



**Tejas R Patil**

[tejas.patil1@proptiger.com](mailto:tejas.patil1@proptiger.com)  
9228222668

# ShortURL Documentation


January 25, 2019

## Overview:

URL shortening is used to create shorter aliases for long URLs. Short links save a lot of space when displayed, printed, messaged or tweeted. Additionally, users are less likely to mistype shorter URLs. In this project, I have created a RESTful API for transforming LongURL to ShortURL and ShortURL to LongURL along with some extra features such as Expiration, Custom Domain, Report etc.

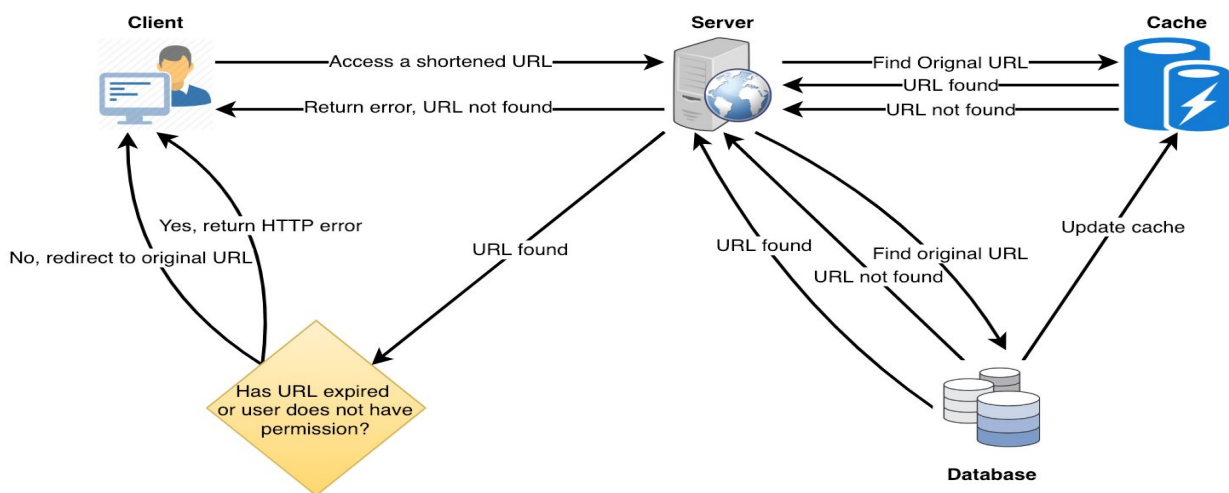
## Goals:

Main Goals of this Project is to implement a service which provides below-mentioned points :

- Given a URL, our service should generate a shorter and unique alias of it. This is called a short link.
  - When users access a short link, our service should provide them the original link.
  - Users should optionally be able to pick a custom domain for their URL.
  - Links will expire after a standard default timespan. Users should be able to specify the expiration time.
  - A report should be available consisting of different data.
- 

Overview of this project is to provide a RESTful API for performing above tasks. Therefore following request are to be implemented for the API.

1. **POST request to get ShortURL:** A POST request which takes JSON object including LongURL and with Optional Domain or Expiration Time(In Days) and which responds with JSON object consisting of ShortURL.
2. **GET request to get LongURL:** A GET request which takes shortURL as a parameter in the URL to which it responds with JSON object consisting of respective LongURL.
3. **GET request to get Report:** A GET request which retrieves a Report in a form of JSON object consisting of different data.



## Technology Specifications:

- Language : Java
- Framework : Spring
- Database : MySQL
- Server : Tomcat
- Caching store : Redis
- API documentation : Swagger

## Working :

### Mapping:

The most important thing in this project is the mapping between LongURL and ShortURL. For creating 7 letters shorter URL I had researched many methods such as:

- Random Generation
- Hashing the Longer URL
- Converting a counter to Base62 string

Finally, I have used a method of converting a counter to Base62 String because of the simplicity and guaranteed no collisions. As a counter, I have used the ID of the tuple of the LongURL in Database and converted it into Base62 string consisting of a-z, A-Z and 0-9. The main advantage of this method above other is that it makes retrieval of long URLs faster as ShortURL is nothing but the position of LongURL in the Database and as ID is always unique there is no point of collisions. It is also safe as a distributed service because we first save the Long URL into the Database and then create its ShortURL which avoids any race conditions. Also in other methods mentioned above, there are chances of collisions.

For Example

*Suppose a request for a new ShortURL comes for a given LongURL and after storing we get its ID as 3 then 3 is converted to 7 letters Base62 String that comes to be aaaaaad. Therefore ShortURL becomes aaaaaad and for finding respective LongURL we just convert this string back to Base10 Integer which gives us 3 the ID of LongURL in the database.*

## DATABASE:

I have used MySql RDBMS for this project and have created 2 tables as below.

<u>URL</u>		<u>Report</u>	
Name	DataType	Name	DataType
id	<i>bigint(20) AI PK</i>	id	<i>bigint(20) AI PK</i>
createdAt	<i>datetime</i>	shortURL	<i>varchar(255)</i>
domain	<i>varchar(255)</i>	domain	<i>varchar(255)</i>
expiresAt	<i>datetime</i>	Timestamp	<i>datetime</i>
longURL	<i>varchar(2100)</i>		
shortURL	<i>varchar(255)</i>		

Both tables have Primary Key as ID which is auto incremented. Queries to work with the Database were written in Normal Query as well as Native Query too.

## CACHING:

I have used Redis for Caching purpose in my project. It caches the method providing JSONObject consisting of LongURL for a given ShortURL and keeps it for 5 minutes. Before caching it also verifies whether JSONObject is null or not.

## SWAGGER:

I have used Swagger for API documentation purpose. Its Documentation can be viewed on <http://localhost:8080/shortURL/swagger-ui.html#/>

## API:

**url-shortner : URL Shortner**
[Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

GET	/urls	<a href="#">View URL's Info mapped to the provided short URL</a>
POST	/urls	<a href="#">Provide LongUrl along with optional parameters to be shorten.</a>
GET	/urls/report	<a href="#">View the Report of the URL Shortner Service.</a>

[ BASE URL: /shortURL , API VERSION: 1.0 ]

<u>GET /urls</u>	
Parameter	Returns
ShortURL	JSONObject[LongURL,ShortURL,expiresAt,Domain]
<u>POST /urls</u>	
Parameter	Returns
JSONObject[LongURL, Optional Domain and ExpiresAt]	JSONObject[LongURL,ShortURL,expiresAt,Domain]
<u>GET /urls/report</u>	
Parameter	Returns
None	JSONObject[ TotalGETRequest, TotalPOSTRequest, TotalGETRequestinLast24hours, TotalPOSTRequestinLast24hours, MostRequestedURL, CountOfMostRequestedURL, MostRequestedDomain, CountOfMostRequestedDomain, MostProvidedDomain,CountOfMostProvidedDomain, Timestamp]

### Code Flow:

- GET /urls [ <http://localhost:8080/shortURL/urls?url={shortURL}> ]

On getting above GET request Controller calls a method in Service where 7 digit string is extracted from the provided URL and converted to a Integer and finally URL object is returned by searching through this Integer i.e ID in the Database. This method is also enabled for caching with the condition that the returned Object is not null so that URL is not mapped with an empty object in cached. Then controller checks whether URL is null or expired and returns with appropriate Error otherwise it calls another Async method of Service to update database and returns the URL object.

- POST /urls [ <http://localhost:8080/shortURL/urls> ]

On getting above POST request Controller calls a method in Service where first the object is saved into the database and then id provided by database is converted to 7 digit base62 string. Using this string along with provided domain a URL is created and saved to the database and copy is returned to the User.

- GET /urls/report [ <http://localhost:8080/shortURL/urls/report> ]

On getting above GET request Controller calls a method in Service where various data such as MostRequestedDomain, MostRequestedURL etc are extracted from the database and returned to the user in the form JSON object.

```

1  {
2    "totalGETRequest": 42,
3    "totalPOSTRequest": 13,
4    "totalGETRequestinLast24hours": 42,
5    "totalPOSTRequestinLast24hours": 13,
6    "countOfMostRequestedURL": 27,
7    "countOfMostRequestedDomain": 42,
8    "timestamp": "Fri Jan 25 10:04:01 IST 2019",
9    "countOfMostProvidedDomain": 9,
10   "mostRequestedURL": "https://www.hos.com/aaaaaaj",
11   "mostProvidedDomain": "tp.pt",
12   "mostRequestedDomain": "hos"
13 }
```

## Models:

<u>URL</u>	<u>Report</u>	<u>ReportFormat</u>
id	id	TotalGETRequest
longUrl	shortUrl	TotalPOSTRequest
shortUrl	reqTime	TotalGETRequestinLast24hours
createdAt	domain	TotalPOSTRequestinLast24hours
expiresAt		MostRequestedURL
domain		CountOfMostRequestedURL
		MostRequestedDomain;
		CountOfMostRequestedDomain
		MostProvidedDomain
		CountOfMostProvidedDomain

## Conclusion:

With the use of 7 letter Base62 String,  $62^7$  i.e. **3,521,614,606,208** unique short URLs can be provided which can last for 111 years if 1000 request per second for new url is fulfilled. Even if it exhausts we can either increase the string to 8 letter or we can remove earlier mappings.

Finally I am very thankful to the mentor Ritu Yadav and Kanav Kalra for helping and providing this wonderful opportunity.