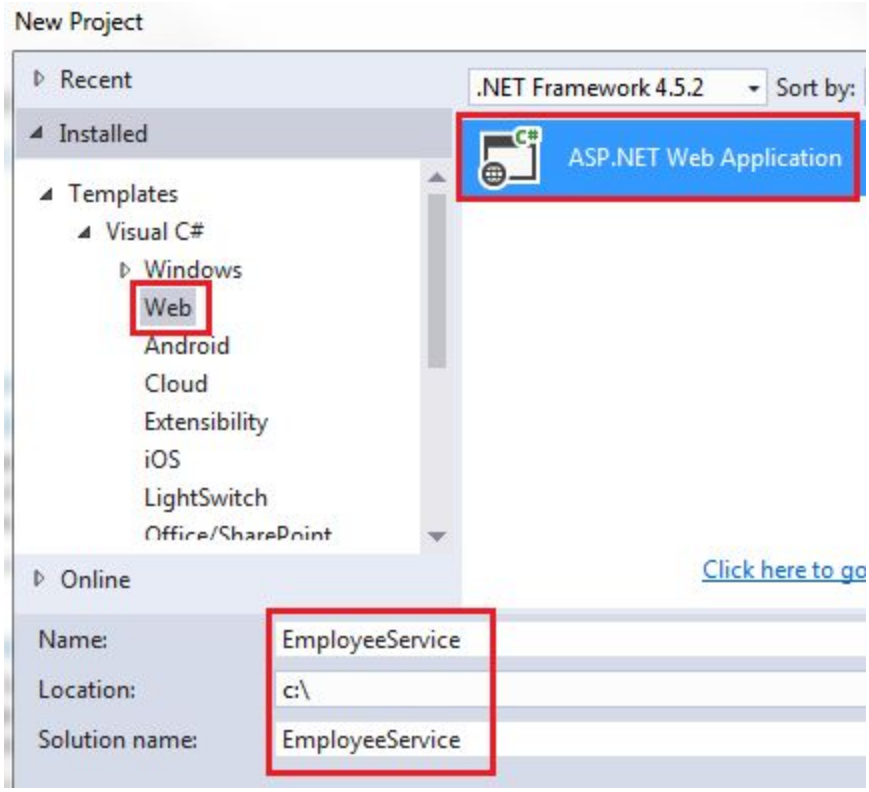
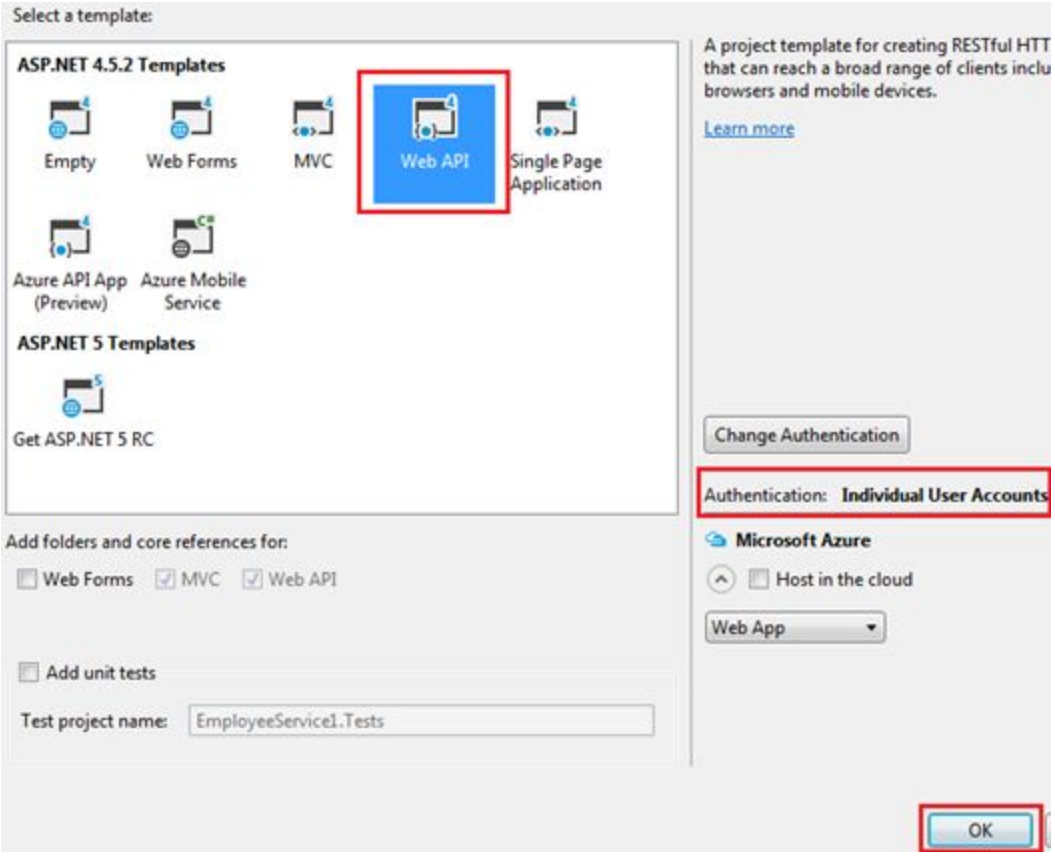
1. Create A Web API Project

Step 1 : Run Visual Studio and Select File - New Project

Step 2 : In the "New Project" dialog box, select "Web" under "Installed" templates. From the middle pane, select "ASP.NET Web Application". Name the project "EmployeeService" and click "OK"



Step 3 : On the next screen select **"Web API"** and set Authentication to **"Individual User Accounts"** and click **"OK"**. The Individual User Accounts option uses a membership database in which the users that we register will be stored. We will discuss the Membership Database in a later video in this series.



Step 4 : Execute the following script in SQL Server Management Studio to create the Employees table and populate it with test data

We will be using the following Employees table for this demo.

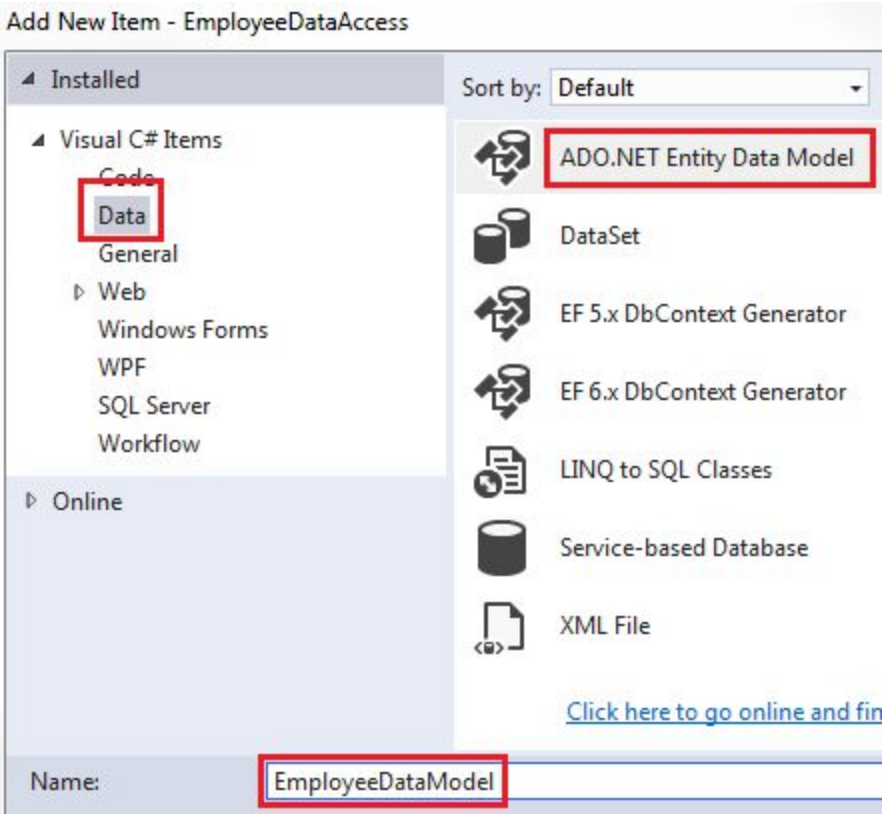
ID	FirstName	LastName	Gender	Salary
1	Mark	Hastings	Male	50000
2	Steve	Pound	Male	45000
3	Ben	Hoskins	Male	70000
4	Philip	Hastings	Male	45000
5	Mary	Lambeth	Female	30000
6	Valarie	Vikings	Female	35000
7	John	Stanmore	Male	80000

```sql
Create Database EmployeeDB
Go

Use EmployeeDB
Go


Create table Employees
(
     ID int primary key identity,
     FirstName nvarchar(50),
     LastName nvarchar(50),
     Gender nvarchar(50),
     Salary int
)
Go

Insert into Employees values ('Mark', 'Hastings', 'Male', 60000)
Insert into Employees values ('Steve', 'Pound', 'Male', 45000)
Insert into Employees values ('Ben', 'Hoskins', 'Male', 70000)
Insert into Employees values ('Philip', 'Hastings', 'Male', 45000)
Insert into Employees values ('Mary', 'Lambeth', 'Female', 30000)
Insert into Employees values ('Valarie', 'Vikings', 'Female', 35000)
Insert into Employees values ('John', 'Stanmore', 'Male', 80000)
Go
```

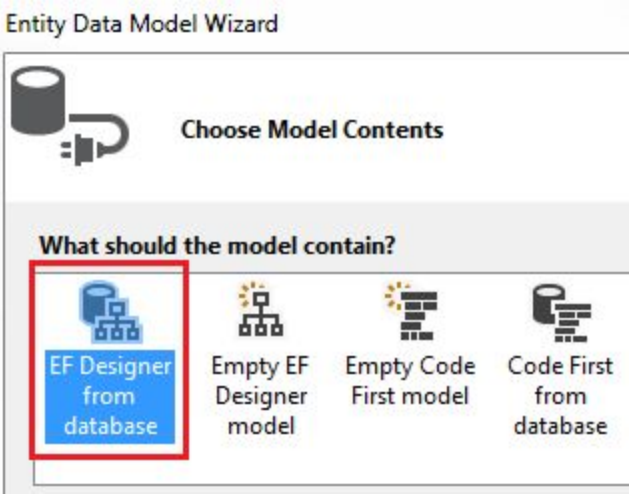
Step 5 : Right click on "EmployeeService" project and select Add - New Item

Step 6 : In the "Add New Item" window,

- Select "Data" from the left pane
- Select ADO.NET Entity Data Model from the middle pane
- In the Name text box, type EmployeeDataModel and click Add



Step 7 : On the Entity Data Model Wizard, select **"EF Designer from database"** option and click next

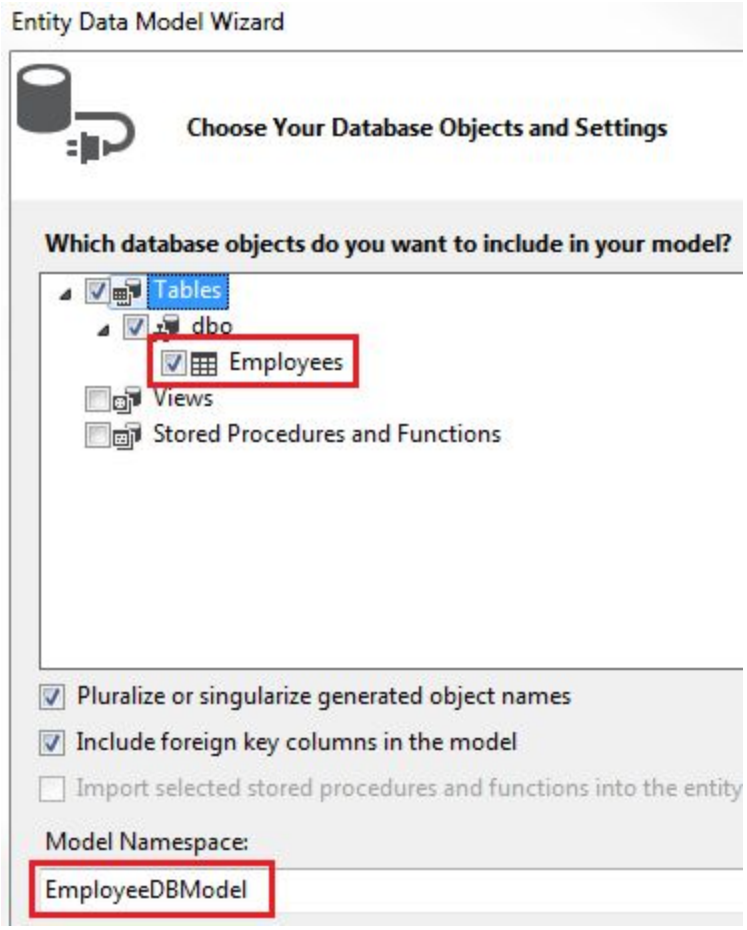


Step 8 : On the next screen, click "New Connection" button

Step 9 : On "Connection Properties" window, set
Server Name = (local)
Authentication = Windows Authentication

Select or enter a database name = EmployeeDB
Click OK and then click Next

Step 10 : On the nex screen, select "Employees" table and click Finish.



Step 11 : Right click on the Controllers folder in EmployeeService project and select Add - Controller

Step 12 : Select "Web API 2 Controller - Empty" and click "Add"

Step 13 : On the next screen set the Controller Name = EmployeesController and click Add

Step 14 : Copy and paste the following code in EmployeesController.cs

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

namespace EmployeeService.Controllers
{
    public class EmployeesController : ApiController
       {
        public IEnumerable<Employee> Get()
       {
            using(EmployeeDBEntities entities = new EmployeeDBEntities())
              {
               return entities.Employees.ToList();
              }
       }
       }
}
```

At this point when you navigate to /api/employees you should see all employees.

Documentation:
SHow the Snapshot of POSTMAN TEST RESULT

Step 15 : Decorate the EmployeesController with [Authorize] attribute.

```
[Authorize]
public class EmployeesController : ApiController
```

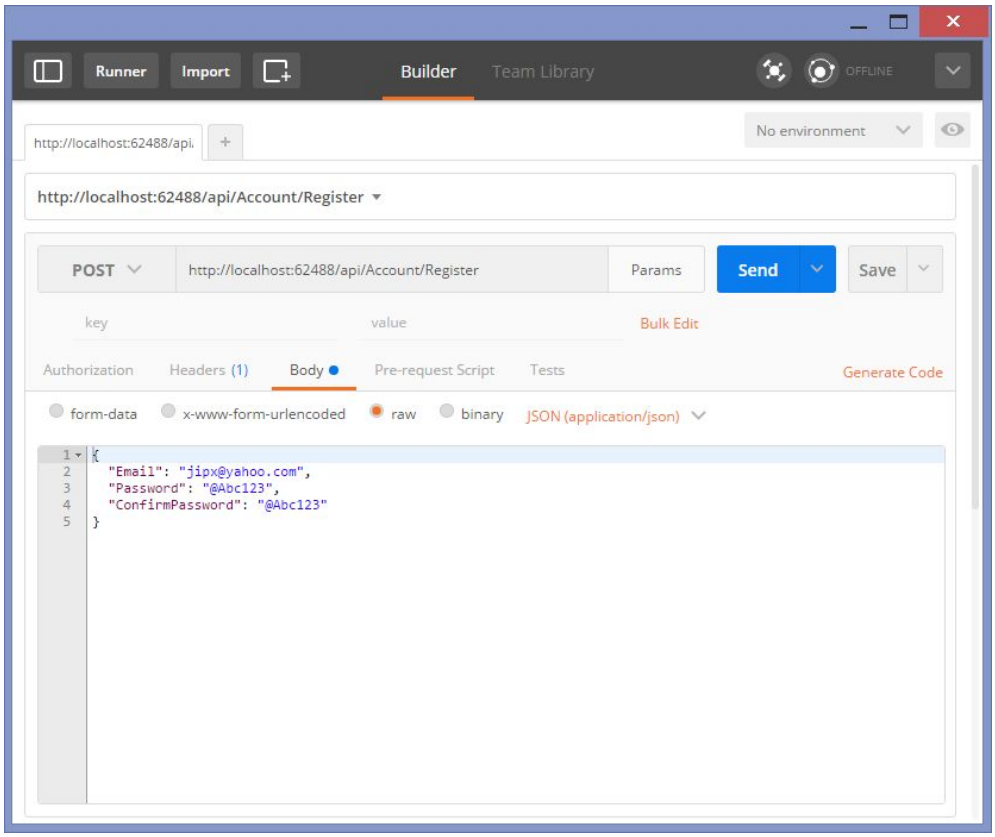
Step 16 : Build the solution and when you navigate to /api/employees we get "Authorization has been denied for this request"

Documentation:
SHow the Snapshot of POSTMAN TEST RESULT

Since the EmployeesController is decorated with [Authorize] attribute, all the actions in the controller must be authenticated. Otherwise, we get 401 error - Authorization has been denied for this request.

2. Using POSTMAN to test ASP.NET Web API token based authentication

User registration:



You need to use your studentID as part of your email: P*******@sp.edu.sg

Documentation:
SHow the Snapshot of POSTMAN TEST RESULT

Now let's use POSTMAN and generate the access token using the above username and password.

- Issue a POST request to /token
- In the request body include username and the password.
- We also need to set grant_type=password. This indicates that we are presenting password for acquiring access token.

Documentation:
SHow the Snapshot of POSTMAN TEST RESULT

Now let's understand how the access token is generated.

The code that generates the access token is provided by ASP.NET Web API out of the box. To see this code open the file "Startup.Auth.cs" that is present in App_Start folder. Notice in the ConfigureAuth() method

- An instance of OAuthAuthorizationServerOptions is created
- The /Token end point to which we have posted username and password is specified in here
- The token expiry is specified using AccessTokenExpireTimeSpan property. In this case the token expires 14 days after it is issued. You can change this to meet your application needs.
- The Provider property is initialised with a new instance of ApplicationOAuthProvider class. This class has GrantResourceOwnerCredentials() method which verifies if the provided username and password are valid. If valid an access token is issued. The token is generated when context.Validated(ticket) method is called.

Now let us see how to call EmployeesController and retrieve employees data.

If we issue a GET request to http://localhost:61358/api/employees we get 401 Unauthorized error. Since the EmployeesController is decorated with [Authorize] attribute, the request needs to be authenticated. So with every request we have to send the Bearer token using Authorization header.

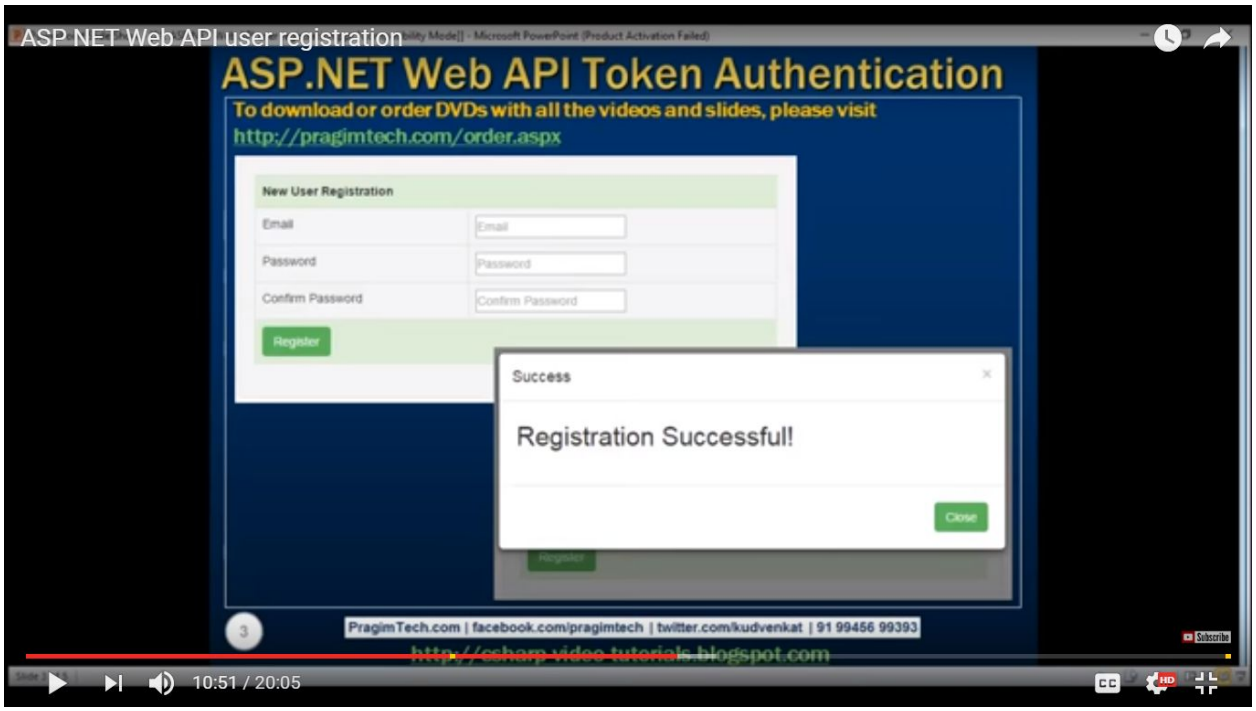
Documentation:
SHow the Snapshot of POSTMAN TEST RESULT

3. ASP.NET Web API user registration page: Register.html

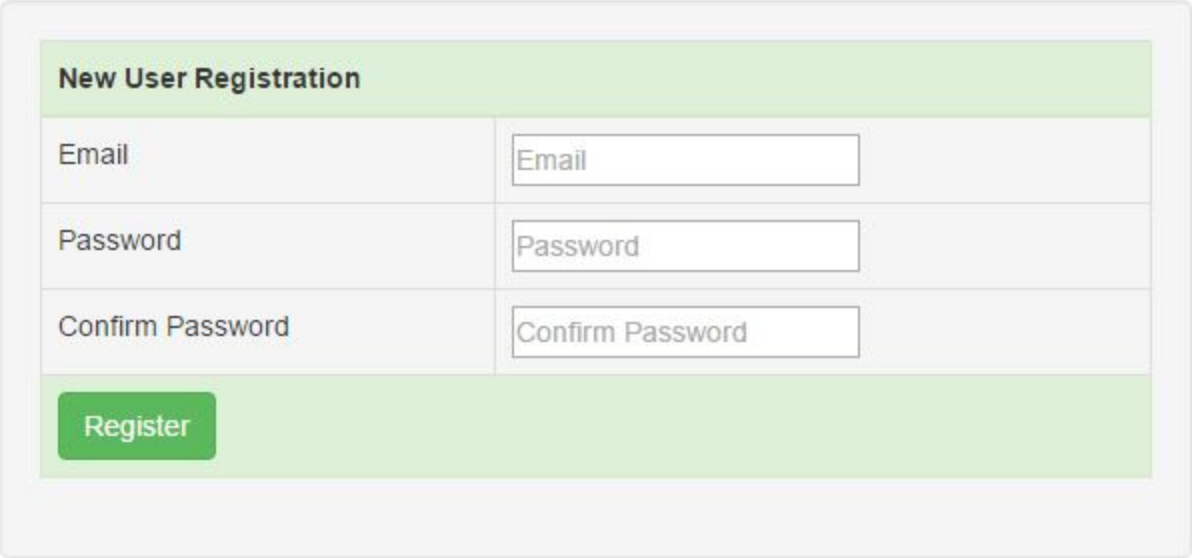
Watch the video:

ASP NET Web API user registration

https://www.youtube.com/watch?v=6Rrnt3AKs2g



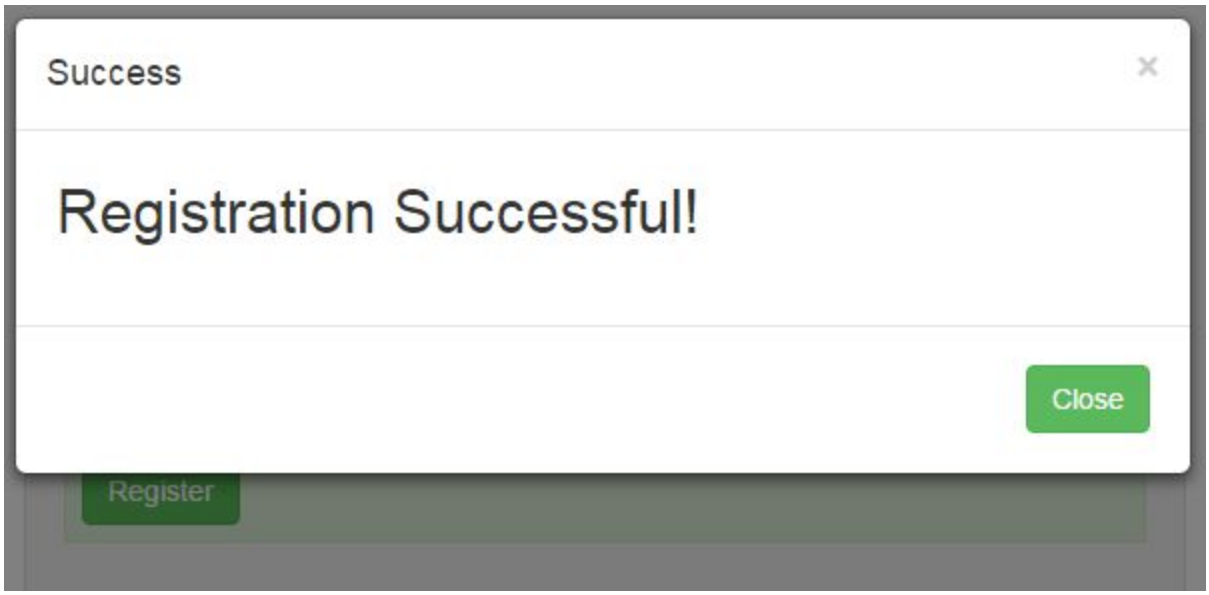
The registration page should be as shown below.



For a new user to register they have to provide

1. Email address
2. Password
3. Confirm password

When all the fields are provided and when the Register button is clicked, the new user details should be saved and a modal dialog should be displayed as shown below.



To achieve this, add an HTML page to EmployeeService project. Name it **Register.html**. Copy and paste the following HTML and jQuery code.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body style="padding-top:20px">
    <div class="col-md-10 col-md-offset-1">
        <div class="well">
            <!--This table contains the fields that we want to capture to register a new
user-->
            <table class="table table-bordered">
                <thead>
                    <tr class="success">
                        <th colspan="2">
                             New User Registration
                        </th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Email</td>
                        <td><input type="text" id="txtEmail" placeholder="Email" /> </td>
                    </tr>
                    <tr>
                        <td>Password</td>
                        <td><input type="password" id="txtPassword"
                                   placeholder="Password" /></td>
                    </tr>
                    <tr>
                        <td>Confirm Password</td>
                        <td><input type="password" id="txtConfirmPassword"
                                   placeholder="Confirm Password" /></td>
                    </tr>
                    <tr class="success">
                        <td colspan="2">
                            <input id="btnRegister" class="btn btn-success"
                                   type="button" value="Register" />
                        </td>
                    </tr>
                </tbody>
            </table>
            <!--Bootstrap modal dialog that shows up when regsitration is successful-->
            <div class="modal fade" tabindex="-1" id="successModal"
                 data-keyboard="false" data-backdrop="static">
```

```
            <div class="modal-dialog modal-sm">
                <div class="modal-content">
                      <div class="modal-header">
                        <button type="button" class="close" data-dismiss="modal">
                            &times;
                        </button>
                        <h4 class="modal-title">Success</h4>
                    </div>
                    <div class="modal-body">
                        <form>
                            <h2 class="modal-title">Registration Successful!</h2>
                        </form>
                    </div>
                    <div class="modal-footer">
                        <button type="button" class="btn btn-success"
                                data-dismiss="modal">
                         Close
                           </button>
                    </div>
                </div>
            </div>
        </div>
        <!--Bootstrap alert to display any validation errors-->
        <div id="divError" class="alert alert-danger collapse">
            <a id="linkClose" href="#" class="close">&times;</a>
            <div id="divErrorText"></div>
        </div>
    </div>
</div>

<script src="Scripts/jquery-1.10.2.min.js"></script>
<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">
   $(document).ready(function () {

        //Close the bootstrap alert
           $('#linkClose').click(function () {
           $('#divError').hide('fade');
           });

        // Save the new user details
           $('#btnRegister').click(function () {
           $.ajax({
                   url: '/api/account/register',
                   method: 'POST',
                   data: {
                   email: $('#txtEmail').val(),
                   password: $('#txtPassword').val(),
```

```
                        confirmPassword: $('#txtConfirmPassword').val()
                        },
                        success: function () {
                            $('#successModal').modal('show');
                        },
                        error: function (jqXHR) {
                        $('#divErrorText').text(jqXHR.responseText);
                        $('#divError').show('fade');
                        }
                });
                });
        });
    </script>
</body>
</html>
```

Please note :

1. The ajax() method posts the data to '/api/account/register'
2. You will find the Register() method in AccountController in Controllers folder
3. AccountController is provided by ASP.NET Web API, which saves data to a local membership database

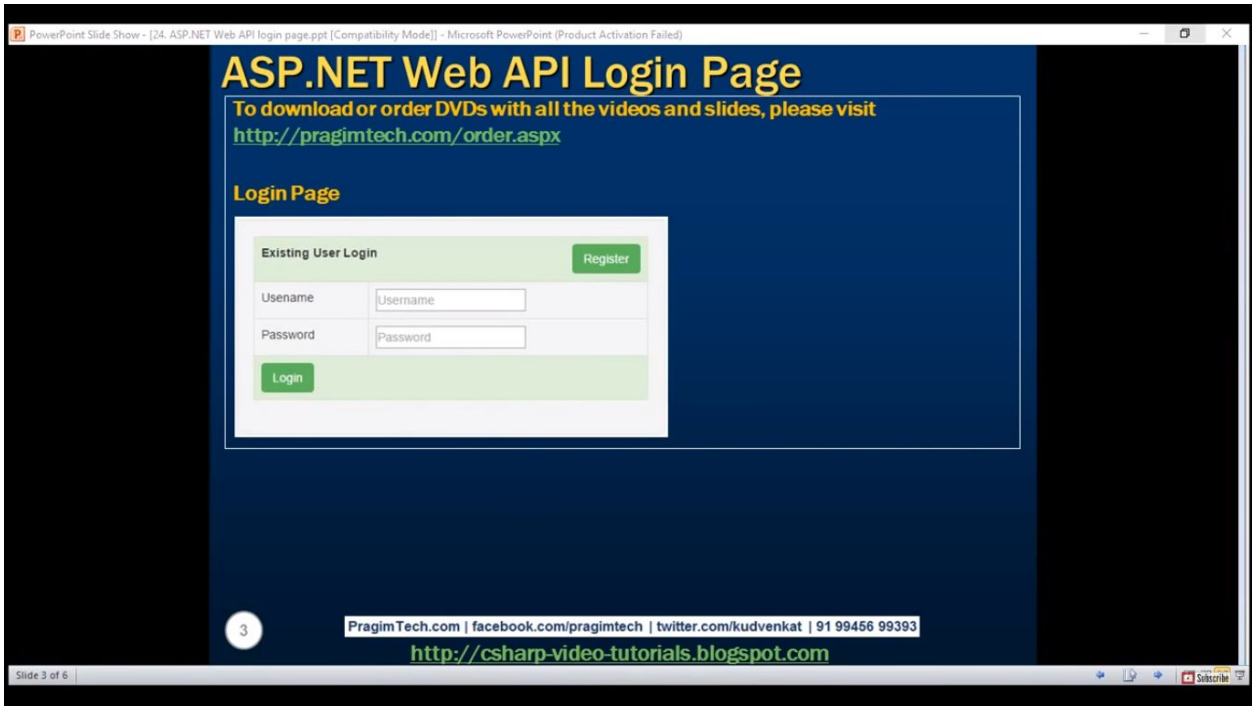
Questions:

Take a screenshot of the data stored in your database.

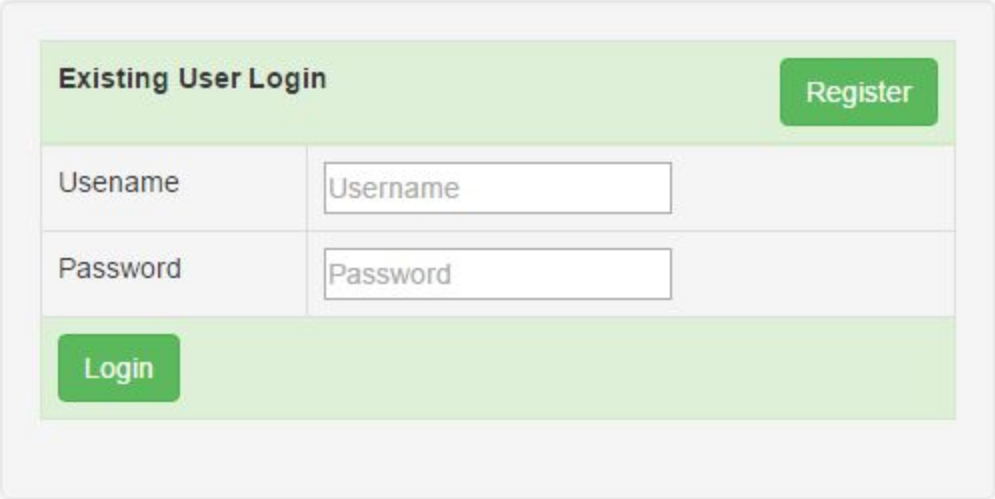
4. Implementing login page Login.html for ASP.NET Web API.

Resource:
http://csharp-video-tutorials.blogspot.sg/2016/12/aspnet-web-api-login-page.html

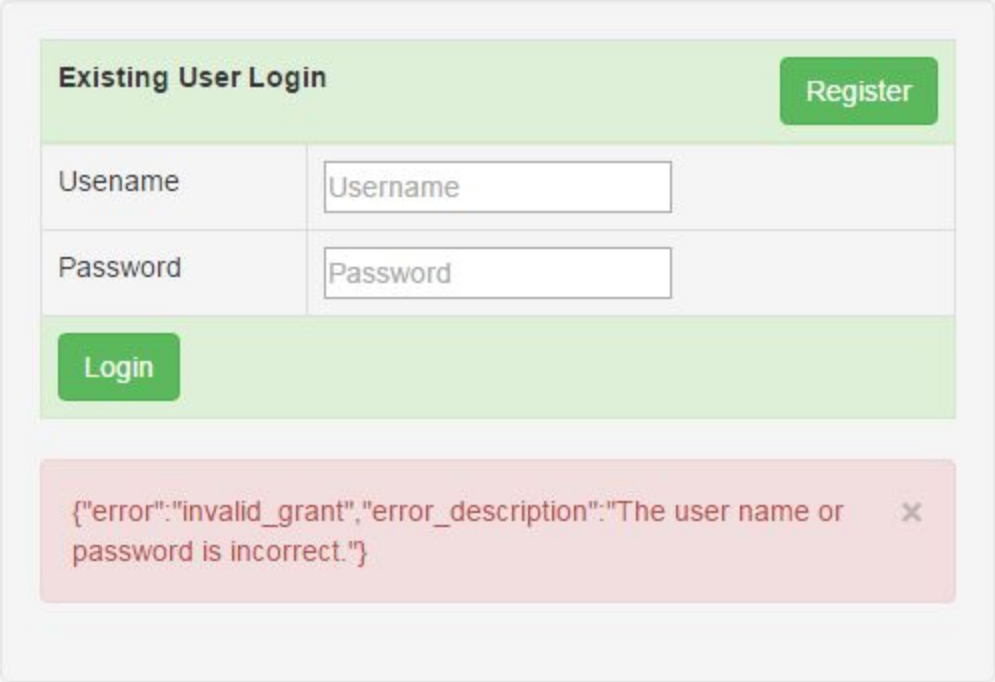
Watch YouTube:



We want to design a login page that looks as shown below



If we provide invalid username and password the error should be displayed as shown below



Add a new HTML page to the EmployeeService project. Name it Login.html. Copy and paste the following HTML & jQuery code.

```html
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body style="padding-top:20px">
    <div class="col-md-10 col-md-offset-1">
        <div class="well">
            <!--Table to capture username and password-->
            <table class="table table-bordered">
                <thead>
                    <tr class="success">
                        <th colspan="2">
                             Existing User Login
                            <a href="Register.html" class="btn btn-success pull-right">
                             Register
                            </a>
                        </th>
                 </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Usename</td>
                        <td>
                            <input type="text" id="txtUsername" placeholder="Username" />
                        </td>
                    </tr>
                    <tr>
                        <td>Password</td>
                        <td>
                            <input type="password" id="txtPassword"
                                   placeholder="Password" />
                        </td>
                    </tr>
                    <tr class="success">
                        <td colspan="2">
                            <input id="btnLogin" class="btn btn-success" type="button"
                                   value="Login" />
                        </td>
                    </tr>
                </tbody>
            </table>
            <!--Bootstrap alert to display error message if the login fails-->
            <div id="divError" class="alert alert-danger collapse">
                <a id="linkClose" href="#" class="close">&times;</a>
                <div id="divErrorText"></div>
            </div>
```

```
        </div>
    </div>

    <script src="Scripts/jquery-1.10.2.min.js"></script>

    <script type="text/javascript">
       $(document).ready(function () {

              $('#linkClose').click(function () {
              $('#divError').hide('fade');
              });

              $('#btnLogin').click(function () {
              $.ajax({
                    // Post username, password & the grant type to /token
                      url: '/token',
                      method: 'POST',
                      contentType: 'application/json',
                      data: {
                      username: $('#txtUsername').val(),
                      password: $('#txtPassword').val(),
                      grant_type: 'password'
                      },
                    // When the request completes successfully, save the
                    // access token in the browser session storage and
                    // redirect the user to Data.html page. We do not have
                    // this page yet. So please add it to the
                    // EmployeeService project before running it
                      success: function (response) {
                      sessionStorage.setItem("accessToken", response.access_token);
                      window.location.href = "Data.html";
                      },
                     // Display errors if any in the Bootstrap alert <div>
                      error: function (jqXHR) {
                      $('#divErrorText').text(jqXHR.responseText);
                      $('#divError').show('fade');
                      }
              });
              });
       });
    </script>
</body>
</html>
```

Please note :

1. sessionStorage data is lost when the browser window is closed.

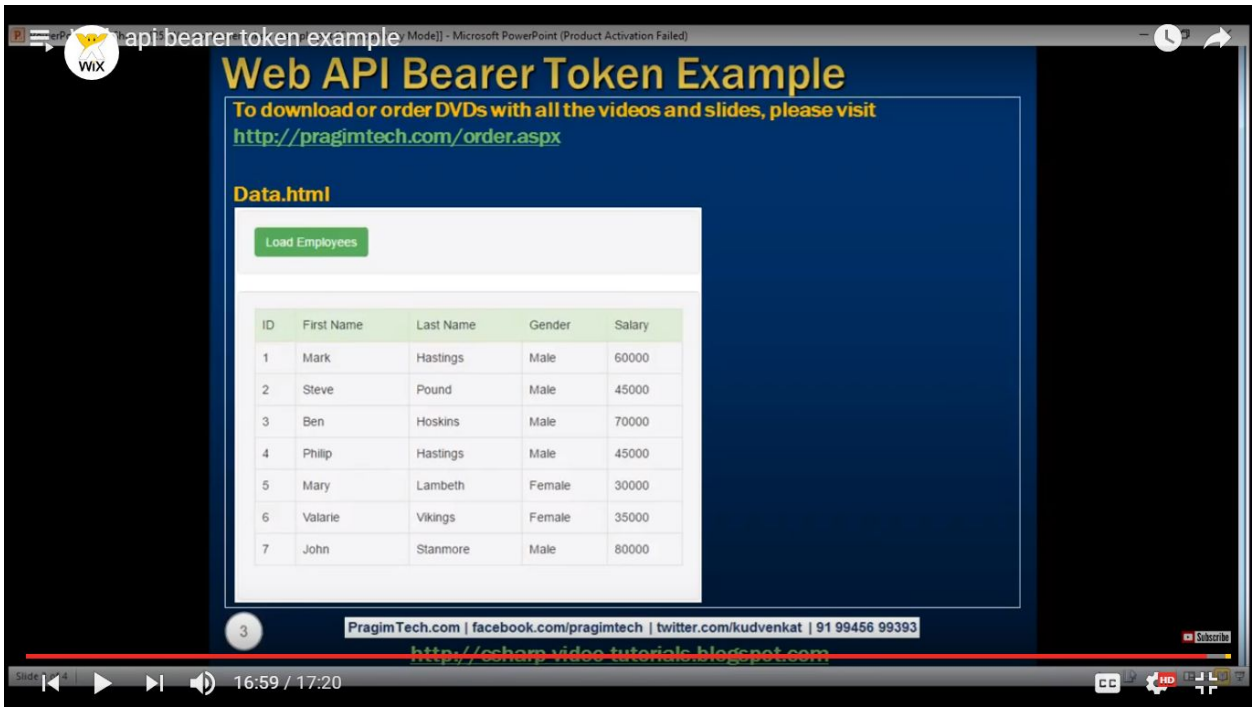
2. To store an item in the browser session storage use setItem() method
   Example : sessionStorage.setItem("accessToken", response.access_token)
3. To retrieve an item from the browser session storage use getItem() method
   Example : sessionStorage.getItem("accessToken")
4. To remove an item from the browser session storage use removeItem() method
   Example : sessionStorage.removeItem('accessToken')

On the Register.html page, we do not have Login button, which takes us to the Login page if the user is already registered. So please include Login button just below "New User Registration" text in the <th> element on Register.html page as shown below.

```
<thead>
    <tr class="success">
        <th colspan="2">
              New User Registration
            <a href="Login.html" class="btn btn-success pull-right">Login</a>
        </th>
    </tr>
</thead>
```

5. implementing the *Data.html* page which retrieves data by calling the EmployeesController using the bearer token.

Watch the Video



how to use bearer token for authentication and retrieving data from the server. This is continuation to Part 24. Please watch Part 24 from ASP.NET Web API tutorial before proceeding.

We want to implement a page that retrieves employee data from the server. If the user is not authenticated, he should be automatically redirected to the login page. The user should be able to get to the data page, only if he is logged in.

Load Employees

ID	First Name	Last Name	Gender	Salary
1	Mark	Hastings	Male	60000
2	Steve	Pound	Male	45000
3	Ben	Hoskins	Male	70000
4	Philip	Hastings	Male	45000
5	Mary	Lambeth	Female	30000
6	Valarie	Vikings	Female	35000
7	John	Stanmore	Male	80000

Add a new HTML page to the EmployeeService project. Name it Data.html page. Copy and paste the following HTML and jQuery code.

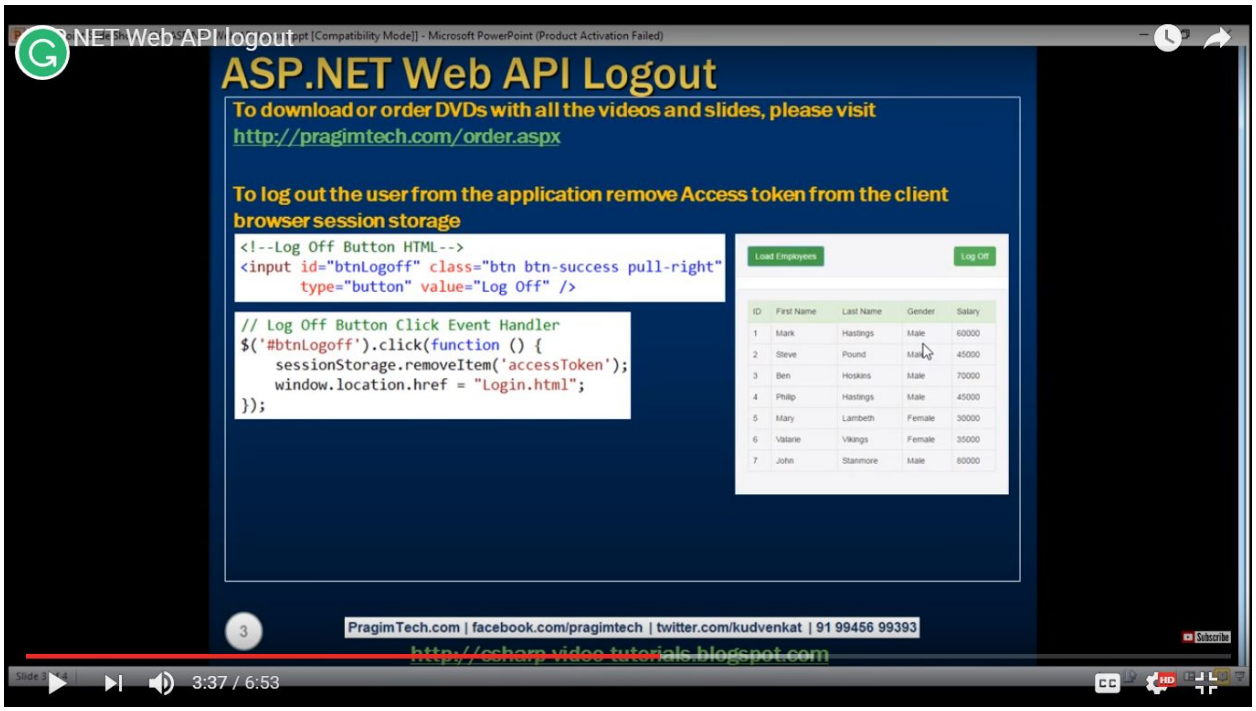
```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body style="padding-top:20px">
    <div class="col-md-10 col-md-offset-1">
        <div class="well">
            <input id="btnLoadEmployees" class="btn btn-success"
                   type="button" value="Load Employees" />
        </div>
        <div id="divData" class="well hidden">
            <table class="table table-bordered" id="tblData">
                <thead>
```

```
                <tr class="success">
                    <td>ID</td>
                    <td>First Name</td>
                    <td>Last Name</td>
                    <td>Gender</td>
                    <td>Salary</td>
                </tr>
            </thead>
            <tbody id="tblBody"></tbody>
        </table>
    </div>
    <div class="modal fade" tabindex="-1" id="errorModal"
         data-keyboard="false" data-backdrop="static">
        <div class="modal-dialog modal-sm">
            <div class="modal-content">
                <div class="modal-header">
                    <button type="button" class="close" data-dismiss="modal">
                        &times;
                    </button>
                    <h4 class="modal-title">Session Expired</h4>
                </div>
                <div class="modal-body">
                    <form>
                        <h2 class="modal-title">Close this message to login again</h2>
                    </form>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-danger"
                            data-dismiss="modal">
                         Close
                    </button>
                </div>
            </div>
        </div>
    </div>
    <div id="divError" class="alert alert-danger collapse">
        <a id="linkClose" href="#" class="close">&times;</a>
        <div id="divErrorText"></div>
    </div>
</div>

<script src="Scripts/jquery-1.10.2.min.js"></script>
<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">
   $(document).ready(function () {
        if (sessionStorage.getItem('accessToken') == null) {
           window.location.href = "Login.html";
           }
```

```
            $('#linkClose').click(function () {
            $('#divError').hide('fade');
            });

            $('#errorModal').on('hidden.bs.modal', function () {
            window.location.href = "Login.html";
            });

            $('#btnLoadEmployees').click(function () {
            $.ajax({
                    url: '/api/employees',
                    method: 'GET',
                    headers: {
                      'Authorization': 'Bearer '
                           + sessionStorage.getItem("accessToken")
                    },
                    success: function (data) {
                    $('#divData').removeClass('hidden');
                    $('#tblBody').empty();
                    $.each(data, function (index, value) {
                          var row = $('<tr><td>' + value.ID + '</td><td>'
                           + value.FirstName + '</td><td>'
                           + value.LastName + '</td><td>'
                           + value.Gender + '</td><td>'
                           + value.Salary + '</td></tr>');
                           $('#tblData').append(row);
                    });
                    },
                    error: function (jQXHR) {
                      // If status code is 401, access token expired, so
                      // redirect the user to the login page
                      if (jQXHR.status == "401") {
                           $('#errorModal').modal('show');
                    }
                  else {
                           $('#divErrorText').text(jqXHR.responseText);
                           $('#divError').show('fade');
                    }
                    }
            });
            });
        });
    </script>
</body>
</html>
```

6. Implemeting LOG OFF Function

At the moment, the only way to log off the user is by closing the browser window. As we are storing the bearer token in browser session storage, when we close the browser we loose it from the session. In our next video we will discuss, how to explicitly log out the user without closing the browser window.



o log out the user from the application all we have to do is remove the Access token from the client browser session storage. Here is what we want to do.

1. Include a Log Off button on the Data.html page

2. When the Log Off button is clicked remove the access token from client browser session storage and redirect the user to the login page.

Load Employees Log Off

ID	First Name	Last Name	Gender	Salary
1	Mark	Hastings	Male	60000
2	Steve	Pound	Male	45000
3	Ben	Hoskins	Male	70000
4	Philip	Hastings	Male	45000
5	Mary	Lambeth	Female	30000
6	Valarie	Vikings	Female	35000
7	John	Stanmore	Male	80000

HTML for the Log Off button. Include the following HTML on the Data.html page immediately below the Load Employees button.

```
<input id="btnLogoff" class="btn btn-success pull-right"
    type="button" value="Log Off" />
```

In the script section include the following jQuery click event handler for the Log Off button as shown below.

```
$('#btnLogoff').click(function () {
        sessionStorage.removeItem('accessToken');
        window.location.href = "Login.html";
});
```

There are 2 ways for the user to Log Off

1. By closing the browser window. Since we are storing the access token in browser session storage, the access token will be lost when we close the browser window.

2. By clicking the "Log Off" button, which explicitly removes the access token from the browser session storage.

If you do not want to loose the access token, when the browser is closed store the access token in browser local storage instead of session storage. The way you store, retrieve and remove items from local storage is exactly the same as storing, retrieving and removing items from session storage, except that you use localStorage object instead of sessionStorage object.

At this point, you may have the following questions.

We are only deleting the access token on the client. We are not invalidating or deleting the access token from the server side. If someone can intercept the access token, will they not be able to use that access token and gain access to the system.
The straight answer to the question is YES. If someone is able to intercept the access token, they will be able to impersonate and gain access to the system. However, most of the systems that use access tokens, work over SSL (Secure Socket Layer), which inhibits intercepting access tokens.

Should we invalidate or delete access tokens from the server
No, there is no need to invalidate or delete access tokens from the server. Access token lives on the client, and it is enough if we remove it from the client. Another good practise is to set the expiry of the access token to as short time as practically possible depending on the nature of your application.

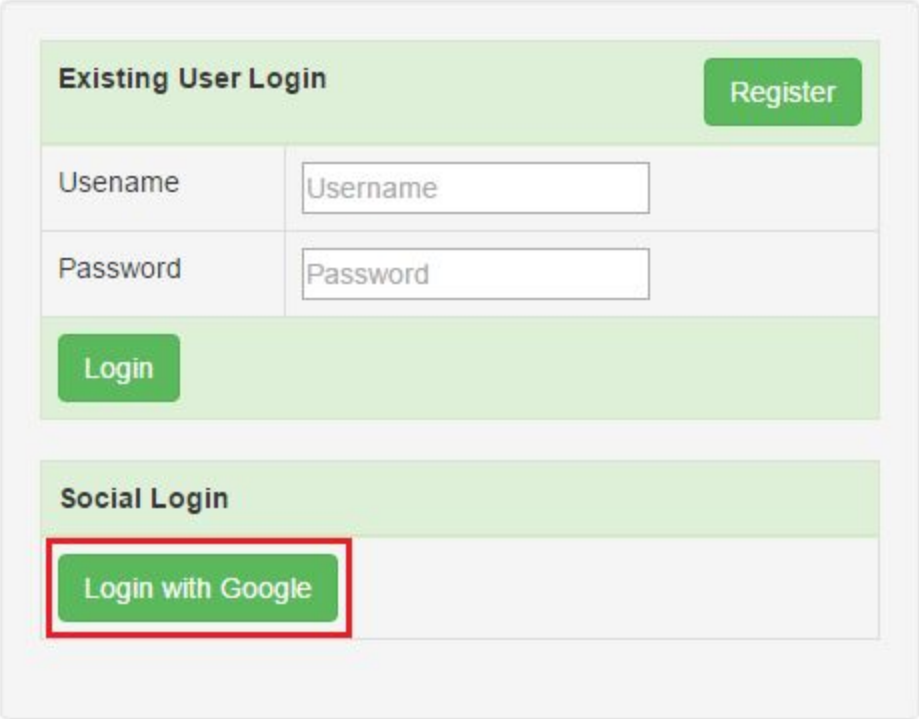
7. ASP.NET Web API google authentication

https://www.youtube.com/watch?v=39LTFT4rTxk&t=14s

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.

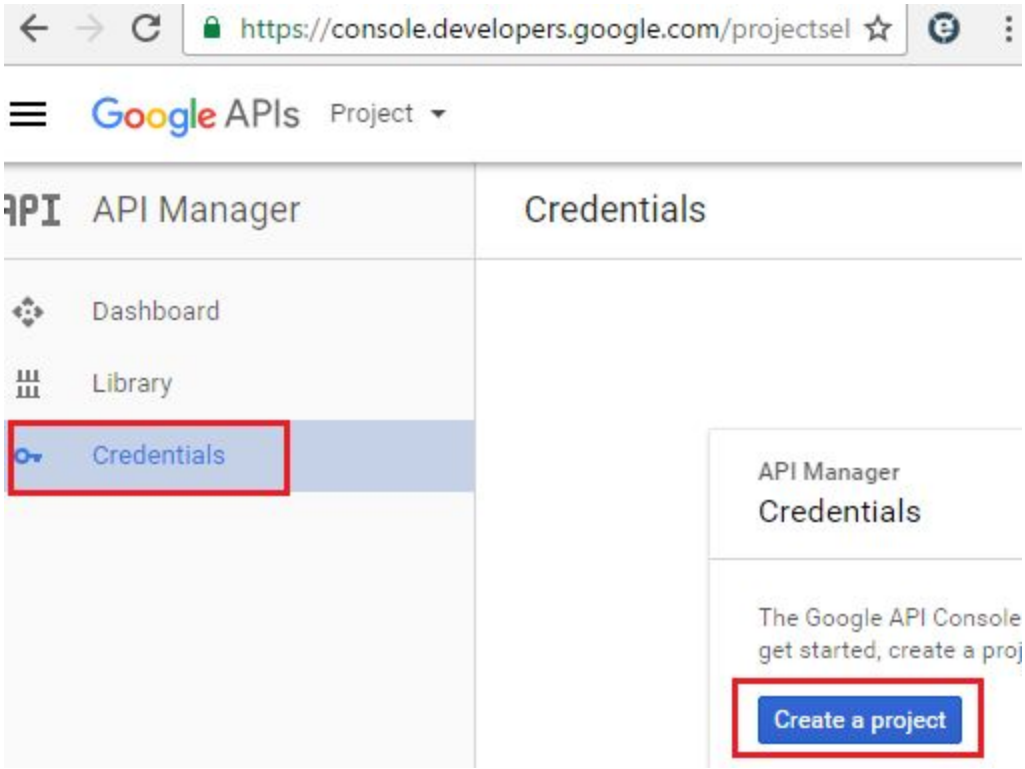
Using Google authentication with ASP.NET Web API : When the user clicks "Login with Google" button, he will be redirected to Google login page. The user will then provide his Google credentials. Once the login is successful, the user will be redirected to our application with an access token, which is 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.

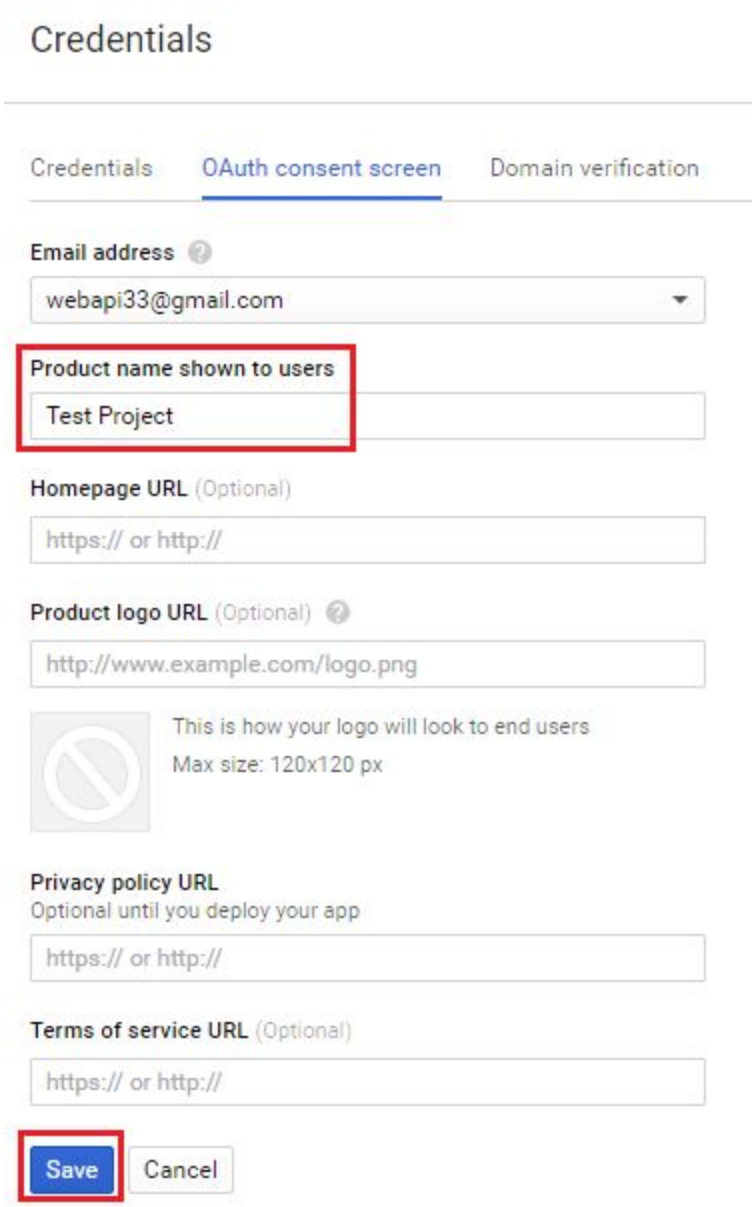
Step 1 : To register your application go to https://console.developers.google.com

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.

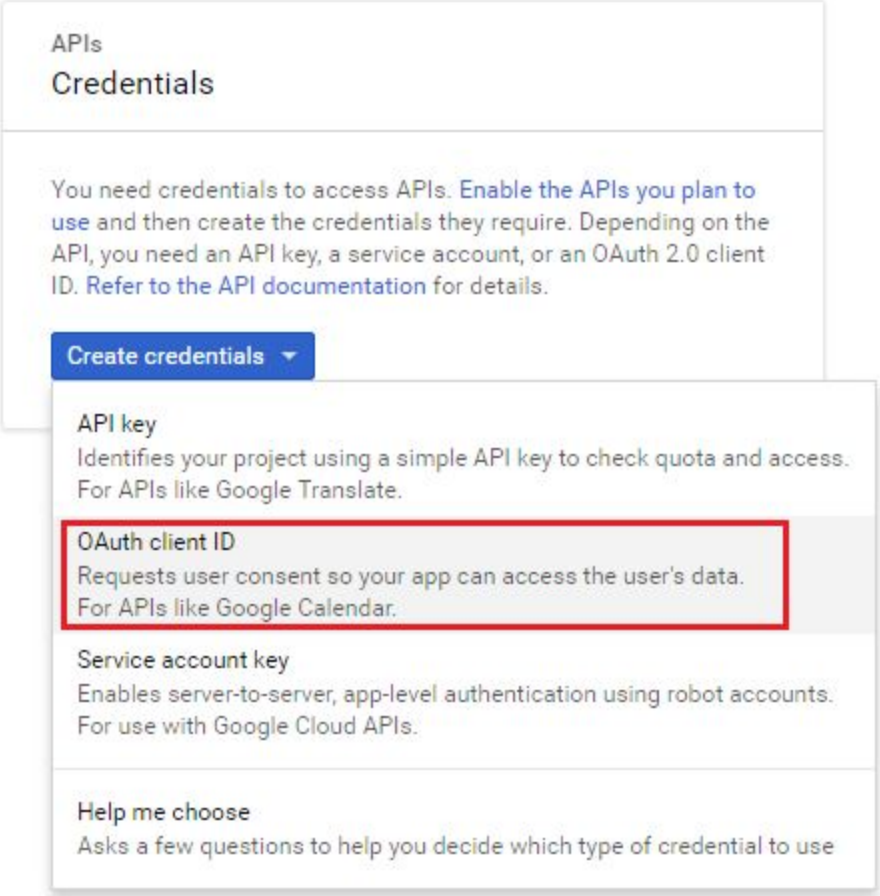


Step 3 : Name your project "Test Project" and click "CREATE" button.

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

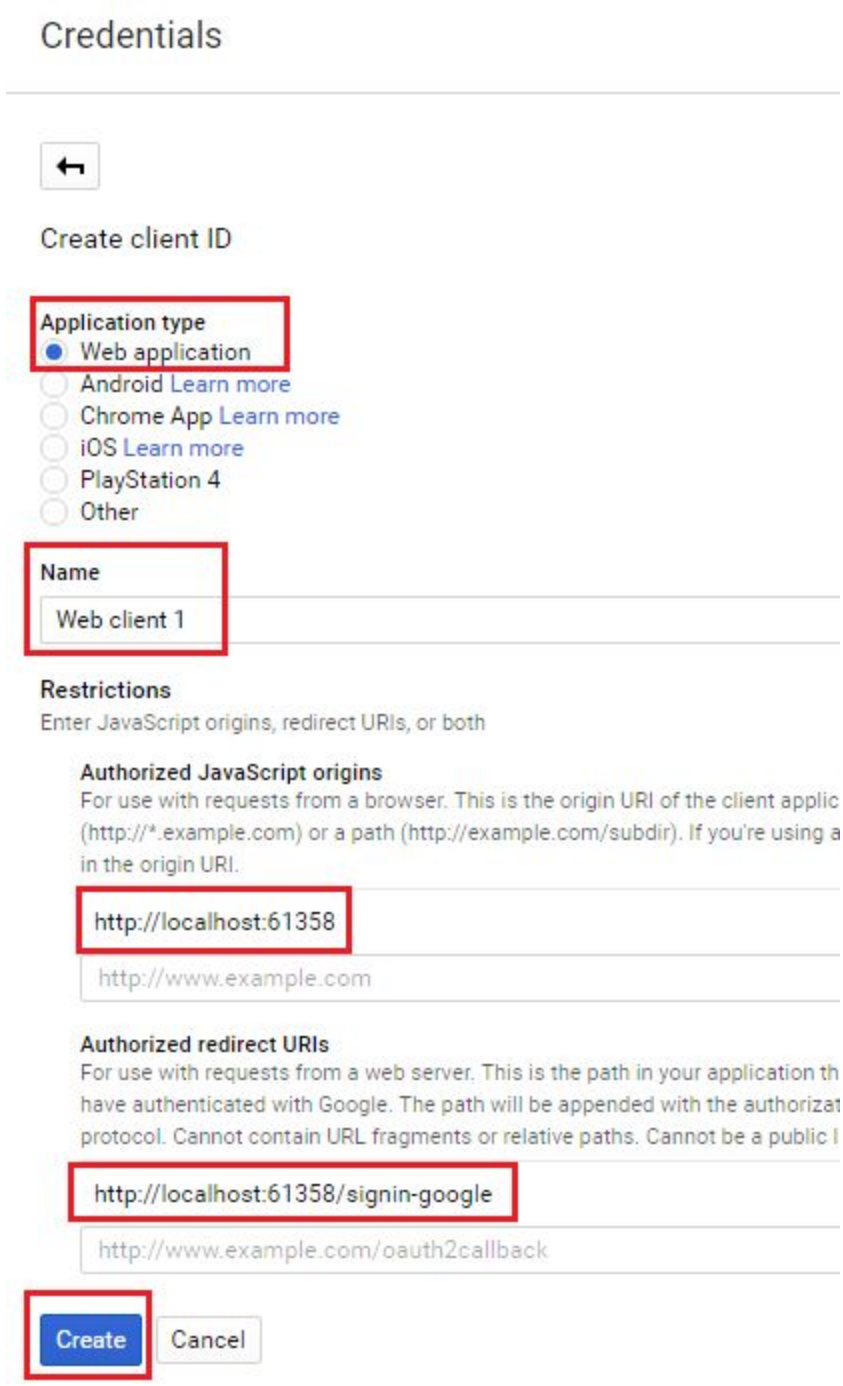


Step 5 : The changes will be saved and you will be redirected to "Credentials" tab. If you are not redirected automatically, click on the "Credentials" tab and you will see "Create Credentials" dropdown button. Click on the button, and select "OAuth client ID" option



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_KpsK57GFSxVueteyBunu02xJTak5m
01";
});
```

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_OclfNAprk1</State>
<Url>
/api/Account/ExternalLogin?provider=Google&response_type=token&client_id=self&redi
rect_uri=http%3A%2F%2Flocalhost%3A61358%2F&state=6Phc_u0Xkj3opJ9TymPhw9olZ
V_zB6Pjv_OclfNAprk1
</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.js** file.

```
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_KpsK57GFSxVueteyBunu02xJTak5m
01";
        }
        });

}
```

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**