# Wing URL Shortener

## Brief Introduction

This serverless single-page application (SPA) supports generating short URL based on a specified URL. After signing in with Google account, the users can create their own specific short URLs, and they can even check statistical information of the URLs they had generated, like the creation time of a short URL, and how many times the URL had been clicked since after creation.

The magic behind this application is Google Shortener API:

<https://developers.google.com/apis-explorer/#p/urlshortener/v1/>

This application does not rely on any other server or service except Google Shortener API.

## Limitation

Whenever you generate a short URL, or browse the URL history that you have generated, you are triggering a Google API invocation from this application. For the URL Shortener API, Google sets the quota as 1,000,000 requests per day, and 100 requests per 100 seconds per user.

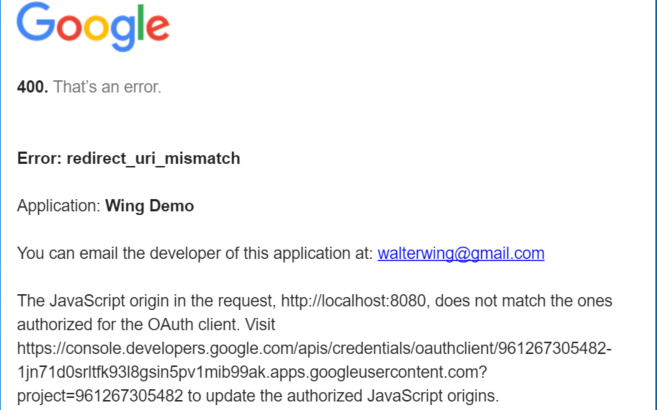
The quota should be enough for experiments and development, just be careful if you are going to test this application with automation scripts.

## Environment Setup

This application is easy to deploy since it only consists of plain HTML, CSS, and JavaScript files, and it imports dependent libraries (jQuery, Bootstrap, and AngularJS) via CDNs during runtime.

You may deploy this application in any server you like, just notice that your hostname should be “localhost”, and your port number should be 8080, 9080, or default (80). That is because Google requires specifying the origin URI (including port number) of the client application for OAuth 2.0 credentials. Currently I only set “http://localhost:8080”, “http://locahost:9080”, and “http://localhost” as the allowed origins, and I have tested the first two with my Tomcat and Liberty server respectively.

If you failed to meet above requirement, you will encounter below error when you are trying to sign in this application:



You may notice that the above error complains about invalid origin of “http://localhost:8080” – that was before I including it in my credentials.

Suppose you deployed this application in Tomcat with default configurations, and your context root is “WingURLShortener”, the entrance of the application will be:

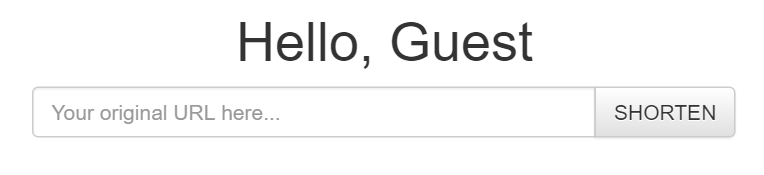
<http://localhost:8080/WingURLShortener/index.html>

## Features

Now let me introduce the key features of my application.

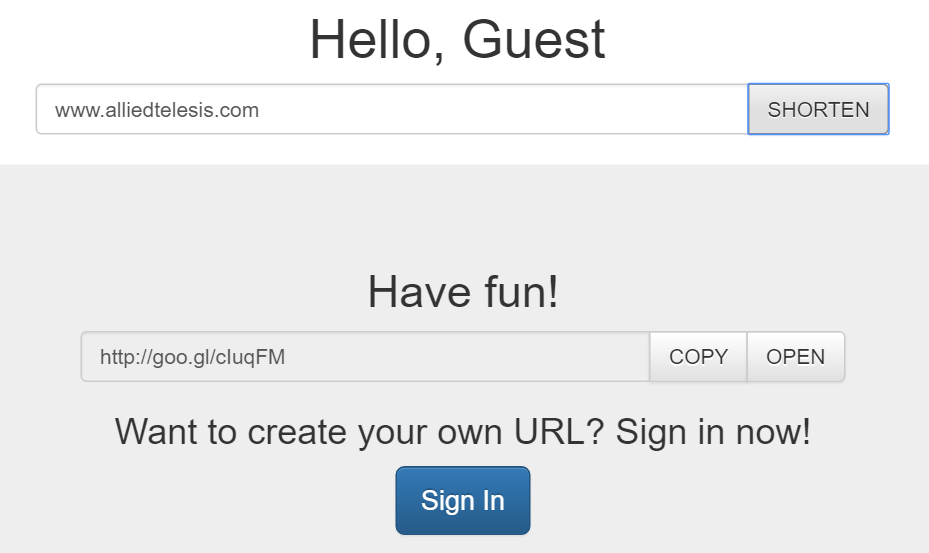
### Generate Short URL (as a Guest)

The first time you open the application, you will see the page like below:



As you can see in above screenshot, the greeting message for the guest user is “Hello, Guest”, which is different from a signed in user (will be shown later).

After inputting your URL, and click the “SHORTEN” button, the generated short URL will be shown as below:



As you can see the short URL had been generated. You can either copy the short URL by clicking the “COPY” button, or open the short URL directly by clicking the “OPEN” button.

Since you have not signed in yet, there will be a message encouraging you to sign in and experience advanced features.

As a guest user, the generated short URL for a certain URL will always be the same, no matter how many times you click the “SHORTEN” button. As a signed in user, however, the generated short URL will be changed every time you click the “SHORTEN” button (will be shown later).

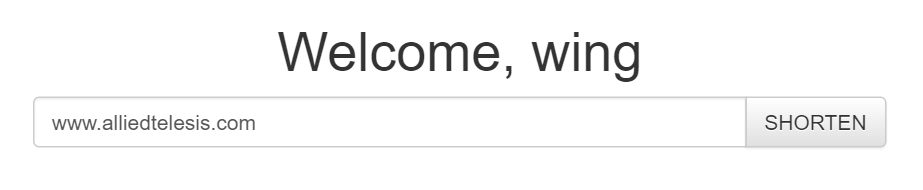
If you click on the “History” button on the navigation bar as a guest user, you will receive the following message:



OK, basically that’s all you can get as a guest user. Let’s continue to the features for signed in user. I’ll bypass the sign in/out features as they are self-explanatory. Have a try singing in with your Google account ☺

### Generate Short URL (as a signed in user)

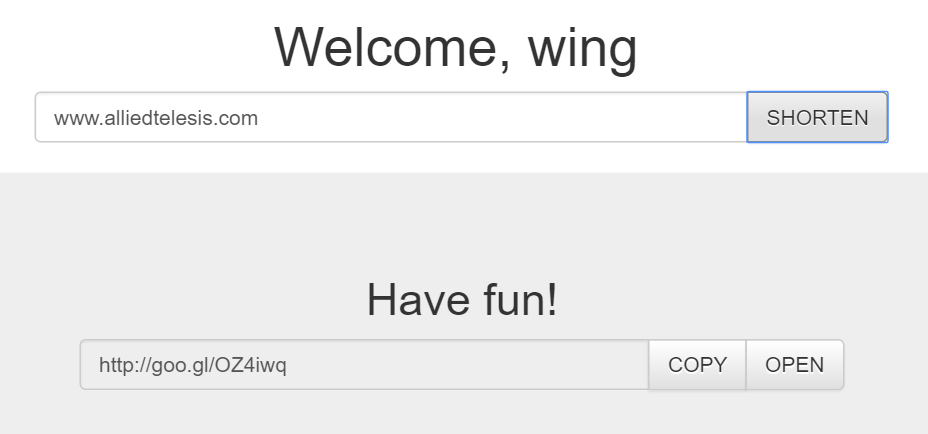
Now I have signed in with my Google account, the page will look like below:



You may notice that there are two differences:

1. The greeting message had changed to “Welcome, wing”, where “wing” is retrieved from Google profile through Google API.
2. The “Jumbotron” area (where the generated short URL is displayed) is hidden. This is to avoid confusion, since the short URL generated by a signed in user will be different.

Now let’s click on the “SHORTEN” button again and see what will happen:

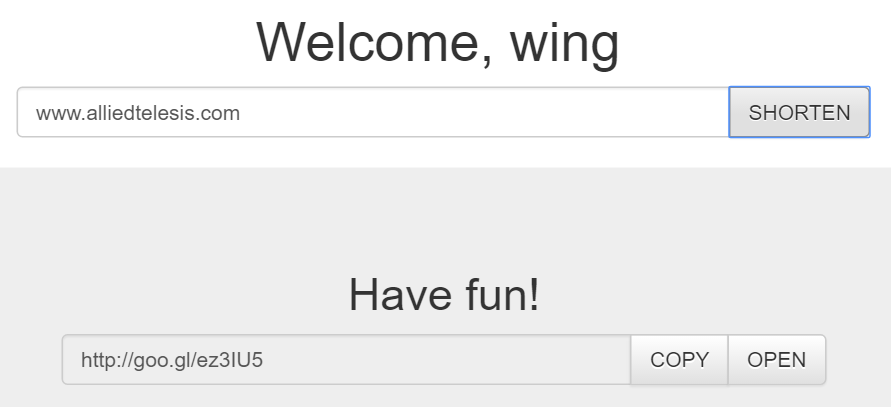


Apparently you will notice another two differences:

1. The generated short URL is different than previous one (generated by the guest user).
2. There is no extra message and sign in button any more.

Like before you can use “COPY” or “OPEN” button to copy or open the short URL as you need.

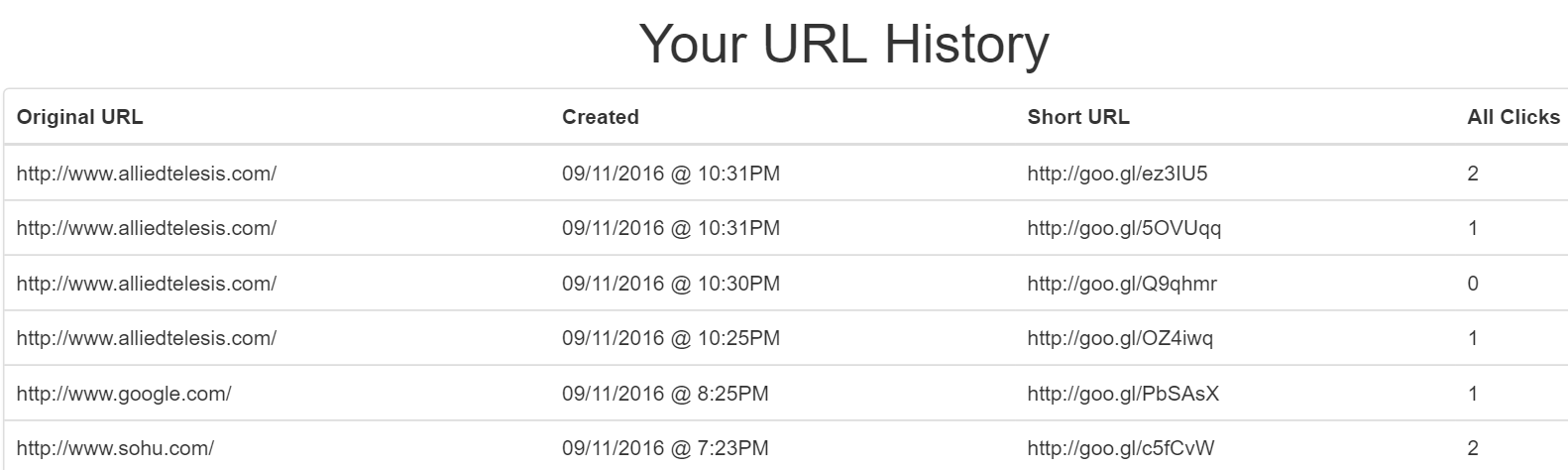
As mentioned before, one privilege of a signed in user is the ability to generate user specific short URL. Furthermore, every time you click the “SHORTEN” button, you will generate a different short URL:



Another privilege of the signed in user is to browse the URL history, so let’s move on to the next key feature.

### Browse URL History (as a signed in user)

Now let’s click on the “History” button on the navigation bar, and you will see the following page:



From the history you will know what you have done before, and how many times your URLs have been used.

OK, we have gone through all main features of this application, and as a cautious user, you will sign out from the website after using it. Just click on the “Sign Out” button on the navigation bar, and you will return as a guest user once again. Now you will see that the history page is requiring you to login, and when you navigate back to the home page by clicking on the “HOME” button on the navigation bar, you will see that the previously generated short URL is disappeared – so you won’t be confused about whether you generated the short URL as a signed in user or as a guest user.

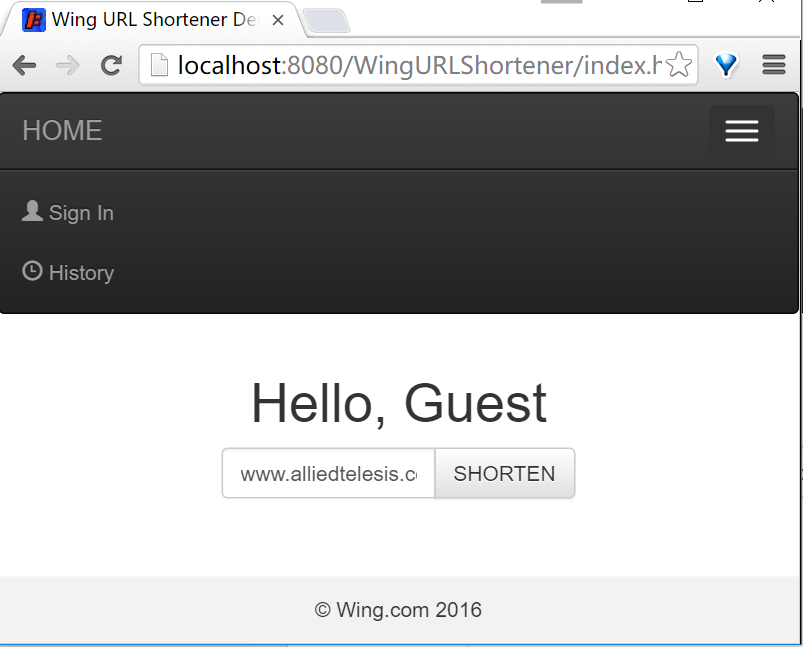
Next let me introduce two more non-functional key features – SPA and Responsive.

### SPA

As you play with this application, you will find that this is truly a single page application since no matter how you navigate between different functions (generate short URL or browse URL history), or switch your sign in status, you are staying in the same page, while seeing different contents dynamically.

### Responsive

This application is also responsive – you may have a try resizing your browser and see the effects:



## Known Issues

There are some know issues of this application.

1. Exception Handling

Currently very few exception flows are covered in this application. For example, if the user inputs some invalid URL deliberately, the Google API will return error and the short URL will not be generated. You will see errors in the console by using developer tools of the browser, but you will not get a hint in the web page.

If time allows I will review all the use cases, design exception flows, and make sure they are all covered.

1. Unit Tests

I did not write unit tests for my scripts. One reason is implementing those features already cost me most of the time, so I choose to implement basic functions first. Another reason is that I am not quite familiar with Jasmine yet. I am practicing Jasmine and Karma recently, and I had written some Jasmine specs for another project named “web-application-skeleton”, which is a simple web application skeleton I wrote by myself. I just started that project several days ago. It’s still the initial version and does not contain many contents yet. Actually I made use of the UI part of that project as the skeleton of this application, and it indeed saved me a lot of time.

I just uploaded two simple Jasmine specs that I wrote several days ago, so you may take a glance at the “web-application-skeleton” as well as those two simple specs:

<https://github.com/walterwing/web-application-skeleton>

1. Pagination Support

The Google API supports loading the URL history with pages. By default, Google URL Shortener API will fetch 30 entries at most for a request. Currently my application does not support pagination yet, so it can show 30 URL entries maximally.

If time allows I will enhance my service method and corresponding HTML to support pagination.

1. Globalization and Accessibility Support

I didn’t implement globalization and accessibility support due to time constraint. I have practical experience of globalization and accessibility support, since I was in charge of globalization and accessibility support in a project before.

If time allows I would use ng-translate to support globalization. While for accessibility, it’s kind of complicated since it covers too many aspects. In IBM we have an internal website that covers all aspects of Accessibility Support, and usually we will select a subset of those policies that match with our project. We also have a Firefox plugin named “Dynamic Assessment Plugin”, which can be used to check web pages automatically and detect potential violations of accessibility.

1. Sign In Directive

At first I did not anticipate that “Sign In” will be widely used in this application, and I ended up with repeating similar functions in different controllers and pages.

If time allows I would refactor my code to encapsulate Sign In function in an AngularJS directive.

1. Incomplete Initialization

During my development and test, I once encountered a problem that when I clicked the “Sign In” button, the console printed an error indicating that Google authentication instance had not been loaded yet. Unfortunately, that problem happened only once, so I am not sure whether the problem had been fixed unintentionally or is still hiding somewhere.

If time allows I will increase the latency time deliberately during the process of initializing Google client, and see if the problem can be reproduced.

## Remark

For those who want to have a try with my application but do not want to setup your own Google developer credentials, you can use my credentials temporarily. Please do not abuse my credentials in any case.

To use my credential, update js/demo.app.js to specify the API Key and Client ID as below:

urlShortenerProvider.setApiKey('AIzaSyCq5fBz2eo3gBgCpVM\_-CUu4OClLEe48ow');

urlShortenerProvider.setClientId('961267305482-1jn71d0srltfk93l8gsin5pv1mib99ak.apps.googleusercontent.com');