Squeezer

A URL Shortener Service

Mit Swami - 25 January 2019

Table of Content

Introduction	3
API Requirements	3
Technical Specification	4
Database Design	4
Implementation	5
Design Methodology	6
API	7
Conclusion	8

Introduction

Squeezer is a URL shortener service. Sharing long urls via SMS or on twitter is bit difficult as there is character length restrictions. Because of that URL shorter service came into existence.

The URL shortener service is basically converts a long url into a small url. The short url have unique 6-7 characters long path value. When Squeezer API is called with long url it returns the corresponding short url. Now user can distribute that short url in SMS and emails. The short url generated by Squeezer contain default domain name “sq.z”. But Squeezer also provides an option for custom domain name. In that case the short url will generate using that custom domain name.

The URL shorter service also provides an analytics of daily usage of the service with the statistics about clicks on short urls and newly long url created.

API Requirements

The Squeezer API should have below requirements:

(1) When user POST a long-url along with an optional domain name, the API should generate a short url for that particular long url which is unique. And if the long url already exist then return the corresponding short url.
(2) When user send a GET request with short url, the API should search into database and return long url.
(3) When user send a GET request for a report of daily no. of clicks, the API should return the data of clicks of all short urls hits today.
(4) When user sends a GET request for a report of daily no. of new short urls created, then API should return the data of received long urls today for short url generation.
(5) Other than above the API should response quickly. To do that the GET method should implement a cacheing mechanism to avoid unnecessary database calls.

Technical Specification

The API is developed in Java.

- **Frameworks:** Spring MVC, Hibernate, Swagger
- **Database:** MySQL
- **Server:** Apache Tomcat
- **Caching Server:** Redis

Database Design

The database consist of two tables. (1) URLs (2) Clicks

(1) **URLs:** This table contain information about Urls.

- **Id:** It is auto incremented primary key.
- **Long url:** It is the original url.
- **Short url:** It is the unique short url of the original url.
- **Domain:** It is the domain name of short url. Default value of domain is "sq.z".
- **Creation date:** It is the date when short url is created for original long url.

(2) **Clicks:** This table contain information about no. of clicks short urls had on a particular day.

- **Short url:** It is same short url referenced to URLs table.
- **Click date:** It is the date on which the short url was clicked.
- **Count:** It is the count of how many times a short url was clicked on a particular date.

In second table, **(Id, Click date) -> Count** functional dependency is a BCNF. The combination of **(Id, Click date)** is unique for each tuple. So it is defined as composite primary key for Clicks table.

To make queries faster long url, creation date columns from URLs table are indexed.

Implementation

In Squeezer, there are two model class defined. (1) URL (2) Click. This two models is mapped to URLs and Clicks table in database respectively. Model classes provide a way to fetch or store data into database.

The UrlDao and ClickDao interface both extends JPA repository which provides easy way to execute query to the database with predefined functions. UrlDao is used to perform query on Urls table and ClickDao is used to perform query on Clicks table. A custom native query was implemented to retrieve data for report.

UrlController receives HTTP requests from server and handles each request in its handlers. The handler calls methods from UrlService class in which main logic of URL Shortener resides. The UrlService class have methods to generate a unique short url for a long url which does not collide with other urls short url. UrlService calls methods of UrlDao and ClickDao to store or fetch data from database.

To optimise the API for reducing database calls, we used Redis as caching server. The ***fetchLongUrl()*** method defined as cacheable. Redis uses short url as key and long url as value. If there is GET request for long url. The short url first checked by Redis in ***ShortToLongCache***. If corresponding key for that short url found then method wont be called. That saves a database call. Same as above ***fetchShortUrl()*** method also defined as cacheable and the name given to cache is ***LongToShortCache***. This method is used to check existence of long url in database when new request for long url received to generate short url.

In UrlService class, ***saveUrl()*** method is annotated as transactional because it calls database multiple times if there will runtime error in between then the whole process will be roll backed.

When we perform GET request for long url, Squeezer API also counts the click so that task is defined as async to avoid delay for the API response.

Other than this, Squeezer also provides a Swagger UI. Swagger UI provides visual documentation. It also used for testing the API.

Design Methodology

There are many URL Shortener services which uses different design and logic to generate a short url for a long url. The main focus for Squeezer API to provide such a way where the process of generating a unique short url which does not collide with other short urls and the process is of generating short url is very fast and efficient.

There are three popular strategies to generate short urls.

(1) Use cryptographic hashing function to hash long url. And take substring of some most significant bit and convert it into 6-8 character long alphanumeric string.
(2) Generate random string until there is a string which do not collide with other.
(3) Use the id which is auto incremented in database table and use that id to generate a base 62 number. (Base 62 because there are 26 upper case letters + 26 lower case letters + 10 digits = 62 characters to encode short url.)

Squeezer uses the third strategy to use auto incremented id and perform base conversion. The main advantage to use this strategy is that in other strategies there is possibility of collision. Where in third strategy there is always new number used so that the collision will never occur. There are 1000 billion urls on internet. But statistics show that very small amount of url compare to 1000 billion are used to generate short url. So the average possible length of short url will be 6-8 characters which is perfect to share the link to other people.

Other reason is that in first and second strategy, we have to expire links to avoid increasing collision as number of urls increases by the time and reuse it again. It is very bad idea to expire the short url as a user never expects that after a year short url redirects to different link as it was reused.

API

Squeezer provides five APIs:

(1) ***http://localhost:8080/Squeezer/url***

Http Method: POST

Input Type: JSON

Input: {

"longUrl":"www.proptiger.com/2348734",

"domain":"pt.com"

}

Output: pt.com/u

(2) ***http://localhost:8080/Squeezer/url***

Http Method: GET

Input Type: Url Parameter

Input: /url/shortUrl?pt.com/u

Output: www.proptiger.com/2348734

(3) ***http://localhost:8080/Squeezer/url/report/clicks***

Http Method: GET

Output Type: List of Clicks

Output: List of short url which are clicked today and count of how many times they are clicked.

(4) ***http://localhost:8080/Squeezer/url/report/clickOn***

Http Method: GET

Input Type: Url Parameter

Input: /url/report/clickOn?shortUrl=pt.com/u

Output Type: List of Clicks

Output: List of dates on which short url was clicked and count of how many times it was clicked on that date.

(5) ***http://localhost:8080/Squeezer/url/report/newUrls***

Http Method: GET

Output Type: List of Urls

Output: List if new urls that are generated today.

(6) ***`http://localhost:8080/Squeezer/url/report/newUrlCount`***

Http Method: GET

Output: Count of no. of url which are requested to generate a short url.

(7) ***`http://localhost:8080/Squeezer/swagger-ui.html`***

This url opens swagger ui. From here user can interact and test the different GET/ POST methods provided by API.

Conclusion

Squeezer is Url Shortener API which provides short urls for eternity without any collision. Squeezer is well optimised also at database layer and also at application layer. That has reduced the response time for API significantly. It also provides data and analysis which is very important for end user.