URL Shortening Service

Table of Contents

URL Shortening Service – Overview

URL Shortening Service Requirements

The service was built according to the specifications provided by Education First. The following functional requirements were made:

- The service supports unique users who authenticate themselves with by a password
- Each user maintains his/her own set of URL short forms for URLs
- The user can access information regarding the registered URLs – the information will include the number of visits and the date on which the short form was registered
- Interaction with the services is through REST based services over HTTP

The following none-functional requirements were made:

- The service had to be designed for scale
- There had to be a mechanism for setting the service locally in less than 30 minutes (assuming the user is technically savvy)

In this section we shall describe the technology stack chosen for this project and the way in which the project has been designed for scale.

Technology Stack for the Project

Falcon

The Falcon framework has been chosen the REST services. The Falcon framework allows for publishing of objects by implementing get_X methods – where X is one of the HTTP verbs (GET, PUT, POST, DELETE and OPTIONS).

The Falcon framework is extremely lightweight and easy to develop for. The Falcon framework does not include a web server and relies on a WSGI compatible web-server to serve the content. In this project the waitress WSGI server has been chosen.

MongoDB

MongoDB is a NoSQL database for document storage. MongoDB uses JSON internally which makes it a natural complement to REST based APIs. The database can be deployed in a stand-alone mode or in a cluster for increased performance.

Memcache

Memcache is a distributed key/value store. By storing frequently accessed information in RAM it is possible to speed up user requests by avoiding a database lookup.

In this application the memcache service is utilized for storing the mapping between the short form and the long form of the URLs.

Designing for Scale

A key requirement of the exercise has been that the project scales. To ensure scale the following have to be accomplished:

- No single component can become a bottleneck
- Components can be scaled up/down depending on the demand

The design of the URL Shortening service follows both principles:

- Each component in the stack can be scaled separately
- The REST component is stateless and can be deployed behind a load balancer
- The database component is offloaded by the memcache layer
- The database component can be scaled depending on the I/O needs

- The URL visits are written to a separate collection and aggregated on user request – this ensures that no single object becomes a hot spot in the database

Interacting with the Service

API End Points

The service has 3 API endpoints:

- The users end point published on /users
- The urls end point published on /urls
- The / end point used for redirection

The service employs the following conventions with respect to HTTP status codes:

Status Code	Used For
200	The operation completed successfully
302	Used for redirect only
400	The user submitted bad parameters or called the service with missing parameters
403	The user failed authentication and is not allowed to carry out an action
404	The requested resource was not found
500	Internal server error

The Users API

The users API is published using the /users URI and accepts the HTTP POST verb only. The service does not require authentication.

The users API exposes only a single method. It allows for users to register themselves to the service. The service expects the user registration information to be a payload in JSON format.

The payload consists of two fields: username and password. Below is an example of a legal payload:

```
{
        “username”: “testuser”,
        “password”:”mysecret”
}
```

The username is case sensitive and the service enforced the uniqueness of the username.

The URLs API

The URLs API is exposed over the /urls URI. It exposes 3 methods:

- A post method on the /urls URI for creating a new URI
- A get method on the /urls URI for retrieving the information regarding all the URLs a user has registered
- A get method on the /urls/url_short_form where URL short form is the short form registered by the user

The service maintains uniqueness of the short form of the URL across the entire user base. It is impossible to register the same short form twice.

When accessing the service the user needs to send the username and password using HTTP headers. The expected header names are username and password respectively. The password is not encrypted or modified in any form.

Get URLs Info/Get URL Info

These two services retrieve information from the database. The information displayed in both cases is the same.

Below is an example of valid output for a single URL:

```
{
    "shortform": "love",
    "url": "http://www.love.com",
    "visits": 0
}
```

Below is an example of valid output for all the URLs for a user:

```
[
    {
        "createtime": "2017-07-04 21:47:51.035474",
        "shortform": "love",
        "url": "http://www.love.com",
        "visits_count": 0
    },
    {
        "createtime": "2017-07-04 21:58:24.342310",
        "shortform": "python",
        "url": "http://www.python.org",
        "visits_count": 0
    }
]
```

The information presented for each URL is the same as for the single URL case. The results are wrapped in a JSON array for easier consumption by computer programs.

The Create URL Shortening API

This API exposes a single method call using the HTTP post verb. The user can interact with the API in two ways:

- Send both the long form of the URL and the short form of the URL

- Send the long form of the URL only – in that case the server calculates a hash of the concatenation of the long form and URL and takes the first 8 symbols as the short form

The service replies with the URL short form in the form of a message. This ensures that users who did not submit a short form know how to access their URLs.

The service expects the parameters to be sent in a JSON payload. Two types of payload are accepted:

- `{“longform”:”http://www.yahoo.com”,”shortform”:”yahoo”}`
- `{“longform”:”http://www.yahoo.com”}`

The Redirect API

The redirect API is exposed over the /shortform URI where short form is one of the previously registered short forms.

The service implements the redirect using the 302 HTTP status code. It sends the redirect location in the HTTP header Location.

Internally the service inserts a record of the visit to the database so it can later be used in aggregations.

Ideas for Improvement

The exercise was completed under time constraints. Had I been given more time I would have implemented the following:

- Collecting information from the HTTP headers of the incoming requests. Information regarding the device type used and the browser employed could be interesting
- Using the IP of the incoming request to geo locate a user using the MaxMind service
- Putting a small control panel in front of the database using Graafana so users could access statistics