

Use Google OpenIDConnect OAuth2

OpenIDConnect
Google OAuth2

Abstract

Google offers a Spring-centric tool to provide OAuth2 protection, called OpenIDConnect. There are three form-factors that Google's OpenIDConnect OAuth2 product supports:

- Android devices
- iOS devices
- Browsers (User-Agent)

This document explores how to register the latter User Agent – a browser.

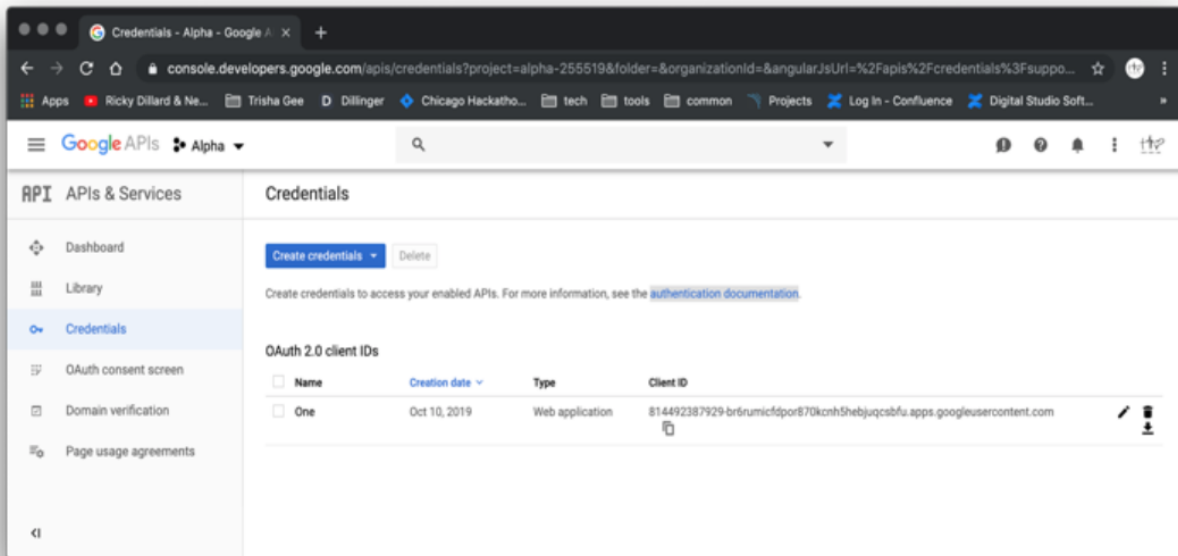
NB Because of the very nature of a browser, please clear cache for the browser on successive sessions “runs”. This assures that you are accessing the OpenIDConnect-protected application in a way that consistently presents a modal login (challenge-response) dialog box.

Setup, Example

The screen shot below shows a completed **Credential** entry for a web application in the Google Developers API Console.

Notice that the value in the **Name** column becomes the title of the challenge/response login page.

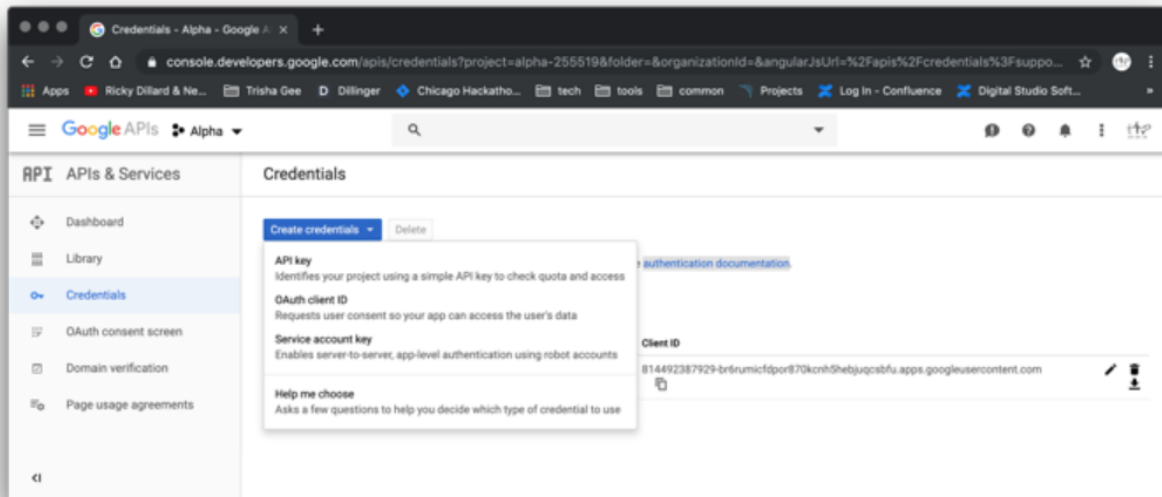
The value in the **Client ID** column below becomes the Key part of the K/V pair in the Spring's canonical `application.properties` file. This mapping, along with four other mappings, is shown later in this doc.



Setup, Process

The section illustrates how to create a **Credential** in Google's adaptation for OpenIDConnect OAuth2.

Below, choose the **OAuth client ID** option from the **Create credentials** pick list.

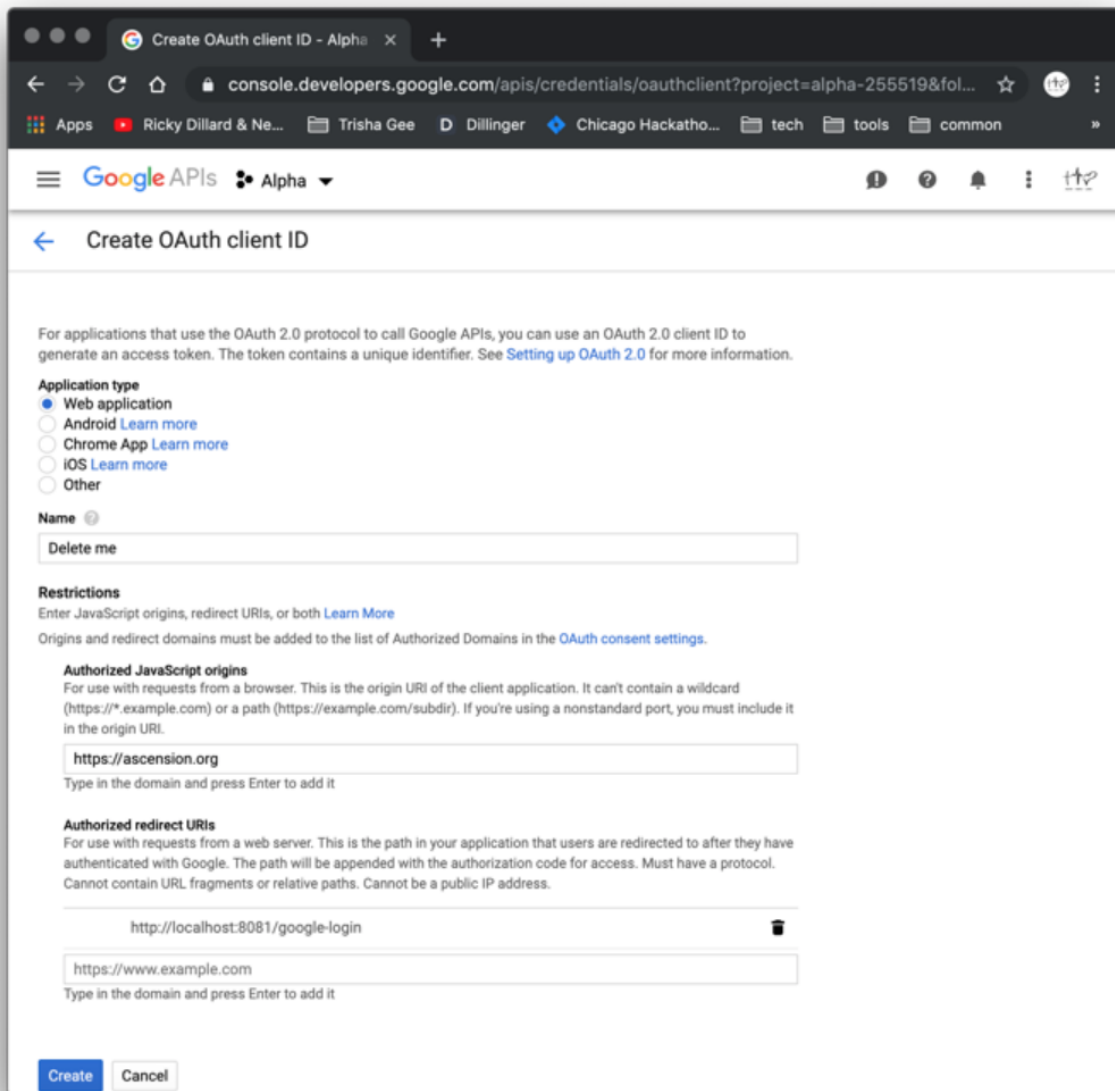


Below, choose the **Web application** radio button (option) in the **Application type** section.

Then, supply this value for the **Authorized redirect URIs** textbox:

<http://localhost:8081/google-login>

Click the **Create** button.



The screenshot shows the 'Create OAuth client ID' page in the Google Developers console. The browser address bar shows the URL: `console.developers.google.com/apis/credentials/oauthclient?project=alpha-255519&fol...`. The page title is 'Create OAuth client ID'. Below the title, there is a brief explanation of OAuth 2.0 client IDs. The 'Application type' section has five radio buttons: 'Web application' (selected), 'Android', 'Chrome App', 'iOS', and 'Other'. The 'Name' field contains 'Delete me'. The 'Restrictions' section includes instructions on adding JavaScript origins and redirect URIs. Under 'Authorized JavaScript origins', the text 'https://ascension.org' is entered. Under 'Authorized redirect URIs', the text 'http://localhost:8081/google-login' is entered. At the bottom, there are 'Create' and 'Cancel' buttons.

Create OAuth client ID - Alpha

← → ↺ ⌂ console.developers.google.com/apis/credentials/oauthclient?project=alpha-255519&fol... ☆ ⓘ

Apps Ricky Dillard & Ne... Trisha Gee D Dillinger Chicago Hackatho... tech tools common

Google APIs Alpha

← Create OAuth client ID

For applications that use the OAuth 2.0 protocol to call Google APIs, you can use an OAuth 2.0 client ID to generate an access token. The token contains a unique identifier. See [Setting up OAuth 2.0](#) for more information.

Application type

- ☒ Web application
- ☐ Android [Learn more](#)
- ☐ Chrome App [Learn more](#)
- ☐ iOS [Learn more](#)
- ☐ Other

Name

Delete me

Restrictions

Enter JavaScript origins, redirect URIs, or both [Learn More](#)

Origins and redirect domains must be added to the list of Authorized Domains in the [OAuth consent settings](#).

Authorized JavaScript origins

For use with requests from a browser. This is the origin URI of the client application. It can't contain a wildcard (`https://*.example.com`) or a path (`https://example.com/subdir`). If you're using a nonstandard port, you must include it in the origin URI.

https://ascension.org

Type in the domain and press Enter to add it

Authorized redirect URIs

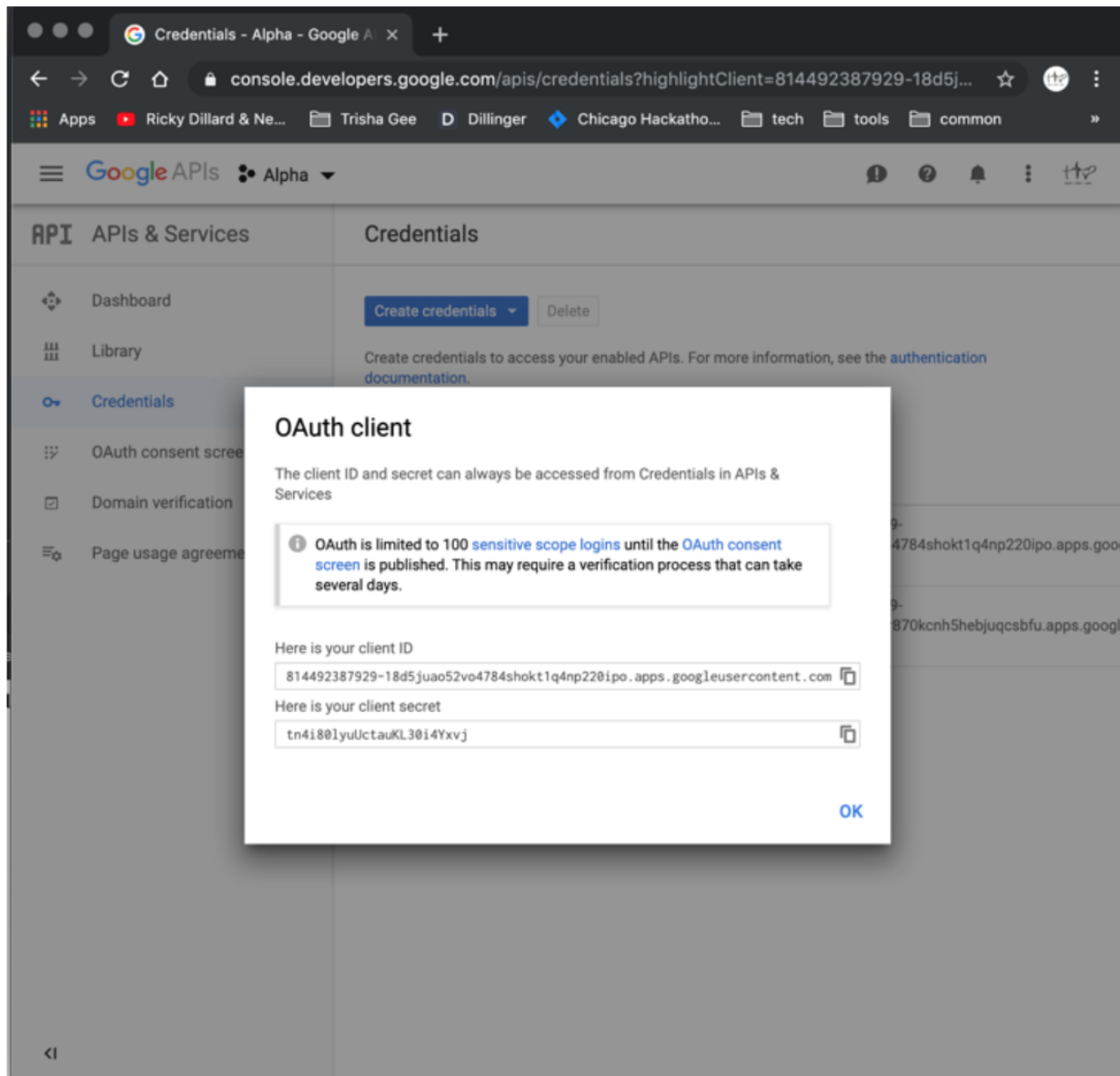
For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

http://localhost:8081/google-login

https://www.example.com

Type in the domain and press Enter to add it

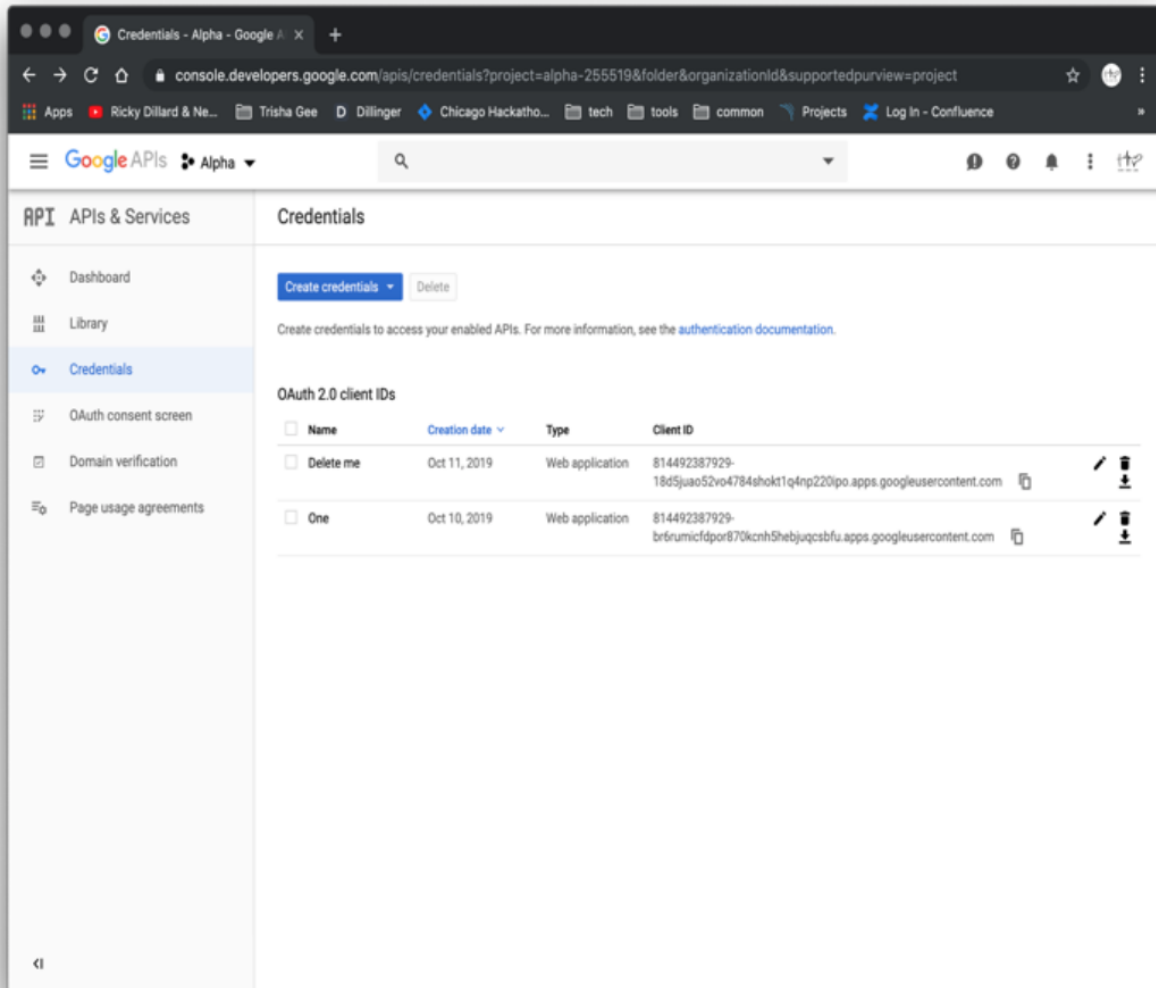
Create Cancel



The pair of values generated by the Google's OpenIDConnect OAuth2 product are used as entries in Spring's canonical `application.properties` file.

They are the [Client ID](#) and the [Client Secret](#).

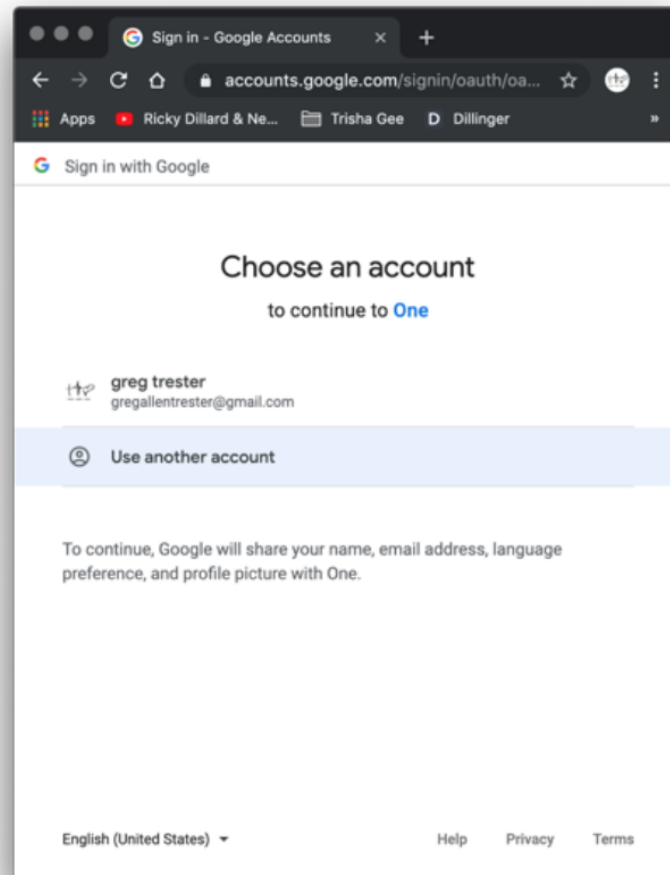
The screen shot below shows a second, completed [Credential](#) entry in the Google Developers API Console, for the web application that we just registered.

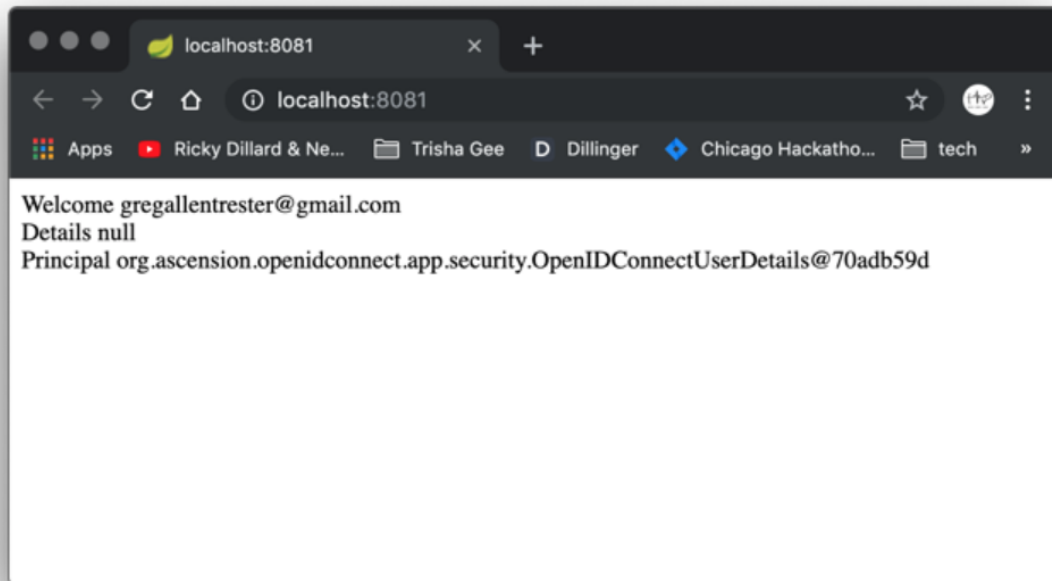


Usage

The browser URL/link for ingress into the OpenIDConnect-protected demo app is (by configurable design):

<http://localhost:8081/google-login>



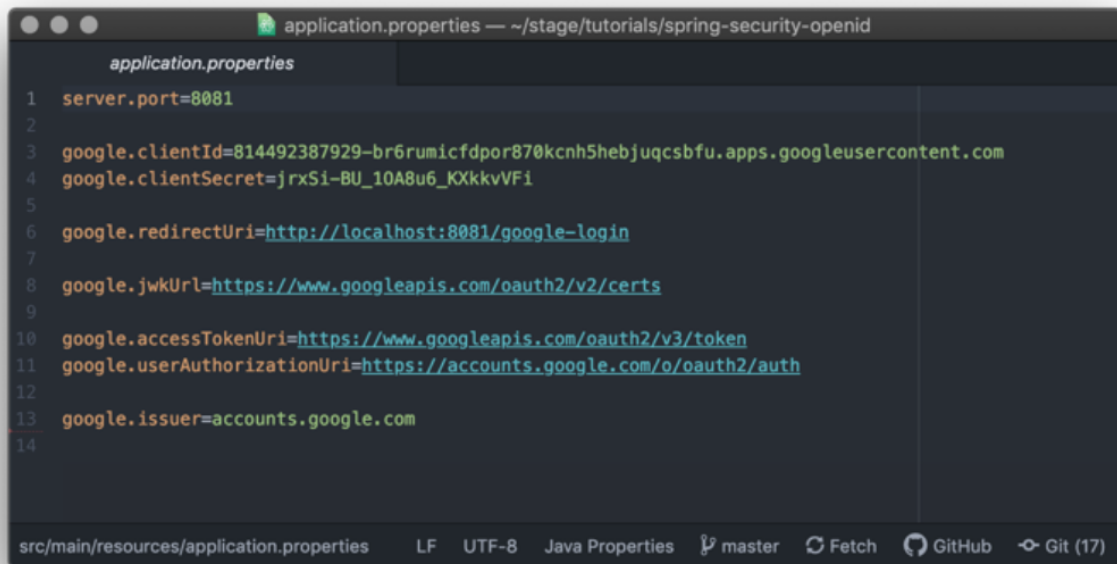


The Code

A Java application's part of the contract with OpenIDConnect is relatively unobtrusive.

The Spring Boot demo application that is discussed in this document uses a few annotations and some (5) entries in the canonical `application.properties` file.

NB In production, the `application.properties` file should have “locked-down” read only file permissions (e.g. 400).

A screenshot of a code editor window titled "application.properties — ~/stage/tutorials/spring-security-openid". The editor shows the contents of the "application.properties" file with the following lines:

```
1 server.port=8081
2
3 google.clientId=814492387929-br6rumicfdpor870kcnh5hebjuqcsbfu.apps.googleusercontent.com
4 google.clientSecret=jrxSi-BU_10A8u6_KXkkvVF1
5
6 google.redirectUri=http://localhost:8081/google-login
7
8 google.jwkUrl=https://www.googleapis.com/oauth2/v2/certs
9
10 google.accessTokenUri=https://www.googleapis.com/oauth2/v3/token
11 google.userAuthorizationUri=https://accounts.google.com/o/oauth2/auth
12
13 google.issuer=accounts.google.com
14
```

The editor interface includes a sidebar on the left showing the file path "src/main/resources/application.properties", a status bar at the bottom with "LF UTF-8 Java Properties", and Git integration icons for "master", "Fetch", "GitHub", and "Git (17)".

Notice above, that the values for the `clientId` and the `clientSecret` keys (lines 3 & 4, respectively) must match the values that were auto generated in the [Credential](#) section of the OpenID Console.

Build-Run

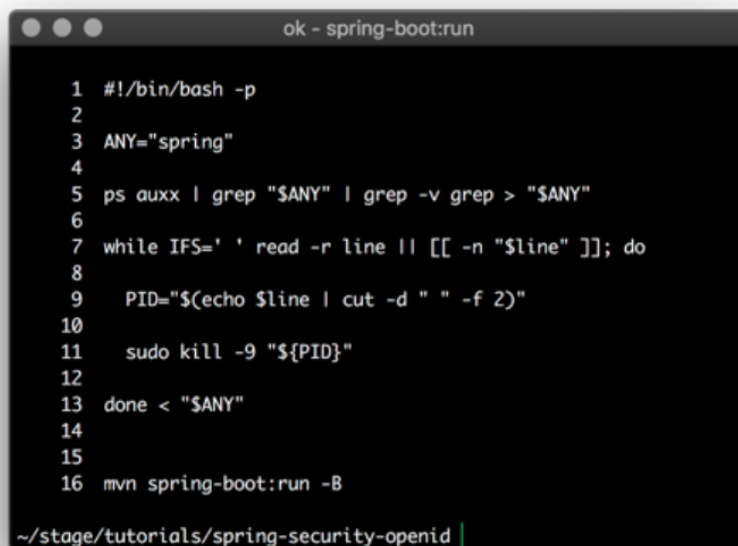
If you happen to have not stopped the Spring-boot application gracefully, the script show next will identify and kill your Spring-boot application.

Then, it compiles and executes the app by simply “sourcing” the script (w/o an extension, named:

`ok`

After killing the errant Spring-boot app, the `ok` script makes a delegating call to:

```
mvn spring-boot:run
```



```
ok - spring-boot:run

1 #!/bin/bash -p
2
3 ANY="spring"
4
5 ps auxx | grep "$ANY" | grep -v grep > "$ANY"
6
7 while IFS=' ' read -r line || [[ -n "$line" ]]; do
8
9     PID="$(echo $line | cut -d " " -f 2)"
10
11     sudo kill -9 "${PID}"
12
13 done < "$ANY"
14
15
16 mvn spring-boot:run -B

~/stage/tutorials/spring-security-openid |
```

BTW, you can modify Line 3 in the above script to point to a fragment-of-a-name of any runtime image/process (ps, or process status).

Invocation

The browser URL/link for ingress into the OpenIDConnect-protected demo app is (by configurable design):

<http://localhost:8081/google-login>

Addendum

These links originate with Google; however, they represent best-practices for the OAuth2 topic.

<https://developers.google.com/identity/protocols/OpenIDConnect#consentpageexperience>

https://developers.google.com/identity/protocols/OAuth2?hl=en_US

<https://openid.net/connect/>

<https://developers.google.com/identity/protocols/OpenIDConnect>

<https://github.com/eugenp/tutorials/tree/master/spring-security-openid>

<https://console.developers.google.com/apis/credentials?project=firstproject-98293&folder&organizationId>

<https://console.developers.google.com/apis/dashboard?project=firstproject-98293>

<https://developers.google.com/identity/protocols/OpenIDConnect#validatinganidtoken>