URL shortener

Authors
Anuja Kench

Reviewers
Rupalim Sarma

Introduction -

This document provides an overview of the URL shortening project. It describes the functional and non-functional requirements, the system design, design considerations, tools to develop a proof of concept.

Functional requirements -

This project should support the following requirements -

1. Generate a short URL from a long URL.
2. Redirect the user using a short URL to the actual long URL.
3. List the number of times that a user accesses a URL in the last 24 hours, one week and all time.
4. Provide an interface to delete a short URL.
5. Support data persistence.
6. Provide data logging for troubleshooting.

Non-Functional requirements -

1. The system is read heavy.
2. The system must eventually scale to support millions of short URLs.
3. The system must provide high throughput and high availability.

Environment -

Go1.22.0
Visual Studio Code
Github

Git bash CLI

Design Decisions -

1. **URL format** - The short URL will look like http://myshorturl.com:8080/aHR0cHM6Ly9nbWFpbC5jb20=. The URL can only contain digits from 0 - 9, lower case letters a - z and upper case letters A - Z. Base64 encoding will be used to create a short URL. Base 64 encoding ensures that the only valid characters are supported.
2. **Data transfer** - JSON is used for encoding and decoding the request data since it's an industry standard.
3. **Access Time of URL -** The logic to find the access time of an URL starts from epoch time which is 0th hour and can only support past week's data in the current version.
4. **Error Handling -** Each API call returns an appropriate HTTP response code or date based on the intent.

Data Structures -

1. A short URL is stored as a string in hashmap for faster access and retrieval. The access time is stored as an unsigned integer.

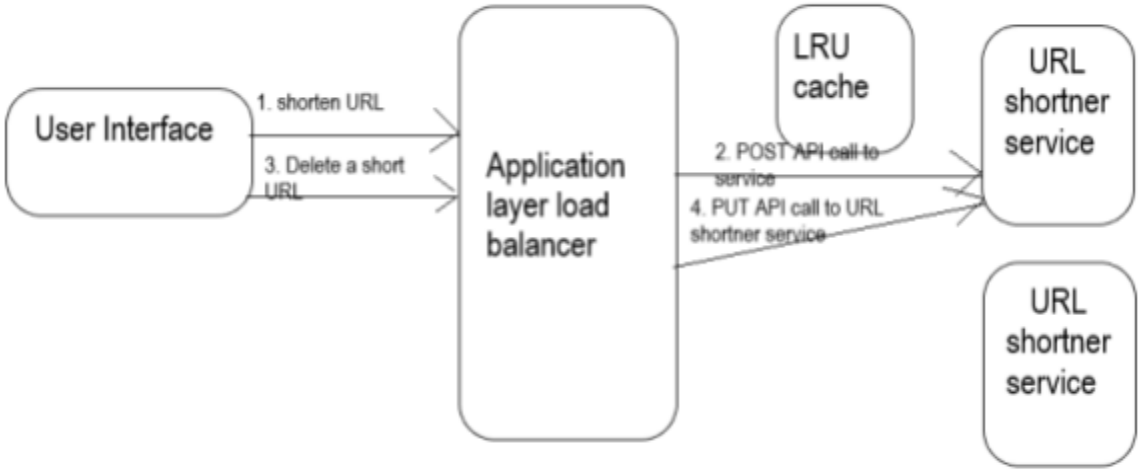
System Design -

A URL shortening service will be designed as a HTTP-based application. REST API's will be used to expose the functionality.
A long URL like https://google.com will be shortened to http://myshorturl.com/aHR0cHM6Ly9nb29nbGUuY29t
Following are the REST Api's implemented in the application -

1. **http://localhost:8080/shortenurl** - This is a POST API call to create a short URL. The request handler has validations to check for invalid HTTP request methods, JSON marshaling/unmarshalling errors and duplicate short URLs.

2. **http://localhost:8080/deleteurl** - This is a PUT API call to delete a short URL. The request handler has validations to check for invalid HTTP request methods, JSON marshaling/unmarshalling errors and duplicate short URLs.
3. **http://localhost:8080/redirecturl** - This is a POST API call to redirect short URL to long URL. The request handler has validations for invalid request methods, JSON marshaling/unmarshalling errors, request data not found errors.
4. **http://localhost:8080/urlaccessed -** This is a GET API call to return the number of times a URL is accessed.

Following is a high level diagram of the URL shortening service.



Future Design considerations -

1. **Cache** - For this project the URL data will be stored in memory using hashmap data structure.The system is **read heavy** so a cache could be implemented for **higher throughput**, **lower latency** and **higher response** time.
   Whenever there is a write operation in the secondary storage, then a request would be made to update the cache(Lazy

loading). Least frequently used(LFU) cache eviction policy can be implemented since users are most likely to access popular URLs like google.com and yahoo.com.

2. **Scalability** - An application level load balancer can be deployed between the client and the server to improve scalability. This is a future requirement and is not in the scope of the project.
3. **High availability** - To support high availability, multiple instances of URL generator service could be run so that in case of failure the load balancer can redirect the request to another service.
4. **Platform support** - This project is developed in a Windows environment. Ideally, a sandbox environment could be created using Vmware or other virtualization techniques so as to support multiple platforms like Linux apart from Windows OS.
5. **Cloud support** - The URL generator service can be enhanced to be run as a microservice that could be run in a EC2 instance in AWS cloud for example. This is not scoped in the first version of the project.