VERSION 1.0

JULY 24, 2017

TinyUrl - URL Shortener!

TINYURL PROJECT REPORT

PRESENTED BY: HIRANAVA

GRADUATE STUDENT
GAINESVILLE FLORIDA

TABLE OF CONTENTS

PROJECT DESCRIPTION

This project has three main aspects

- it takes input a long URL and converts it to a short URL.
- it takes a short URL and provides the original URL.
- And a short URL can be added or removed to a blacklist.

Blacklisted URLs cannot be used to get the original URL until it is removed from the blacklist.

We have designed a simple user interface which interacts with the RESTful APIs in the backend to complete this project. We have covered most of the requirements provided in the Hackerrank question.

SOFTWARE REQUIREMENTS

Below are the software's used for this project and are required to successfully run it.

USER INTERFACE

- HTML 5
- JavaScript
- jQuery
- Bootstrap
- AJAX

REST API'S

- Java (JRE 1.8 & JDK 8)
- Jersey
- External libraries – JSONParser, UrlValidator

JUNIT TESTCASES

- External Library – HTTPClient

BUILD

- Maven

DEVELOPER TOOLS

- Eclipse Neon
- Postman
- TomCat

OPERATING SYSTEM

- Windows 10

SCOPE OF PROJECT

1) A simple UI which can perform 3 operations
    a. Accepts a long URL and shows the Short URL
    b. Accepts a short URL (created previously) and returns the long URL
    c. Blacklists a short URL
    d. Removes a Short URL from blacklist.
2) RESTful API with 3 basic functions
    a. Encode a long URL to generate a short URL and send it back to user.
    b. Decode a short URL to get back the long URL
    c. Blacklisting or removing a short URL from blacklist

The Java code using Jersey has capability of basic validations along with the basic functionalities.

3) JUnit test cases checking different operations mentioned above and different failure scenarios.

The Encoding and Decoding algorithm used here can encode 62^6 URLs.

The Short URL generated always has a length of 25 characters only.

The model is scalable as we use hashmaps. It could have been made more scalable by using a database which we discuss in detail in the future scope section.

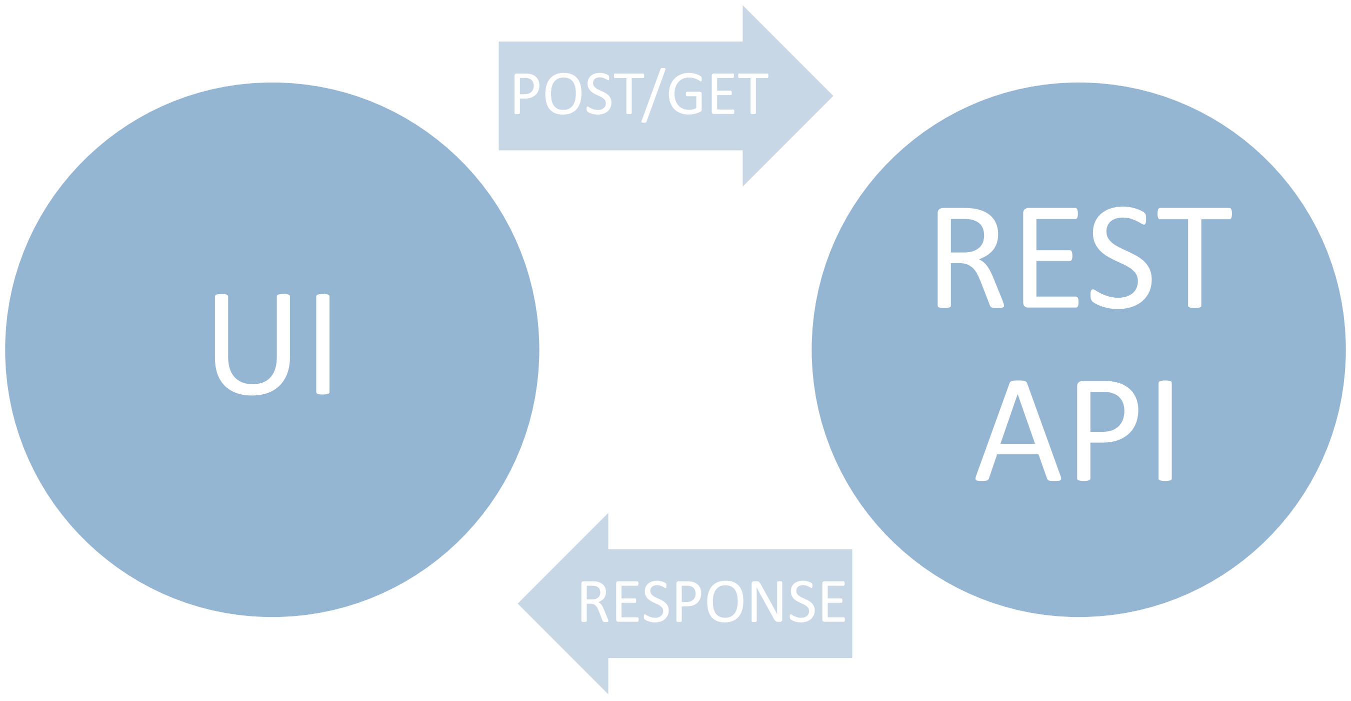
Duplicity is handled in this project, if same long URL is sent different short URL is not generated.

BUILD & RUN INSTRUCTIONS

Steps to build the project.

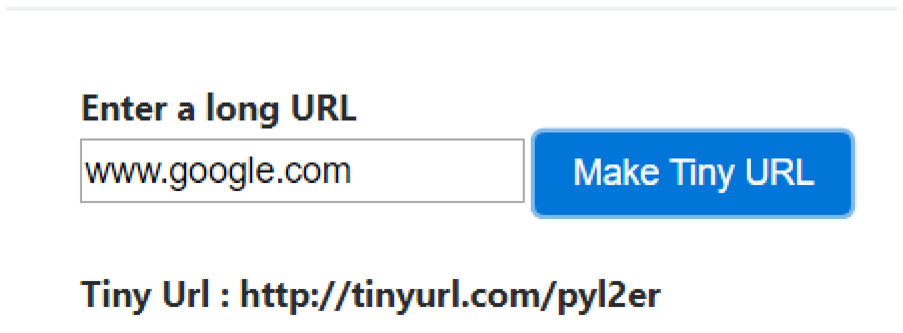
1. Unzip the project file.
2. Start the TomCat server.
3. Open a command prompt window and move to the project folder.
4. Build using the command "mvn clean install"
5. Copy the war file generated in the target folder to TomCat webapps folder and paste it.
6. To run the Junit test cases, modify the properties file and set the url value to http://localhost:8080
7. In your browser open "http://localhost:8080/tinyurl/home.html to go to the home page

PROJECT ARCHITECTURE



USER INTERFACE

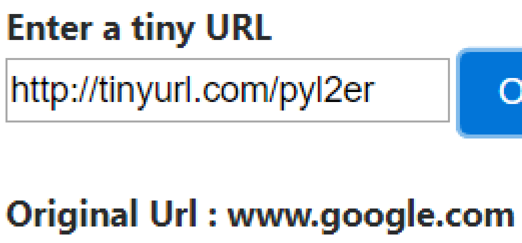
1. **Make Tiny URL**: Type in your long URL in the provided textbox and press the button to get the short URL in the area below the area below the text box.



Tiny Url : http://tinyurl.com/pyl2er

FIGURE 1MAKE TINY URL

2. **Original URL**: To get the original URL from the short URL type in the previously generated short URL and press the button to get the original URL back shown in the span below the textbox.

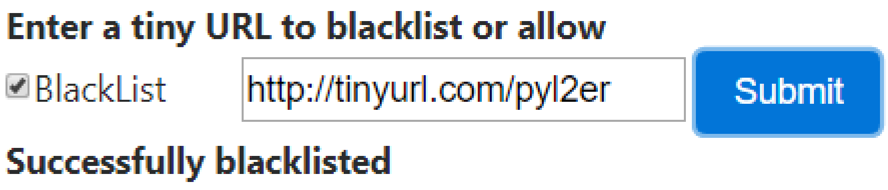

Enter a tiny URL

http://tinyurl.com/pyl2er

Original Url : www.google.com

FIGURE 2GET ORIGINAL URL

3. **Blacklist URL:** To blacklist a URL select the checkbox and enter the previously generated short URL and press the button. You can see the confirmation below.


Enter a tiny URL to blacklist or allow

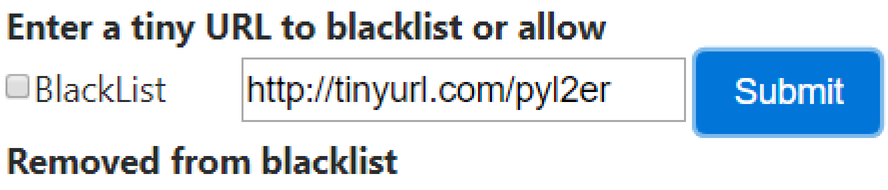
[✓] BlackList

http://tinyurl.com/pyl2er

Successfully blacklisted

FIGURE 3 ADD SHORT URL TO BLACKLIST

4. **Remove URL from blacklist:** To remove a URL from blacklist keep the checkbox unchecked and enter the short URL and press the button. You can see the confirmation below.


Enter a tiny URL to blacklist or allow

[ ] BlackList

http://tinyurl.com/pyl2er

Removed from blacklist

FIGURE 4REMOVED FROM BLACKLIST

REST APIS

1. **/encode** : This handles the post request from the users and consumes JSON file of the format given below.

```
{
    urlName: http://mockurl.com
}
```

If URL sent is null, not valid URL or empty string it sends an HTTP error code 400 i.e. BAD_REQUEST. Otherwise it generates a key which is concatenated with the string http://tinyurl.com/ and send back a status code of 201 i.e. CREATED.

2. **/decode :** This handles the GET request from the UI to return the Original URL. The Short URL is accepted by query parameter. If Short URL sent by user is not found in hashmap an error code of 404 is sent with an alert message in the response.
3. **/blacklist :** This handles the POST request from client to blacklist a URL or remove a URL from blacklist. It consumes an JSON file as input in below format.

```
{
    url:"http://mockurl.com",
    status:false
}
```

If URL sent is not found an error message is sent back in response with error code 404. If URL was already in blacklist a message is sent to notify user with HTTP code 200. Similarly, if the short URL sent was not present in blacklist and still the user tries to remove it from blacklist a message is sent with HTTP status code 200.

TEST CASES

JUnit test cases are given below:

1. **allGreenScenario** : Tests an end to end green path scenario.
2. **blacklistURLNotFound** : Test the scenario when URL given to blacklist is not present in hashmap.
3. **blacklistURLIsMalformed:** Test the scenario where request sent to blacklist a URL is malformed.
4. **urlNotFoundInDecode**: Test the scenario where short URL is not present in hashmap.
5. **badRequestFormedInEncode**: Test the scenario where encode request from client is not proper
6. **checkDuplicacy**: Check if two long URLs are same then whether same short URL is returned.
7. **checkBlankEncode**: Check if URL in encode request is blank.
8. **emptyOriginalUrlenScenario**: Test the scenario where Original URL sent to encode is empty.
9. **inValidOriginalUrlenScenario**: Test the scenario where invalid long URL is provided to encode.

OPEN ISSUES

1. More strict validations can be added both in UI and server side.
2. More test cases can be written to test various functionalities.

ALTERNATIVE SOLUTIONS AND THEIR PRO'S AND CON'S

Alternative solutions possible are given below with their respective pro's and con's

- Simple Integer could have been used as key

  PRO's

  - Simple Coding

  CON's

  - Number of unique keys would be much less and be limited by maximum integer value.
  - The code generated is very predictable and may cause security issue.
  - The length of URL will vary.
- Hash function could have been used to generate the key.

  PRO's

  - Hash function automatically generates a unique key.

  CON's

  - Probability of collision is high and increases steeply with increasing number of URL's.
  - Since Hash function uses integer calculation this is also limited by maximum integer value.
- Using a increasing counter and use that to get characters from a predefined string containing all characters from a-z, A-Z and 0-9 by selecting the character present at counter modulus 62.

  PRO's

  - Uses all the characters available.
  - Is cheaper than a Hash or Random function

  CON'S

  - This method is also limited by the range of integer as keys would get repeated after that.
  - Code generated can be predicted
- Using database to store the Long URL, short URL and an ID.

  PRO's

  - The ID could be an auto increment field and can be used to get the unique number every time which can then be coupled with the encoding decoding of process 3.
  - No hashmaps would have been required one table would have been enough.

  CON's

  - Would need an extra resource.

FUTURE SCOPE

The future scopes of this project are as below:

1. A better UI with more validations.

2. More unit test cases.
3. Implementation of authorization token to increase security
4. The blacklist functionality could have been made with a secret password so that only a few people can use that functionality.

REFERENCE

1. http://tinyurl.com/
2. https://en.wikipedia.org/wiki/TinyURL
3. https://stackoverflow.com/questions/742013/how-to-code-a-url-shortener