Assignment 1.2 - RESTFUL Service

Web services and Cloud-based systems - 2020

Done by,
Haritha Jayaraman - 12975052
Kailainathan Muthiah Kasinathan - 12937827
Richard Bieringa -10691065

Design & Implementation of URL Shortener:

Python was chosen as language for Assignment 1 so we chose flask to build the service. For the design we used the HTTP response design given. This resulted in the following API design:

/:

GET -> Display all the links within the web service.
- Results in a list of links between full URLs and shortened URLs.

POST -> Allow the user to post a url (form encoded) to the server to create a short link, accepts the url form encoded value as data.
- 201 -> OK, response contains shortened URL.
- 400 -> URL posted is not a valid URL.

DELETE -> Allows the user to delete a url link from the server, accepts the url form encoded value as data.
- 204 -> Successfully deleted a link.
- 404 -> Could not find a link associated with the url.

/:id:

GET -> Redirects the user to the original URL behind the shortened URL.
- 301 -> Successful redirect.
- 404 -> Could not find a link for the specified shortened URL.

PUT -> Update or create a link for the specified shortened URL, accepts the url form encoded value as data.
- 400 -> Invalid url specified.
- 404 -> Shortened URL not found.
- 200 -> Successfully updated URL.

DELETE -> Allows the user to delete a url link from the server, accepts the url form encoded value as data.
- 204 -> Successfully deleted a link.
- 404 -> Could not find a link associated with the passed url.

The implementation of the flask REST server is able to serve multiple users, since the links between the shortened and original URLs are stored in memory on the server. Doing it this way the every user is able to access shortened URLs created by other users. This implementation choice was made since it was a simple solution to create a standalone service which also contains the data it needs to operate.

How to implement a URL-shortener for multiple users?

Our RESTFUL service stores the links between the shortened URLs in memory using a python dictionary. Since this is not bound to any specific user every user that accesses the server can utilise the shortened URLs created by other users of the service. The server contains its own state in this case and can serve the multiple users the same content.

This implementation however has some downsides:

- Once the server is restarted the state and therefore URLs mapped are lost
- If the service is deployed through a load balancer each server will have its own, different state. This resolves in clients accessing the service experience inconsistent behaviour.

A better approach to combat this would be to remove the state from the server's memory. By moving the links between the original URLs and the shortened URLs to a database it is possible to maintain the state of the links outside of the REST server, making it possible for the server to be completely stateless and upon restarting not be affected by a state reset. This would also help in a load balanced situation where each server will request the database for links making the user experience similar if assigned to a different server.

An even better proposal for the design is to implement caching of frequently used or recently accessed URLs to lower the load on the database containing the links and improve performance on the URL shortener service.