traffic influx and mitigate the exploitation of system resources.



Guarantees provided by our URL shortening service

Scalability

Our design is scalable because our data can easily be distributed among horizontally sharded databases. We can employ a consistent hashing scheme to balance the load between the application and database layers.

Alternate to consistent hashing in load balancer

Our choice of the database for mapping URLs, MongoDB, also facilitates horizontal scaling. Some interesting reasons for selecting a NoSQL database are:

1. When a user accesses our system without logging in, our system doesn't save the `UserID`. Since we're flexible with storing data values, and it aligns more with the schematic flexibility provided by the NoSQL databases, using one for our design is preferable.
2. Scaling a traditional relational database horizontally is a daunting process and poses challenges to meeting our scalability requirements. We want to scale and automatically distribute our system's data across multiple servers. For this requirement, a NoSQL database would best serve our purpose.

Moreover, the large number of unique IDs available in the sequencer's design also ensures the scalability of our system.

Readability

The use of a base-58 encoder, instead of the base-64 encoder, enhances the readability of our system. We divide the readability into two sections:

- **Distinguishable characters** like `0` (zero), `O` (capital o), `I` (capital i), anc (lower case L) are eliminated, excluding the possibility of mistaking a character for another look-alike character.

- **Non-alphanumeric characters** like `+` (plus) and `/` (slash) are also eliminated to only have alphanumeric characters in short URLs. Second, it also helps avoid other system-dependent encodings and makes the URLs readily available for all modern file systems and URLs schemes. Such characters may lead to undesired behavior and output during parsing.

This non-functional requirement enhances the user interactivity of our system and makes short URL usage less error-prone for the users.

Fulfilling Non-functional Requirements

Requirements	Techniques
Availability	• The Amazon S3 service backs up the storage and cac can restore them upon fault occurrence. • Global server load balancing to handle the system's t • Rate limiters to limit each user's resource allocation.
Scalability	• Horizontal sharding of the database. • Distribution of the data based on consistent hashing. • MongoDB - as the NoSQL database.
Readability	• Introduction of the base-58 encoder to generate shor • Removal of non-alphanumeric characters. • Removal of look-alike characters.
Latency	• Unnoticeable delay in the overall operation. • MongoDB, because of its low latency and high throu tasks. • Distributed cache to minimize the service delays.
Unpredictability	• Randomly selecting and associating an ID to each re of unused and readily available unique IDs.

Latency

Our system ensures low latency with its following features:

- Even the most time-consuming step across the short URL generation process, encoding, takes a few milliseconds. The overall time to generate a short URL is relatively low, ensuring there are no significant delays in this process.
- Our system is redirection-heavy. Writing on the database is minimal compared to reading, and its performance depends on how well it copes with all the redirection requests, compared to the shortening requests. We deliberately chose MongoDB because of its low latency and high throughput in reading-intensive tasks.
- Moreover, the probability of the user using the freshly generated short URL in the next few seconds is relatively low. During this time, synchronous replication to other locations is feasible and therefore adds to the overall low latency of the system for the user.
- The deployment of a distributed cache in our design also ensures that the system redirects the user with the minimum delay possible.

As a result of such design modifications, the system enjoys low latency and high throughput, providing good performance.

Unpredictability

One of the requirements is to make our system's short URLs unpredictable, enhancing the security of our system.

As the sequencer generates unique IDs in a sequence and distributes ranges among servers. Each server has a pre-assigned range of unique IDs, assigning them serially to the requests will make it easy to predict the following short URL. To counter it, we can randomly select a unique ID fro the available ones and associate it to the long URL, encompassing the unpredictability of our system.

Conclusion

The URL shortening system is an effective service with multiple advantages. Our design of the URL shortening service is simple, yet it fulfills all the requirements of a performant design. The key features offered by our design are:

1. A dynamic short URL range
2. Improved readability

A possible addition could be the introduction of (local) salt to further increase the unpredictability (security) of the design.

← Back
Encoder for TinyURL

Next →
Quiz on TinyURL's Design

✓ Mark as Completed