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 based on a timeframe.
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.

Assumptions and Design decisions -

1. **URL format** - The short URL will look like http://myshorturl.com:8080/acfr10as. The URL can 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.

2. **Cache assumptions** - 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.
3. **Scalability assumptions** - 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.
4. **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.
5. **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.
6. **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.

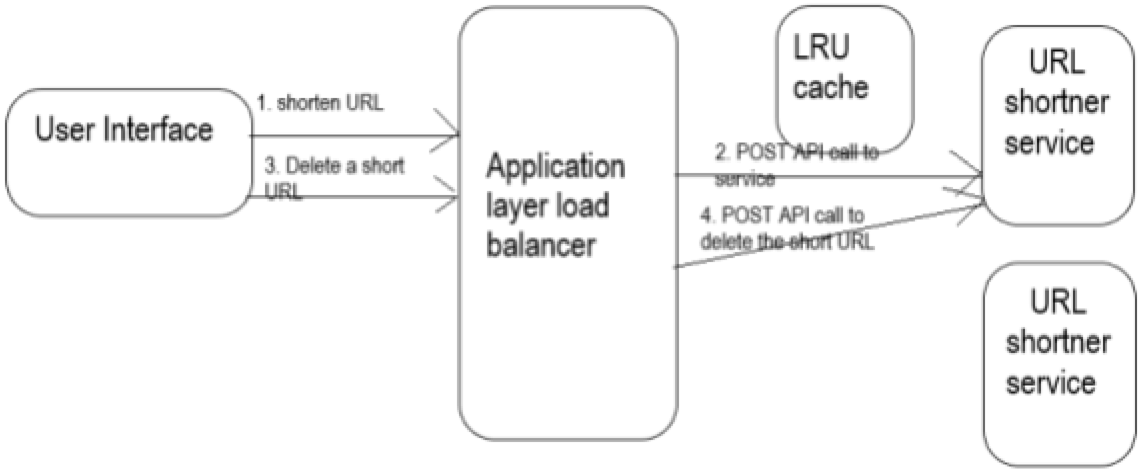
Environment -
Go1.22.0
Visual Studio Code
Github
Git bash CLI

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://www.cloudflare.com/cybersecurity/ can be converted to short URL like https://myshorturl.com:8080/azxd60pa
Following is the format and purpose of the Rest API's -

1. **Generate a short URL** - The API call will look like localhost:8080/shortenurl
2. **Delete short URL** - The API call will look like localhost:8080/deleteurl
3. **Redirect URL** - The API call will look like localhost:8080/redirecturl

To generate a short URL, the long URL will be encoded to the short URL.The encoded string will then be appended to localhost:8080 to create the short URL.

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


User Interface
1. shorten URL
3. Delete a short URL
Application layer load balancer
LRU cache
2. POST API call to service
4. POST API call to delete the short URL
URL shortner service
URL shortner service