**OAuth 2.0**

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

OAuth defines four roles:

**resource owner**

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

**resource server**

The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

**client**

An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g.,whether the application executes on a server, a desktop, or other

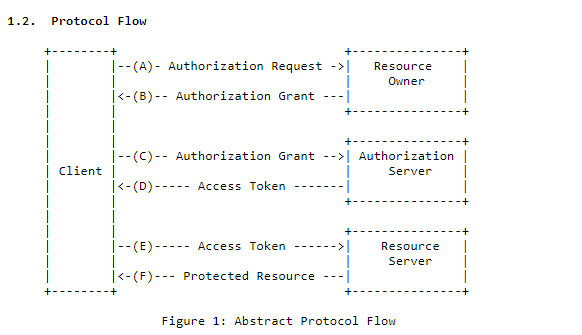
devices).

**authorization server**

The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The interaction between the authorization server and resource server is beyond the scope of this specification. The authorization server may be the same server as the resource server or a separate entity.

A single authorization server may issue access tokens accepted by multiple resource servers.



The abstract OAuth 2.0 flow illustrated in Figure 1 describes the interaction between the four roles and includes the following steps:

(A) The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the authorization server as an intermediary.

(B) The client receives an authorization grant, which is a credential representing the resource owner's authorization, expressed using one of four grant types defined in this specification or using an extension grant type. The authorization grant type depends on the method used by the client to request authorization and the types supported by the authorization server.

(C) The client requests an access token by authenticating with the authorization server and presenting the authorization grant.

(D) The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.

(E) The client requests the protected resource from the resource server and authenticates by presenting the access token.

(F) The resource server validates the access token, and if valid, serves the request.

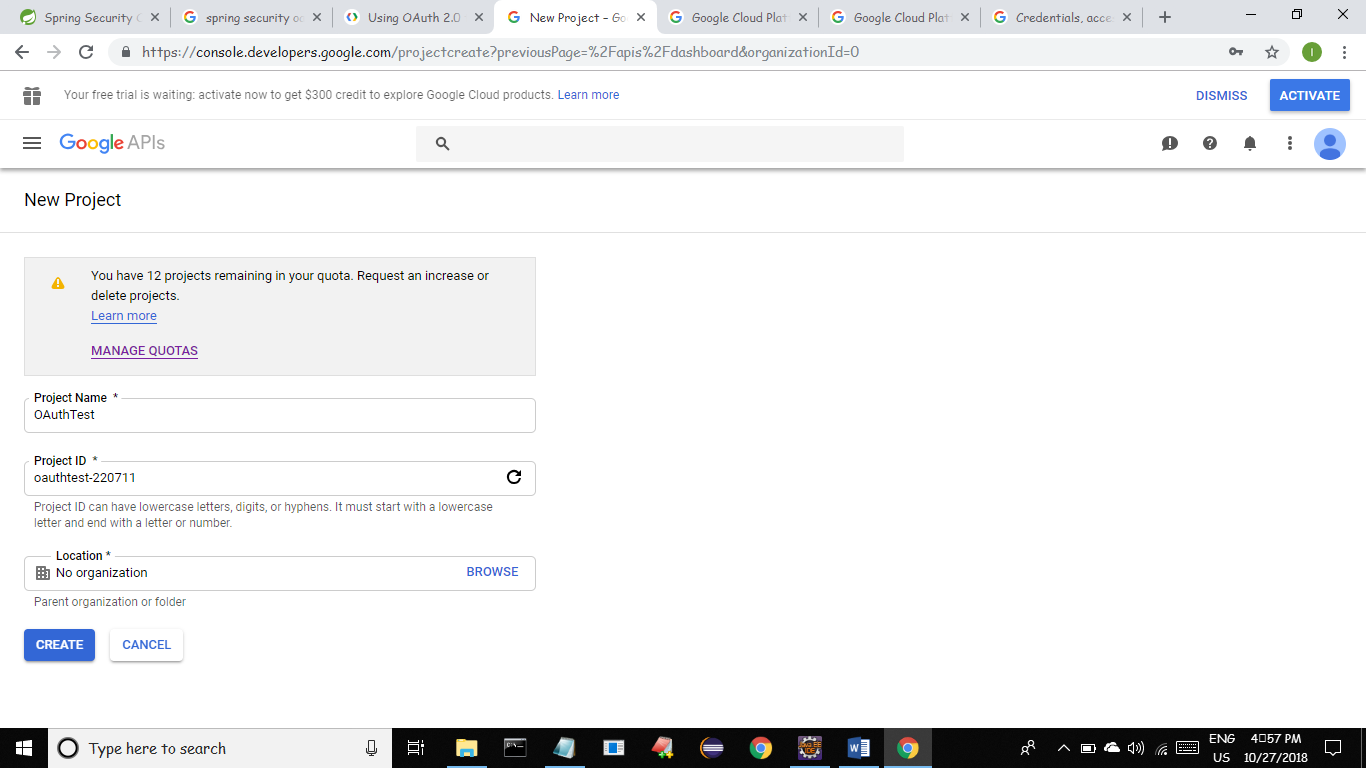
The preferred method for the client to obtain an authorization grant from the resource owner (depicted in steps (A) and (B)) is to use the authorization server as an intermediary

**Spring Boot and OAuth2**

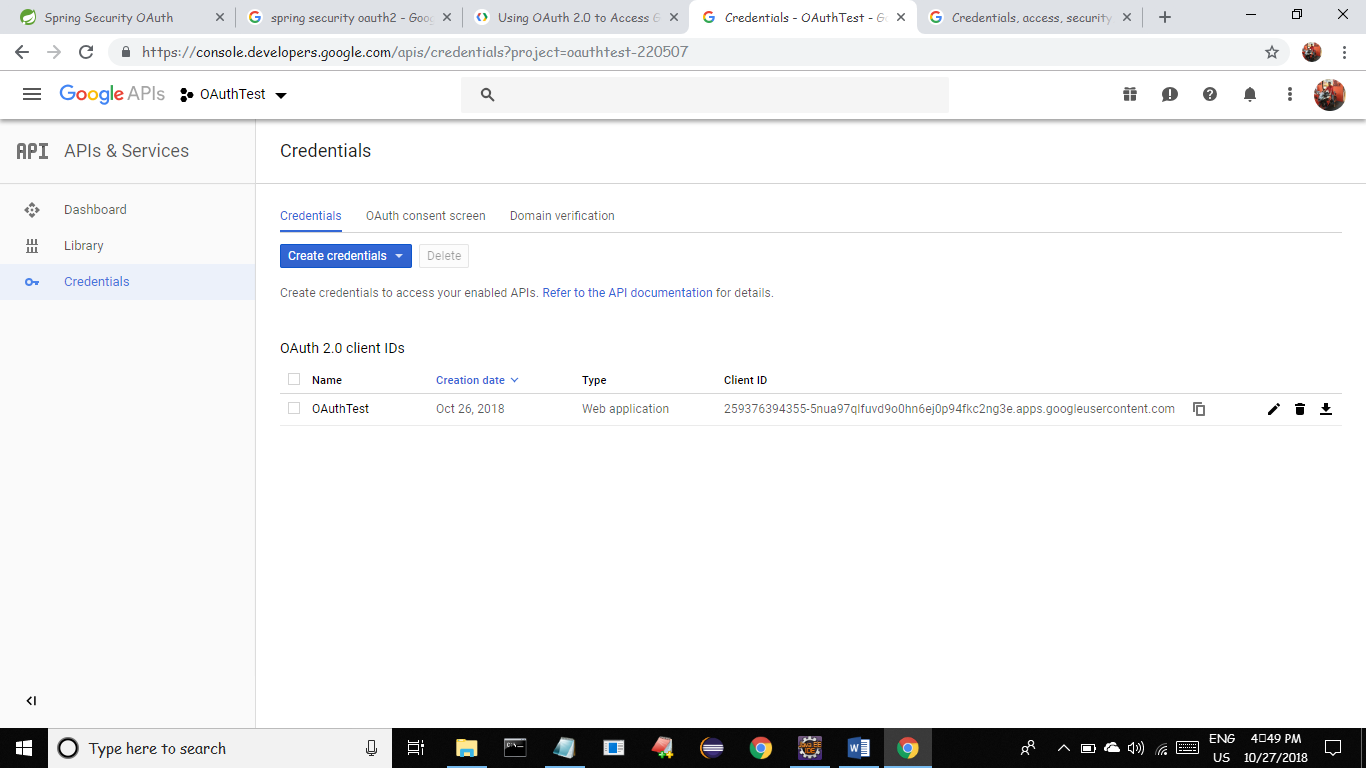
Spring Boot supports OAuth1 and OAuth2 to develop the application that can do various things with “social login”. It starts with a simple, single-sign on and works up to a self-hosted OAuth2 Authorization server with a choice of third party authentication providers like Google, Facebook and GitHub.

Example 1: Sign in With Google API(Gmail)

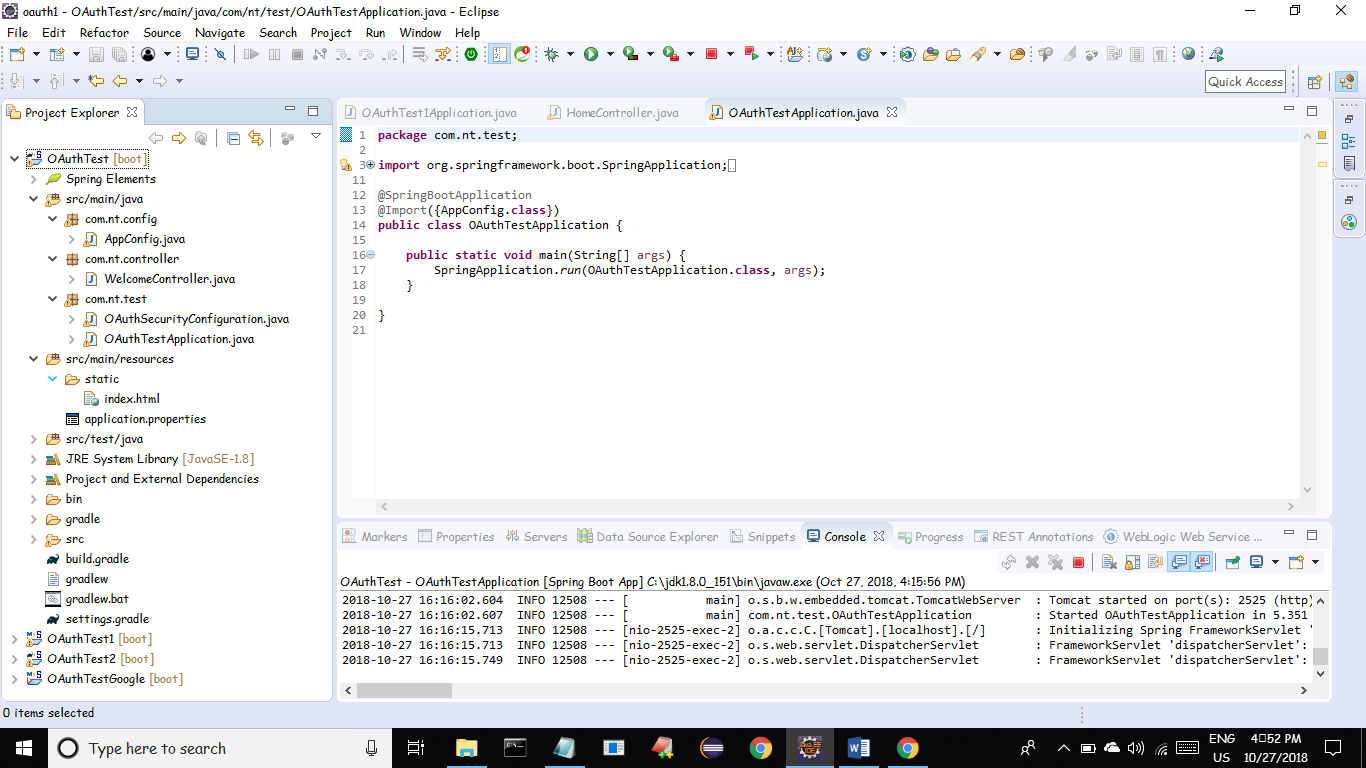
Visit *console.developers.google.com* and create a project



* Enable Gmail API and Create Client OAuth credentials such as Client id and secret by specifying redirect uri.
* This Redirect URI is useful to redirect after successful authentication.



* Create a Spring Boot Project with **Spring Security** and **Cloud OAuth2** Starters.



Index.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>OAuthTest</title>

</head>

<body>

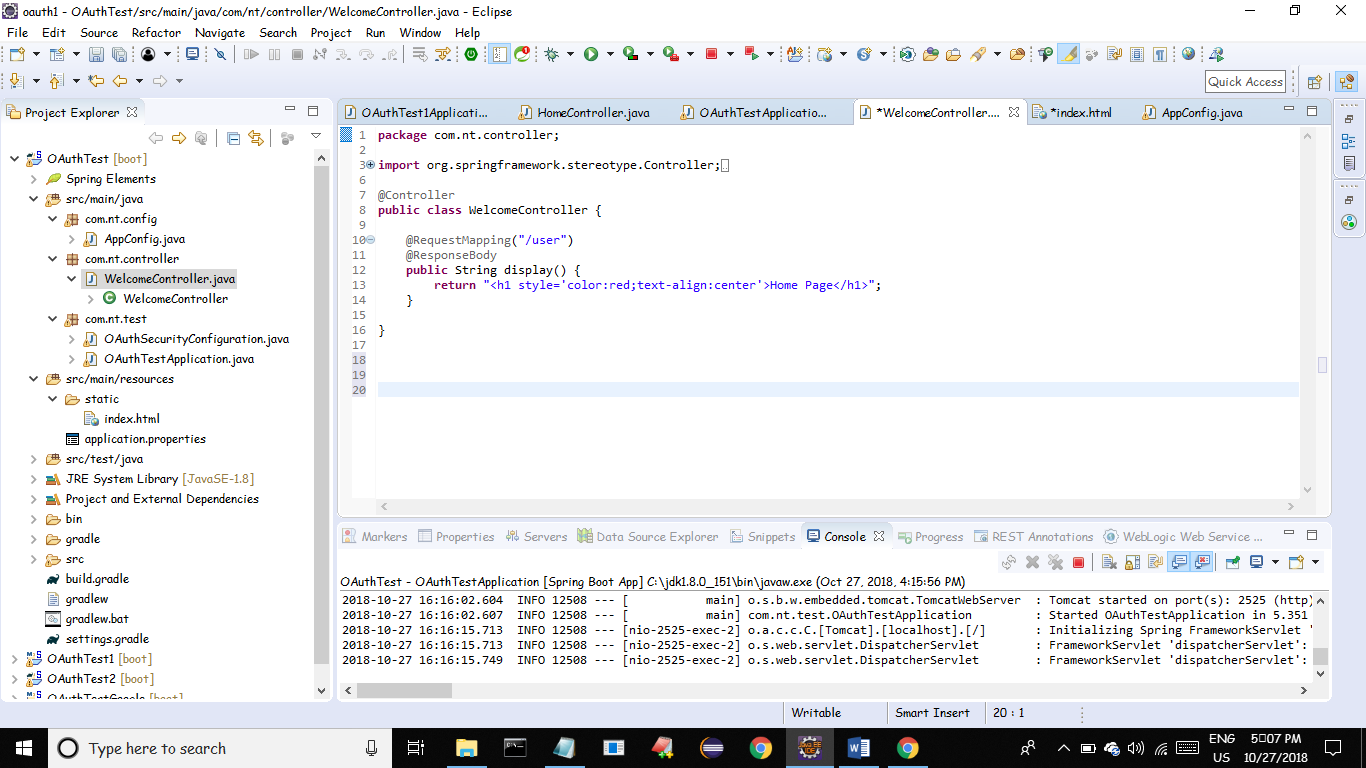
<h2 align=*"center"*>OAuth2 with Spring Boot</h2>

<a href=*"/user"*>Home page</a>

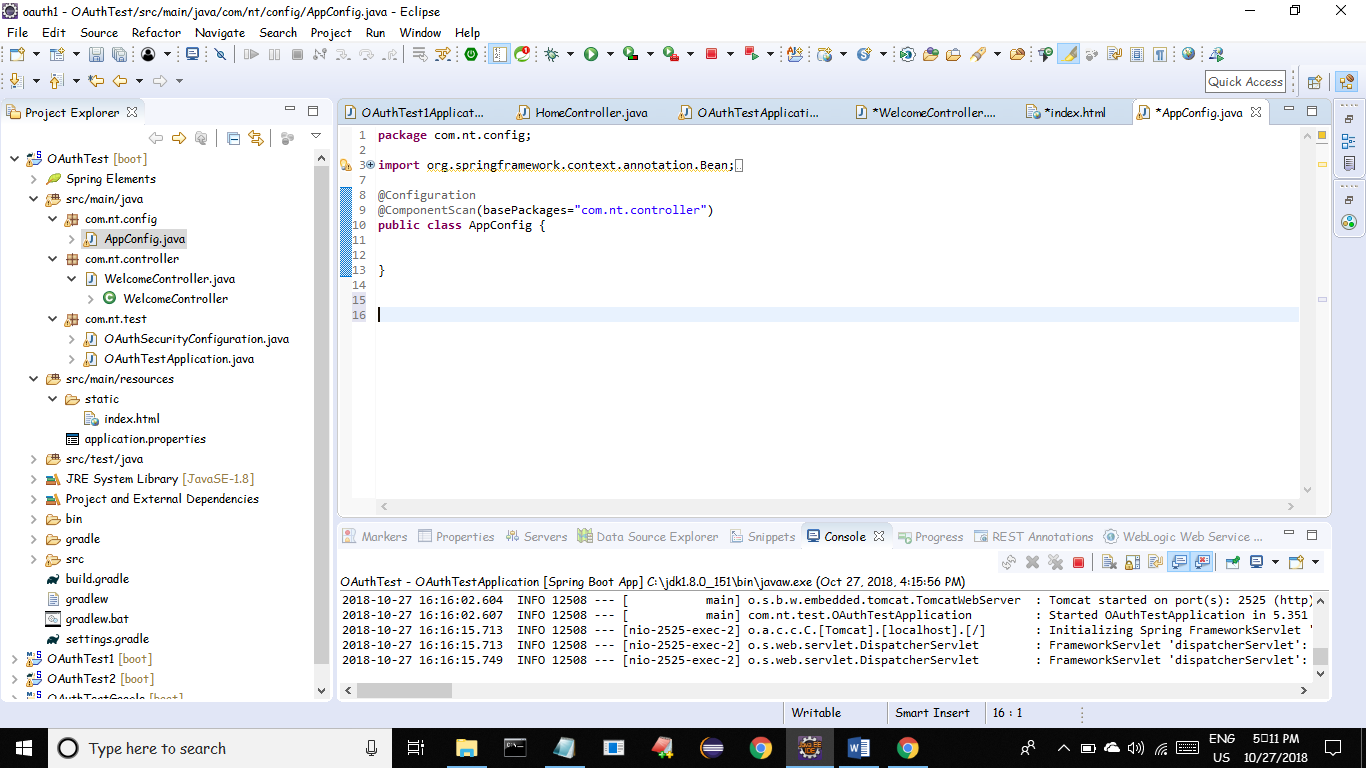
</body>

</html>

WelcomeController.java

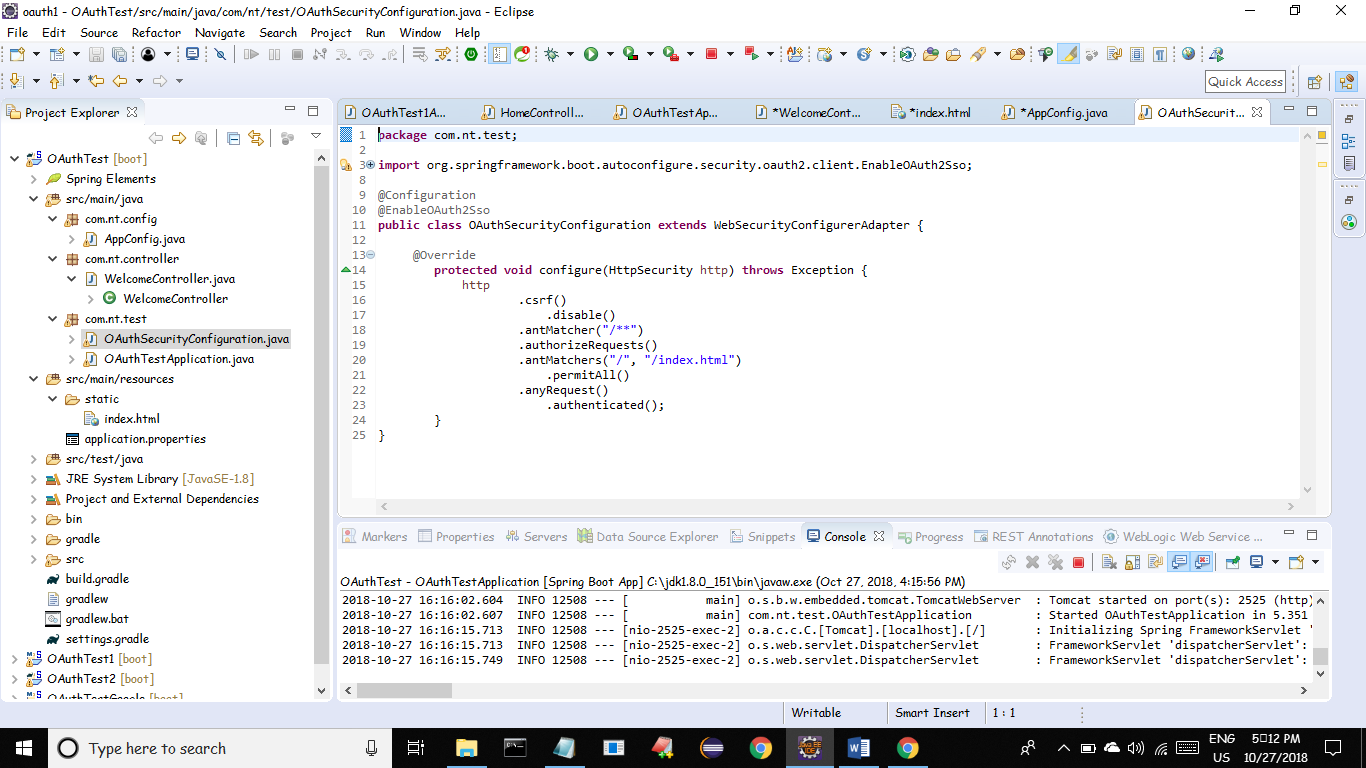


AppConfig.java

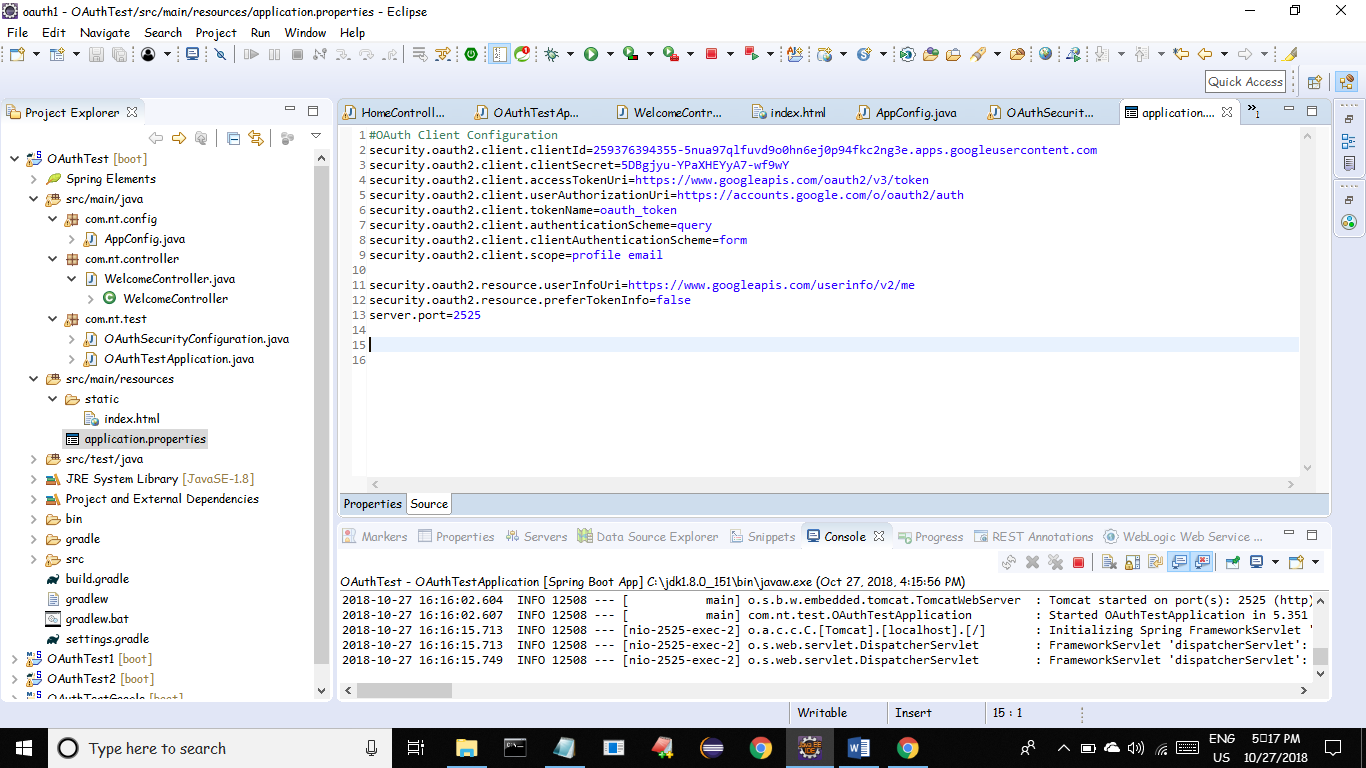


Enable OAuth2 by adding @EnableOAuth2Sso annotation on top of WebSecurityConfigurerAdapter.

OAuthSecurityConfiguration.java



Configure OAuth Clientid and secret of this application which is collected from Google console in **application.properties**



Run the Above Application. This application will allow all users to index.html

To access remaining resource it asks to login through google.

After Authentication we can get below details from google api about users through javax.security.Pricipal Object.

{

"authorities": [

{

"authority": "ROLE\_USER"

}

],

"details": {

"remoteAddress": "0:0:0:0:0:0:0:1",

"sessionId": "43Fxxxxxx",

"tokenValue": "ya29.xxxxxxxxx",

"tokenType": "Bearer",

"decodedDetails": null

},

"authenticated": true,

"userAuthentication": {

"authorities": [

{

"authority": "ROLE\_USER"

}

],

"details": {

"id": "106xxxxx",

"email": "xxxxxxxx@gmail.com",

"verified\_email": true,

"name": "xxxx yyyyyy",

"given\_name": "xxxxxx",

"family\_name": "yyyyy",

"link": "https://plus.google.com/xxxxxxxxxx",

"picture": "https://lh5.googleusercontent.com/xxxxxx/photo.jpg",

"locale": "en"

},

"authenticated": true,

"principal": "106xxxxx",

"credentials": "N/A",

"name": "106xxxxxxx"

},

"principal": "106xxxxxxxxxxx",

"clientOnly": false,

"credentials": "",

"oauth2Request": {

"clientId": "xxxxxxxxx.apps.googleusercontent.com",

"scope": [],

"requestParameters": {},

"resourceIds": [],

"authorities": [],

"approved": true,

"refresh": false,

"redirectUri": null,

"responseTypes": [],

"extensions": {},

"refreshTokenRequest": null,

"grantType": null

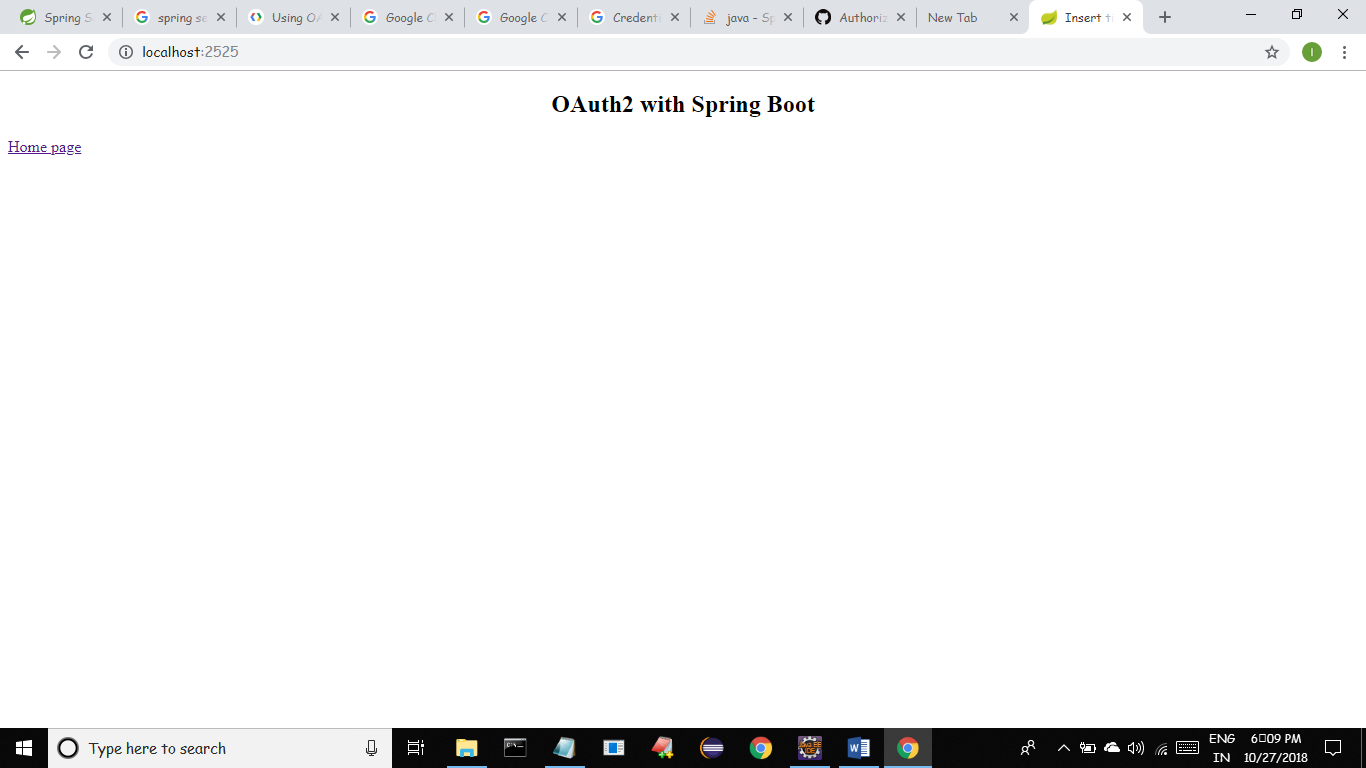
},

"name": "106xxxxxxxxxx"

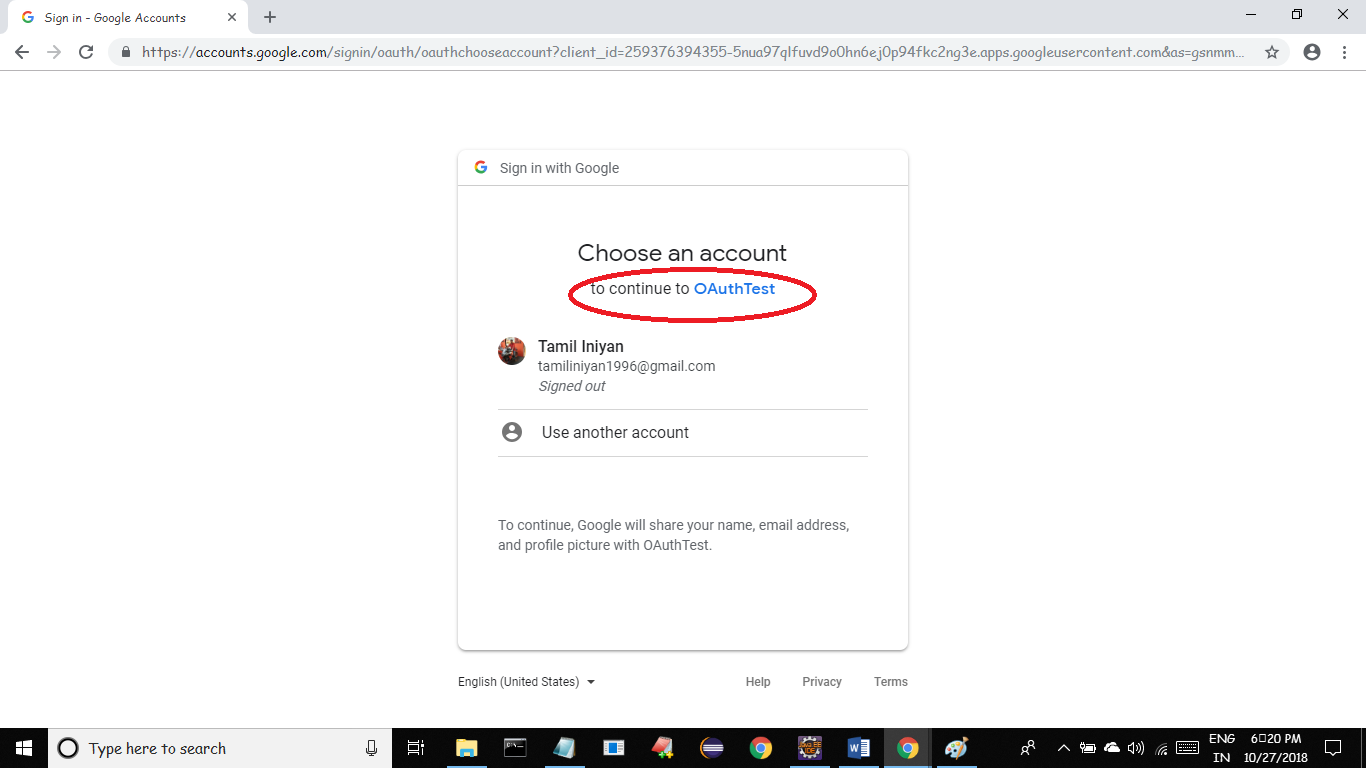
}

Spring Provides OAuth2Authentication Class to retrieve Specific details from the above.

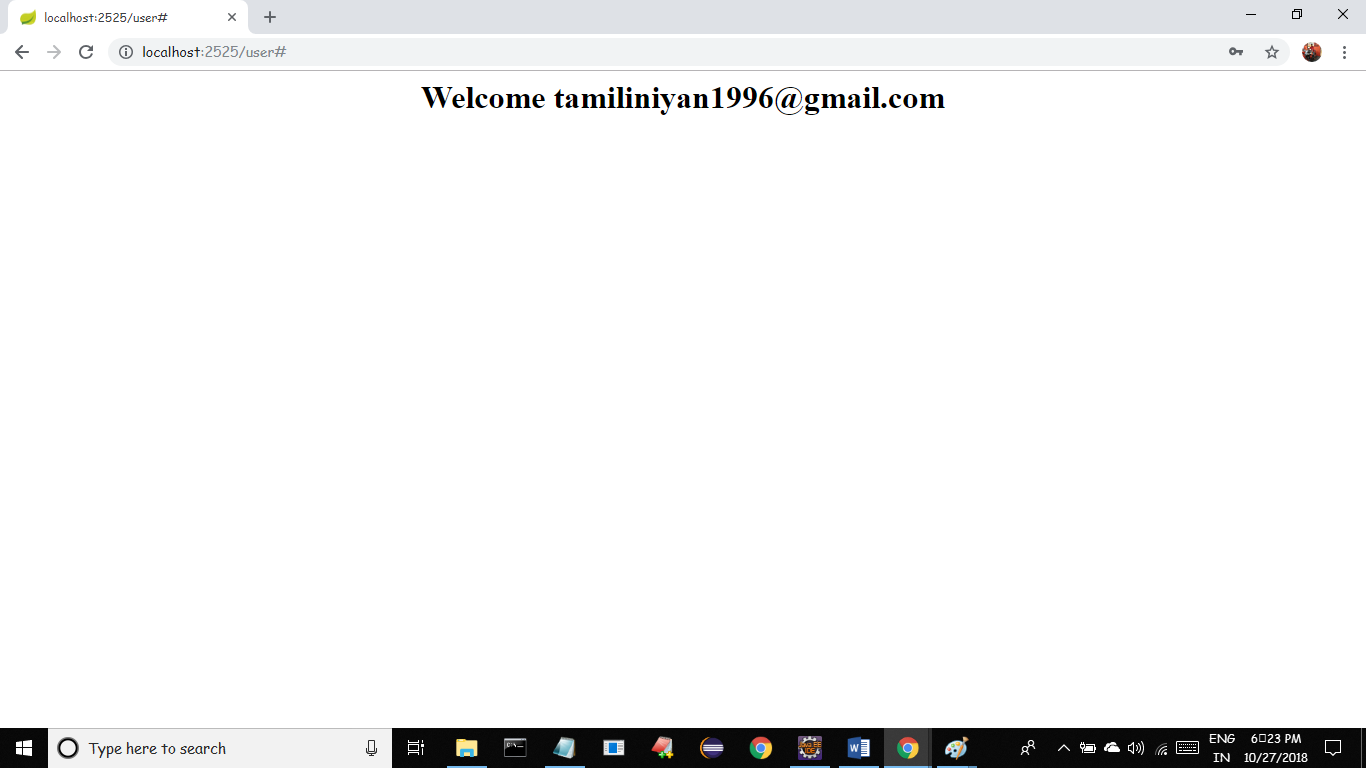
If you run the application you can see the below index page and to access anything from this site it will ask google authentication.



If you click home page hyperlink you can see it is redirecting to <https://accounts.google.com>



Once you authenticated successfully again it redirects to our target resource.

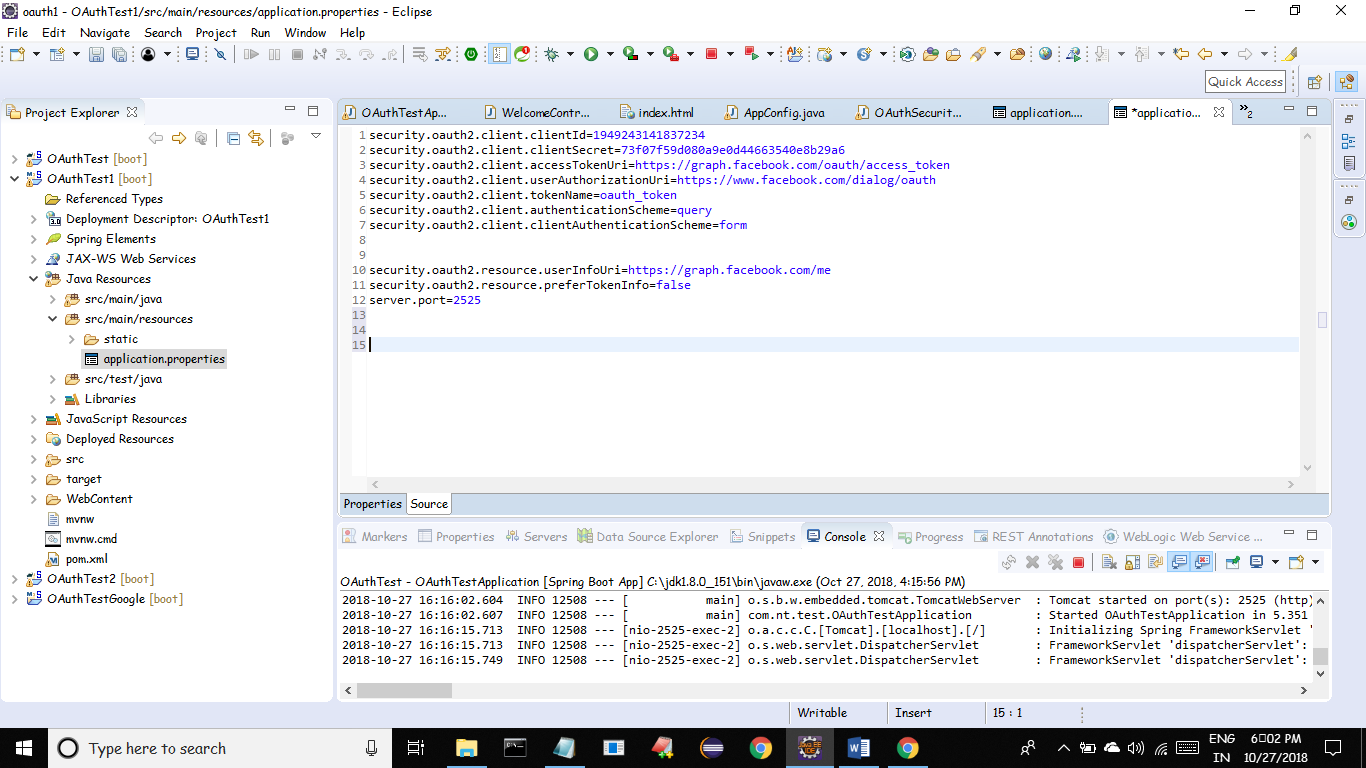


**Example 2 (Using Facebook as Authentication Provider)**

Same like the above example we need to collect OAuth clientId and secret from

*developers.facebook.com* by registering our application and redirect uri.

Add those clientid and secret in application.properties and run the application



Similarly in order to use GitHub as authentication provider visit

*developer.github.com*