System Design – URL Shortening

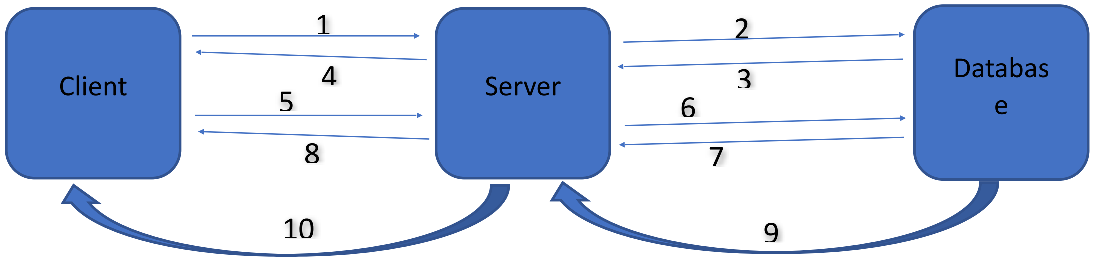
1. Requirement

- Given a long URL, the service should generate a shorter and unique alias of it.
- Click to the short URL should redirect the user to the original long URL
- URL redirection should be fast and should not degrade at any point of time

Requirements which are not part of the implementation but can help to improve system design:

- Links will expire after a standard default time span.
- The system should be highly available using multiple Master-Slave servers. This is really important to consider because if the service goes down, all the URL redirections will start failing.

2. High Level Design



1. Request for a shorten URL
2. Save shortened URL (created by server) and original URL to the database
3. Successfully stored
4. Return shorted URL to the client
5. Access a shorten URL
6. Get long URL
7. Return long URL
8. Redirect to long URL
9. No long URL found
10. Return not found error (do nothing)

3. Short URL Create Endpoints

- **Shorten_url():** This method calls encode() method to create unique alias and check whether it is already created before or not.
- **Encode():** convert base10 to base62 for alphanumeric short url creation.
    - **Time Complexity:** O(n * log_62(id)), Adding one item to dictionary takes O(1), so if we have n items adding new item takes O(n). converting base10 to base62 takes O(log_62(id)). therefore time complexity is **O(n*log_62(id))**
- **Return Value:** The short Url generated, or error code in case of the inappropriate parameter.

4. Database Schema

Database: Flask-SQLAlchemy

- **id:** A unique id or API key to make data distinguishable
- **short:** 6 character long unique short URL
- **long:** The original long URL

5. URL Shortening Logic (Encoding)

Initial Approach I've used is MD5 but it can create a lot of collisions. For two or many different long URL inputs we may get the same unique id for short URL and that could cause data corruption.

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).

6. Problems with above design :

- There is only one WebServer which is single point of failure.
- System is not scalable
- Base62 Encoding 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 same tiny URL for different long URLs, and even after checking the database, they will be allowed to insert the same tiny URLs simultaneously (which is the same for different long URLs) in the database and this may end up corrupting the data.

- There is only single database which might not be sufficient for 6 TB of storage and high load of 8000/s read requests

    - Assuming lifetime of service to be 10 years and with 100 million shortened links creation per month, total number of data points/objects in system will be = 100 million/month * 10 (years) * 12 (months) = 12 billion

      Assuming size of each data object (Short url, long url, created date etc.) to be 500 bytes long, then total require storage = 12 billion * 500 bytes =**6TB**

    - Assuming 200:1 read/write ratio

      Number of unique shortened links generated per month = 100 million

      Number of unique shortened links generated per seconds = 100 million /(30 days * 24 hours * 3600 seconds) ~ 40 URLs/second

      With 200:1 read/write ratio, number of redirections = 40 URLs/s * 200 = **8000 URLs/s**

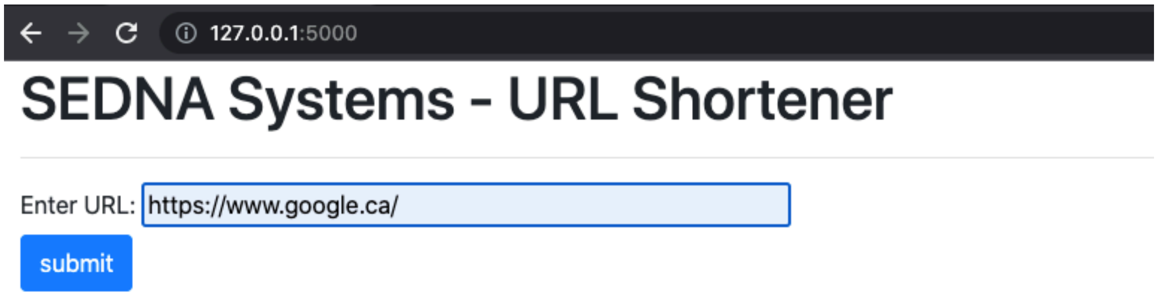
To cater above limitations we :

- Add a load balancer in front of Servers
- Implement sharded the database to handle huge object data
- Add cache system to reduce load on the database.

7. How to Run….

- Go to the application dir and run “docker-compose up” command.
- Open any browser and go to “127.0.0.1:5000” to access an application.
- Then enter any website in the text box and it will show shorted url of the original url.
- Open the short url link in the new browser window and it will redirect you to the original website.
- To see all the short urls go to "127.0.0.1:5000/all_urls”

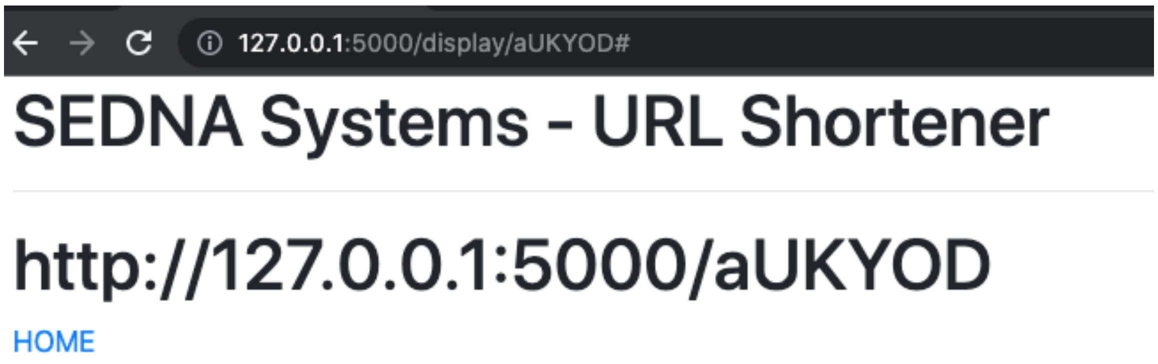
8. Application UI



127.0.0.1:5000

SEDNA Systems - URL Shortener

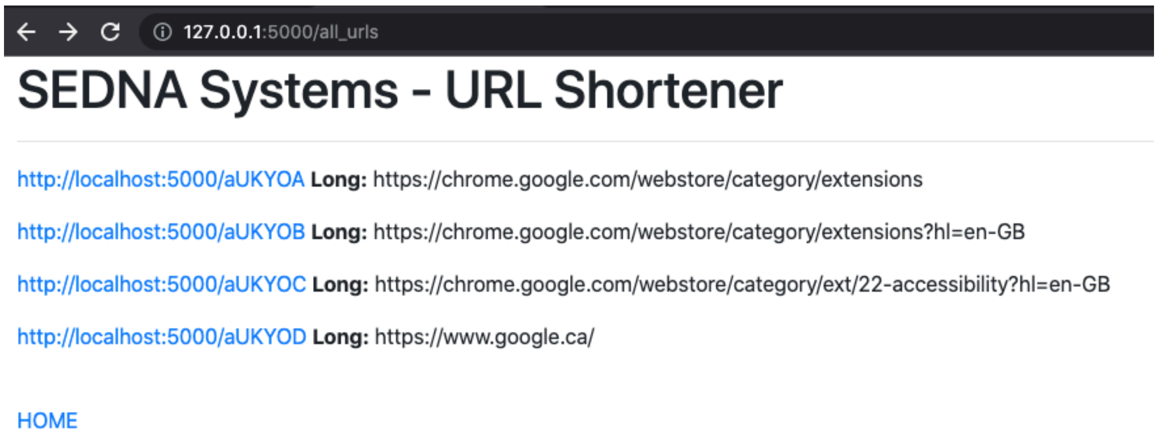
Enter URL: https://www.google.ca/



127.0.0.1:5000/display/aUKYOD#

SEDNA Systems - URL Shortener

http://127.0.0.1:5000/aUKYOD



127.0.0.1:5000/all_urls

SEDNA Systems - URL Shortener

http://localhost:5000/aUKYOA **Long:** https://chrome.google.com/webstore/category/extensions

http://localhost:5000/aUKYOB **Long:** https://chrome.google.com/webstore/category/extensions?hl=en-GB

http://localhost:5000/aUKYOC **Long:** https://chrome.google.com/webstore/category/ext/22-accessibility?hl=en-GB

http://localhost:5000/aUKYOD **Long:** https://www.google.ca/