SYSTEM DESIGN - III

Tiny URL:

A free URL shortening service that produces short, memorable alias URLs for existing web pages. It enables users to enter a long URL and add a custom parameter to create the shortened URL. This URL then redirects to the web page from the original URL. The shortened URL never expires, so as long as the destination URL is the same, the shortened version will always be functional.

These URLs are universally unique. These kinds of URLs are most commonly used in email campaigns, social media posts, print advertisements, and other marketing efforts where an easy-to-remember URL (or one with limited characters) is beneficial for driving traffic to a specific page. It's also helpful for adding UTM parameters to a link without making them obvious to the audience.

In addition to Tiny URL, other examples of URL shortening services include Bit.ly, Rebrandly, Owl.ly, Cut.ly, and Shorte.ST. Unlike some of its competitors, Tiny URL has always provided all of its features for free. Some social media platforms like Twitter and LinkedIn also have inherent URL shortening services, but users must create an account to draft a post and access those features.

Software designing is more about ***"thinking"*** and ***"asking questions"***. Get specific and minimum requirements for the project. When questioned about how to make a tiny URL, find out the requirements for the short URL. Few things that we should ask them is:

1. **MVP → M**inimal **V**iable **P**roject.
    a. How many users do I have to serve? Asking about the **scale** of the project.
    b. Do the system login / sign-up? Mostly, interviewer will not ask for this.
    c. What is Read-to-Write ratio? (1:10, for every 1 **write** you have 10 **reads**)
    d. Availability is important than consistency, highly available.
    e. If some gives a big URL, the system should give short URL
    f. Necessity of Analytics – to know how many people are visiting, how many redirects, how many requests are failing and many such details can be found. This is done by Analyst / Product Managers.
    g. The above list of requirements is called ***Functional*** and **Non-*Functional Requirements.***

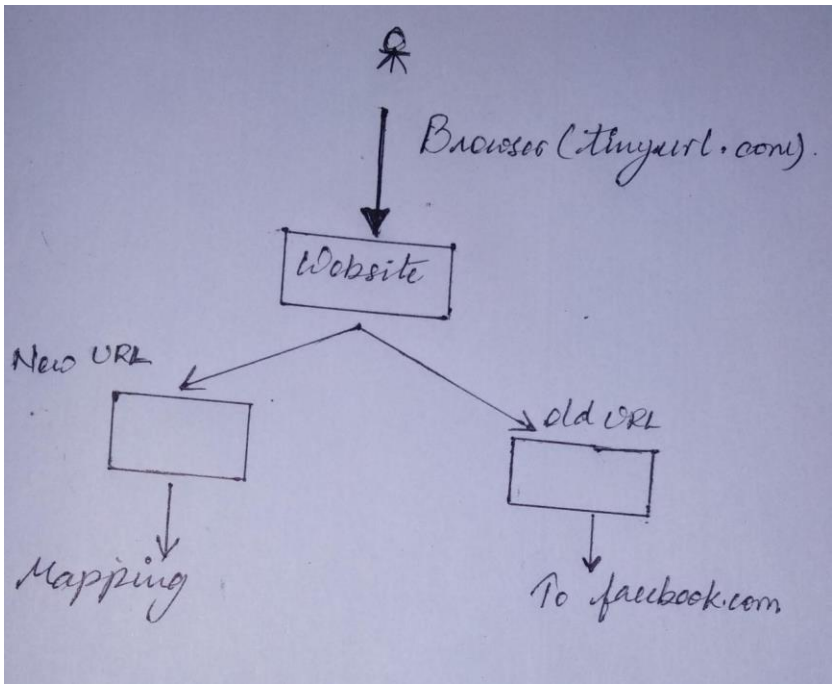
2. **Calculations**: We do some random calculations to make an estimate about size of database we need and how many servers we need.

    a. The interview says 1 billion users, (1 Billion = 10^{10}) are to be served. If we have 1 billion users are coming on daily basis, then we would have **1,157,40 users / second**. As per the read to write ratio (1:10), we would have **11,574** would be **writer** and **104,166** would be **readers**. If interviewer says on an average, one computer can handle 200 users, then we need approximately **6 computers for master** and the same number for **slaves**. These are called back of the envelop calculations.

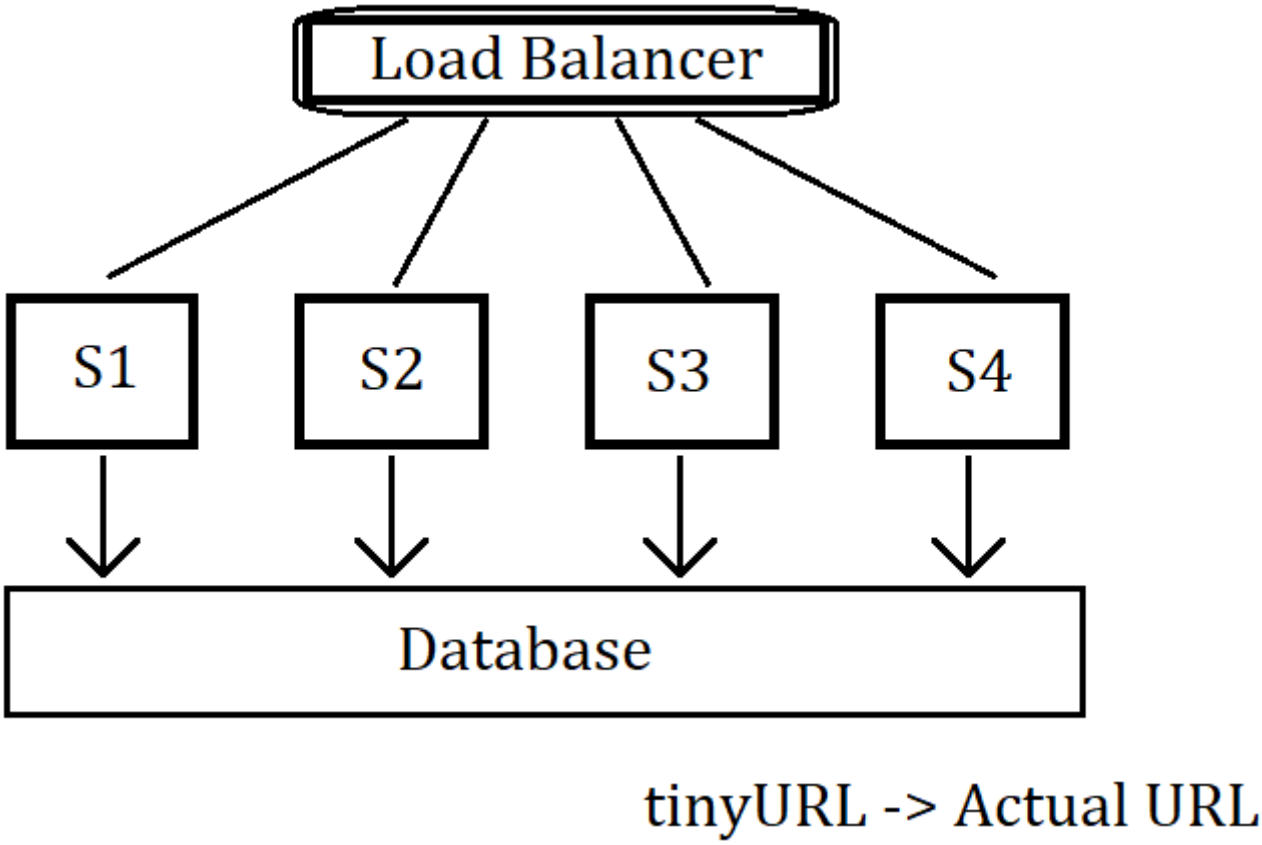
    b. For sudden increase in the users, then we have to calculate for the number of servers in addition as back up to handle the huge number of loads.

    c. **Storage Calculation:** Now the length of the URL is **1000**, but we are giving Tiny URL. The original length of URL would be around **1000** bytes (assume) so for Tiny URL, we have to 20 bytes. So, the total space we need would be **1000 bytes + 20 bytes = 1020 bytes = 1 MB (approx.)** for 1 sec, it would write **11,574**, so for 1 second, the storage would be **11,574 X 1MB**. And for one day, we would need **11574 X 1MB X 24 X 60 X 60.** If we are to run this website for 5 yrs. then we need **11574 X 1MB X 24 X 60 X 60 X 5 MB** of storage capacity.

3. **High Level Picture:** of the architecture**.**
The client in his browser hit Tiny URL. In the box, he will have two options, one is a Tiny URL and the other is new URL. The Tiny URL will check the mapping if the Tiny URL is already mapped to a particular website previously. This is the high-level view of the system.



Final Picture of the Design



We have a load balancer with 4 servers, connected to a database. Between RAM and Database, RAM is faster. So, it would be better to store the data in the RAM instead of Database. We can have cache in the server. By using consistent hashing, load balancer can decide in which particular server the user has to be redirected to. All of this data is available in the caches or RAM. If the data is not present in the RAM, then the server will hit the database, get the data and put that in the RAM.

Since our reads are very big, it is always better for us to optimize the cache, because their request can be solved by the cache.

This is the system design of the Tiny URL.

Functional vs Non-Functional Requirements

Requirements analysis is very critical process that enables the success of a software project to be assessed. Requirements are generally split into two.

Functional Requirements	Non-Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How software system fulfils the functional reqs?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Verifies the functionality of the software.	Verifies the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc	Non-Functional Testing like Stress, Performance, Usability, Security
Usually easy to define.	Usually more difficult to define.
Example 1) Authentication of user whenever he/she logs into the system. 2) A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** 1) Emails should be sent with a latency of no greater than 12 hours from activity. 2) The site should load in 3 seconds when the number of simultaneous users is > 10000

They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility