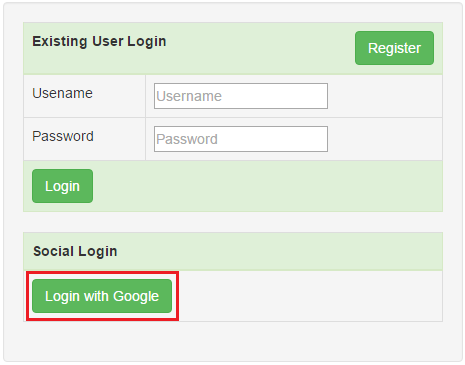
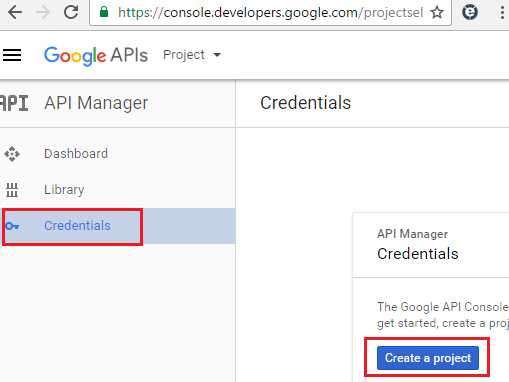
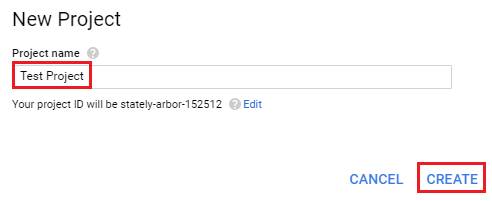
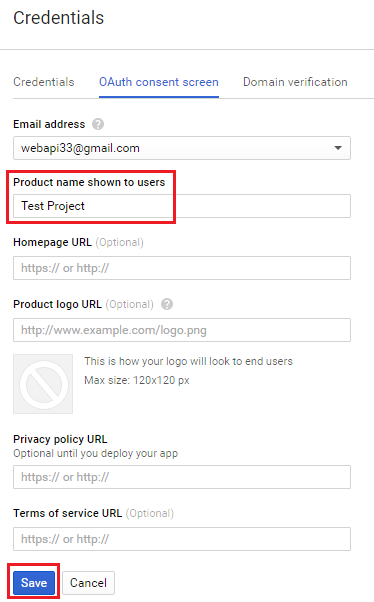
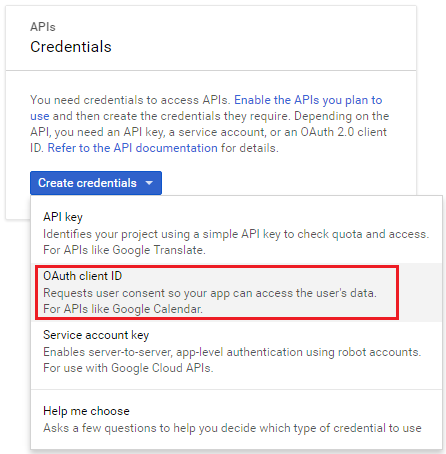
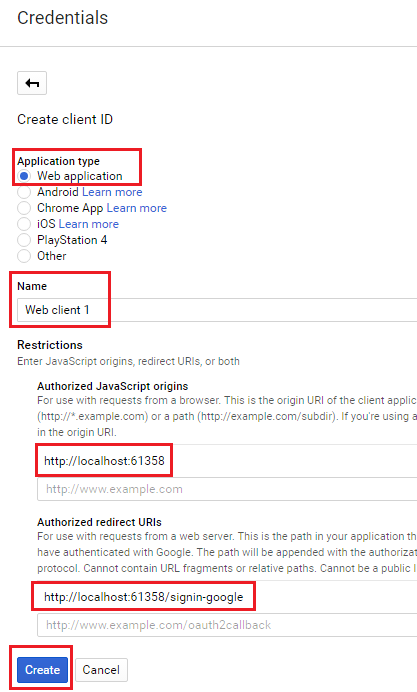
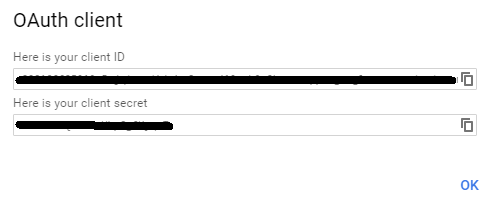
**In this video we will discuss**  
1. Benefits of social logins  
2. Using Google authentication with ASP.NET Web API   
  
  
  
**Benefits of social logins :**Registration is simple and easy. All they have to provide is their social login username and password and the user is registered with our application. This also means one less password to remember. When users don’t have to remember mulitple usernames and passwords to login to multiple web sites, there will be less failed logins. As you know remembering multiple usernames and passwords is definitely as hassle.  
  
From development point of view, we do not have to write code to manage usernames and passwords. All this is done by the external authentication providers like Google, Facebook, Twitter, Microsoft etc.   
  
  
  
**Using Google authentication with ASP.NET Web API :** When the user clicks "Login with Google" button, he will be redirected to Google login page. The user will then provide his Google credentials. Once the login is successful, the user will be redirected to our application with an access token, which is a proof that the user is successfully authenticated and our web application grants access to the protected resources.  
  
   
  
To use Google account for authentication, we will have to first register our application with Google. Here are the steps to register your application with Google. Once we successfully register our application with Google, we will be given a Client ID and Client Secret. We need both of these for using Google authentication with our Web API service.  
  
**Step 1 :** To register your application go to   
[https://console.developers.google.com](https://console.developers.google.com/)  
  
**Step 2 :** Login with your GMAIL account. Click on Credentials link on the left, and then create a new project, by clicking on "Create Project" button.   
  
   
  
Step 3 : Name your project "Test Project" and click "CREATE" button.   
  
   
  
**Step 4 :** The new project will be created. Click on "OAuth consent screen". In the "Product name shown to users" textbox type "Test Project" and click "Save" button   
  
   
  
**Step 5 :** The changes will be saved and you will be redirected to "Credentials" tab. If you are not redirected automatically, click on the "Credentials" tab and you will see "Create Credentials" dropdown button. Click on the button, and select "OAuth client ID"option   
  
   
  
**Step 6 :** On the next screen, 

* Select "Web application" radio button.
* Type "Web client 1" in the "Name" textbox.
* In the "Authorized JavaScript origins" textbox type in the URI of your application. I have my web api application running at http://localhost:61358
* In the "Authorized redirect URIs" textbox type in the redirect URI i.e the path in our application that users are redirected to after they have authenticated with Google. I have set it to http://localhost:61358/signin-google
* Click the "Create" button

   
  
You will see a popup with OAuth client ID and client secret. Make a note of both of them. We will need both of these later.   
  
   
  
**Step 7 :** Enable Google+ API service. To do this click on "Library" link on the left hand pane.Under "Social APIs" click on "Google+ API" link and click "Enable" button.  
  
**Enable Google OAuth authentication in ASP.NET Web API service**  
  
**Step 1 :**In Startup.Auth.cs file in App\_Start folder un-comment the following code block, and include ClientId and ClientSecret that we got after registering our application with Google.

app.UseGoogleAuthentication(new GoogleOAuth2AuthenticationOptions()

{

    ClientId = "Your Google Client Id",

    ClientSecret = "Your Google Client Secret"

});

**Step 2 :** In Login.html page include the following HTML table, just below "Existing User Login" table

<table class="table table-bordered">

    <thead>

        <tr class="success">

            <th>

                Social Logins

            </th>

        </tr>

    </thead>

    <tbody>

        <tr>

            <td>

                <input type="button" id="btnGoogleLogin"

                        value="Login with Google" class="btn btn-success" />

            </td>

        </tr>

    </tbody>

</table>

**Step 3 :**In the script section, in "Login.html" page, wire up the click event handler for "Login with Google" button. 

$('#btnGoogleLogin').click(function () {

    window.location.href = "/api/Account/ExternalLogin?provider=Google&response\_type=token&client\_id=self&redirect\_uri=http%3a%2f%2flocalhost%3a61358%2fLogin.html&state=GerGr5JlYx4t\_KpsK57GFSxVueteyBunu02xJTak5m01";

});

Notice when we click the button we are redirecting the user to/api/Account/ExternalLogin.   
  
The obvious question that we get at this point is from where do we get this URL. To get this URL, issue a GET request to api/Account/ExternalLogins?returnUrl=%2F&generateState=true. Since in my case the application is running at http://localhost:61358, the complete URL is http://localhost:61358/api/Account/ExternalLogins?returnUrl=%2F&generateState=true. The following is the response I got

<ArrayOfExternalLoginViewModel xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/EmployeeService.Models">

<ExternalLoginViewModel>

<Name>Google</Name>

<State>6Phc\_u0Xkj3opJ9TymPhw9olZV\_zB6Pjv\_OcIfNAprk1</State>

<Url>

/api/Account/ExternalLogin?provider=Google&response\_type=token&client\_id=self&redirect\_uri=http%3A%2F%2Flocalhost%3A61358%2F&state=6Phc\_u0Xkj3opJ9TymPhw9olZV\_zB6Pjv\_OcIfNAprk1

</Url>

</ExternalLoginViewModel>

</ArrayOfExternalLoginViewModel>

Notice the Url, it is encoded. Now go to <http://www.url-encode-decode.com/>. Paste the URL in "Enter the text that you wish to encode or decode" textbox and click on "Decode Url" button. Notice the redirect\_uri query string parameter. It is set to http://localhost:61358/ This parameter specifies the URI to redirect the user after they have been authenticated by Google. In our case we want the user to be redirected to the Login.html page. So URL encode the following URL  
http://localhost:61358/Login.html  
  
After you URL encode the above URL, it looks as shown below. Set it as the value for redirect\_uri query string parameter  
http%3A%2F%2Flocalhost%3A61358%2FLogin.html%2F  
  
**Step 4 :** Open "ApplicationOAuthProvider.cs" file from "Providers" folder, and modify ValidateClientRedirectUri() method as shown below. The change is to set the Redirect URI to Login.html

public override Task ValidateClientRedirectUri

    (OAuthValidateClientRedirectUriContext context)

{

    if (context.ClientId == \_publicClientId)

    {

        Uri expectedRootUri = new Uri(context.Request.Uri, "/Login.html");

        if (expectedRootUri.AbsoluteUri == context.RedirectUri)

        {

            context.Validated();

        }

    }

    return Task.FromResult<object>(null);

}

**Step 5 :** At this point build the solution and navigate to Login.html. Click on "Login with Google" button. Notice we are redirected to "Google" login page. Once we provide our Google credentials and successfully login, we are redirected to our application Login.html page with access token appended to the URL.  
  
http://localhost:61358/Login.html#access\_token=Pwf1kU\_LkrdueJbnaDtZohLsUHMDBvrYrdMxL59c4pilUC0&token\_type=bearer&expires\_in=1209600&state=GerGr5JlYx4t\_KpsK57GFSxVueteyBunu02xJTak5m01  
  
**Step 6 :** Next we need to retrieve the access token from the URL. The following JavaScript function does this. Add a new JavaScript file to the Scripts folder. Name it GoogleAuthentication.js. Reference jQuery. You can find minified jQuery file in the scripts folder. Copy and and paste the following function in it. Notice we named the function getAccessToken(). 

function getAccessToken() {

    if (location.hash) {

        if (location.hash.split('access\_token=')) {

            var accessToken = location.hash.split('access\_token=')[1].split('&')[0];

            if (accessToken) {

                isUserRegistered(accessToken);

            }

        }

    }

}

**Step 7 :** Notice the above function calls isUserRegistered() JavaScript function which checks if the user is already registered with our application. isUserRegistered() function is shown below. To check if the user is registered we issue a GET request to /api/Account/UserInfo passing it the access token using Authorization header. If the user is already registered with our application, we store the access token in local storage and redirect the user to our protected page which is Data.html. If the user is not registered, we call a different JavaScript function - signupExternalUser(). We will discuss what signupExternalUser() function does in just a bit. Now copy and paste the following function also in GoogleAuthentication.js file.

function isUserRegistered(accessToken) {

    $.ajax({

        url: '/api/Account/UserInfo',

        method: 'GET',

        headers: {

            'content-type': 'application/JSON',

            'Authorization' : 'Bearer ' + accessToken

        },

        success: function (response) {

            if (response.HasRegistered) {

                localStorage.setItem('accessToken', accessToken);

                localStorage.setItem('userName', response.Email);

                window.location.href = "Data.html";

            }

            else {

                signupExternalUser(accessToken);

            }

        }

    });

}

**Step 8 :** If the Google authenticated user is not already registered with our application, we need to register him. This is done by signupExternalUser() function show below. To register the user with our application we issue a POST request to /api/Account/RegisterExternal, passing it the access token. Once the user is successfully registered, we redirect him again to the same URL, to which the user is redirected when we clicked the "Login with Google" button. Since the user is already authenticated by Google, the access token will be appended to the URL, which will be parsed by getAccessToken() JavaScript function. getAccessToken() function will again call isUserRegistered() function. Since the user is already registered with our application, we redirect him to the Data.html page and he will be able to see the employees data. Copy and paste the following function also in GoogleAuthentication.jsfile.

function signupExternalUser(accessToken) {

    $.ajax({

        url: '/api/Account/RegisterExternal',

        method: 'POST',

        headers: {

            'content-type': 'application/json',

            'Authorization': 'Bearer ' + accessToken

        },

        success: function () {

            window.location.href = "/api/Account/ExternalLogin?provider=Google&response\_type=token&client\_id=self&redirect\_uri=http%3a%2f%2flocalhost%3a61358%2fLogin.html&state=GerGr5JlYx4t\_KpsK57GFSxVueteyBunu02xJTak5m01";

        }

    });

}

**Step 9 :** In AccountController.cs, modify RegisterExternal() method as shown below. Notice we removed "RegisterExternalBindingModel" parameter and if (!ModelState.IsValid) code block.

// POST api/Account/RegisterExternal

[OverrideAuthentication]

[HostAuthentication(DefaultAuthenticationTypes.ExternalBearer)]

[Route("RegisterExternal")]

public async Task<IHttpActionResult> RegisterExternal()

{

    var info = await Authentication.GetExternalLoginInfoAsync();

    if (info == null)

    {

        return InternalServerError();

    }

    var user = new ApplicationUser() { UserName = info.Email, Email = info.Email };

    IdentityResult result = await UserManager.CreateAsync(user);

    if (!result.Succeeded)

    {

        return GetErrorResult(result);

    }

    result = await UserManager.AddLoginAsync(user.Id, info.Login);

    if (!result.Succeeded)

    {

        return GetErrorResult(result);

    }

    return Ok();

}

**Step 10 :** Finally, on Login.html page reference GoogleAuthentication.js file and call get getAccessToken() function  
  
Build the solution and navigate to Login.html page and click on "Login with Google"button. Notice we are redirected to Google Login page. Once we provide our Google credentials and Login, we are redirected to Data.html page. When we click "Load Employees" button we see employees data.  
  
At this point if you query AspNetUsers and AspNetUserLogins tables you will see an entry for your login is made into these 2 tables

* Select \* from AspNetUsers
* Select \* from AspNetUserLogins