**AUTENTICACIÓN CON AZURE ACTIVE DIRECTORY B2C –END TO END-**

# **REQUISITOS**

Cuenta en Azure

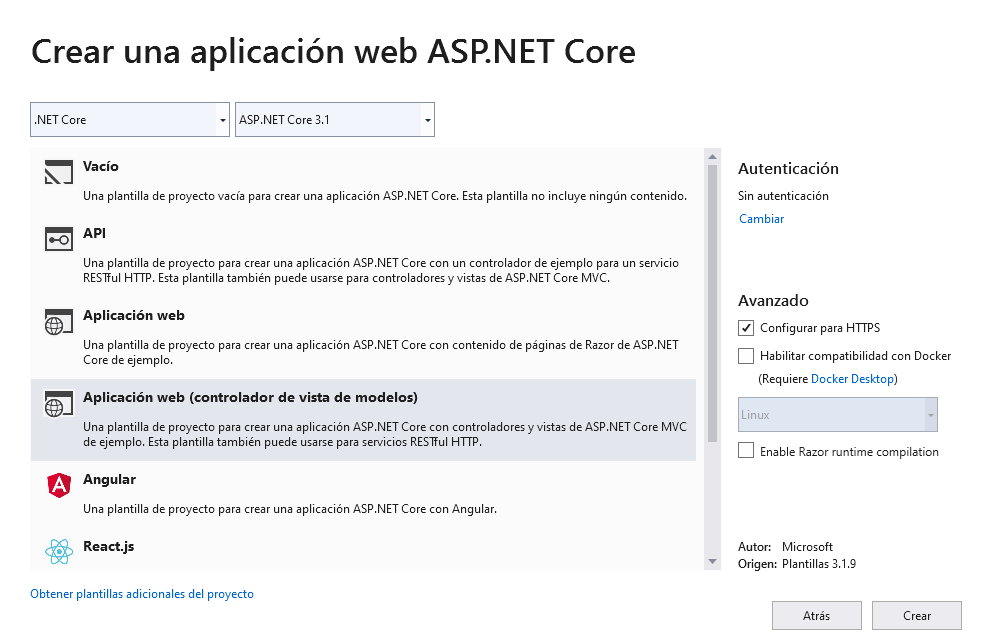
Recurso de Azure Active Directory B2C

Visual Studio 2019

# **AUTENTICACIÓN DE UNA APLICACIÓN WEB CON ASP.NET CORE Y AADB2C**

## **CREACIÓN DE UNA APLICACIÓN WEB CON ASP.NET CORE 3.1**

**Crear una aplicación web (MVC), .NET Core 3.1 o superior, sin autenticación y configurar para HTPS.**



**Instalar los paquetes Nuget necesarios:**

Microsoft.AspNetCore.Session

Microsoft.Identity.Web

Microsoft.Identity.Web.UI

Newtonsoft.Json

**Configurar el startup.cs para agregar todas las dependencias necesarias.**

Configuración de sesiones y cookies.

services.AddDistributedMemoryCache();

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.Unspecified;

// Handling SameSite cookie according to https://docs.microsoft.com/en-us/aspnet/core/security/samesite

options.HandleSameSiteCookieCompatibility();

});

Configuración de Azure Active Directory B2C y el API de backend

// Configuration to sign-in users with Azure AD B2C

services.AddMicrosoftIdentityWebAppAuthentication(Configuration, "AzureAd")

.EnableTokenAcquisitionToCallDownstreamApi(new string[] { Configuration["TestApi:Scope"] })

.AddInMemoryTokenCaches();

Agregar la política de autenticación para los controladores y el controlador para la autenticación

services.AddControllersWithViews(options =>

{

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

options.Filters.Add(new AuthorizeFilter(policy));

}).AddMicrosoftIdentityUI();

services.AddRazorPages();

Agregar OpenID con la configuración de Azure

//Configuring appsettings section AzureAdB2C, into IOptions

services.AddOptions();

services.Configure<OpenIdConnectOptions>(Configuration.GetSection("AzureAd"));

En el método Configure() agregar las siguientes configuraciones

app.UseCookiePolicy();

app.UseAuthentication();

app.UseAuthorization();

En el método Configure() modificar el método app.UseEndPoints() de la siguiente forma

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

endpoints.MapRazorPages();

});

**Modificar el HomeController para agregar la autenticación**

En el HomeController agregar la notación Authorize

[Authorize]

public class HomeController : Controller

{

Agregar e instanciar las siguientes propiedades en el controlador.

private readonly ITokenAcquisition \_tokenAcquisition;

private IConfiguration Configuration { get; }

public HomeController(ILogger<HomeController> logger, IConfiguration configuration, ITokenAcquisition tokenAcquisition)

{

\_logger = logger;

\_tokenAcquisition = tokenAcquisition;

Configuration = configuration;

}

En el método Index() agregar la captura del usuario logueado

var user = HttpContext.User;

ViewData["User"] = user;

**Modificar las vistas para mostrar info del usuario logueado**

Mostrar la info de los Claims del usuario logeado en la página Index.cshtml

@{

var user = ViewData["User"] as ClaimsPrincipal;

}

<table class="table table-striped table-bordered table-condensed table-hover">

<tr>

<th>ClaimType</th>

<th>Value</th>

</tr>

@foreach (var claim in user.Claims)

{

<tr>

@{

if (claim.Type == "groups")

{

<td><b>@claim.Type</b></td>

}

else

{

<td>@claim.Type</td>

}

}

<td>@claim.Value</td>

</tr>

}

</table>

Agregar una página plantilla para el login y logout del usuario que se llame \_LoginPartial

@using Microsoft.Identity.Web

@if (User.Identity.IsAuthenticated)

{

<ul class="nav navbar-nav navbar-right">

<li class="navbar-text">Hello @User.GetDisplayName()!</li>

<li class="navbar-btn">

<form method="get" **asp-area**="MicrosoftIdentity" **asp-controller**="Account" **asp-action**="EditProfile">

<button type="submit" class="btn btn-primary" style="margin-right:5px">Edit Profile</button>

</form>

</li>

<li><a **asp-area**="MicrosoftIdentity" **asp-controller**="Account" **asp-action**="SignOut">Sign out</a></li>

</ul> }

else

{

<ul class="nav navbar-nav navbar-right">

<li><a **asp-area**="MicrosoftIdentity" **asp-controller**="Account" **asp-action**="SignIn">Sign in</a></li>

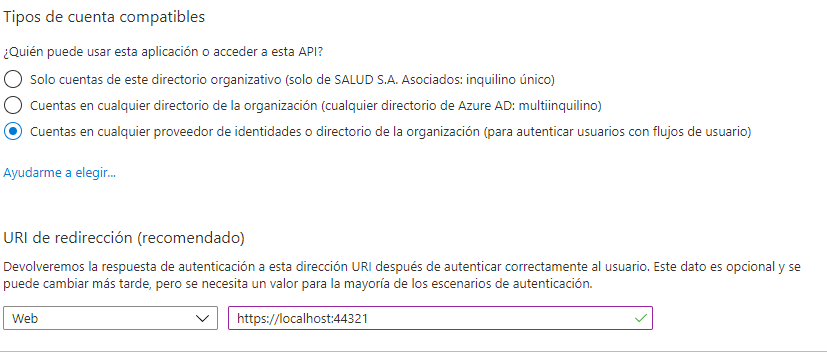
</ul>}

Agregar la página parcial en la página \_Layout

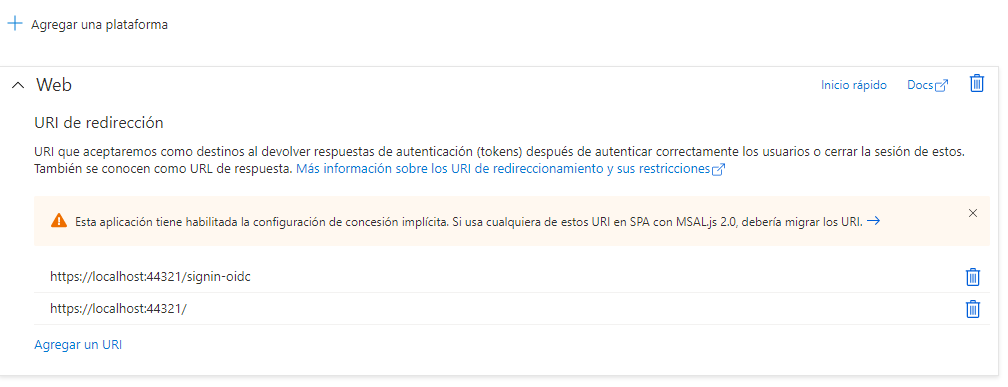
<**partial** **name**="\_LoginPartial" />

**Registrar la aplicación cliente (front) en Azure Active Directory B2C**

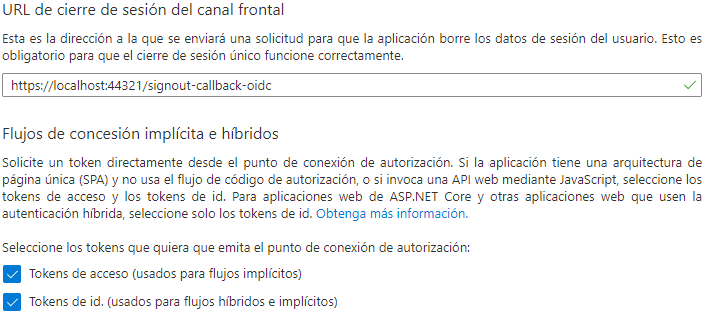
Registrar la aplicación



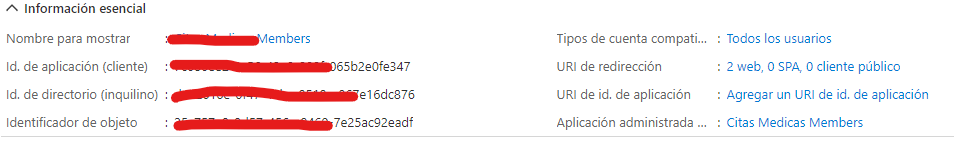
Agregar una plataforma web o SPA y agregar las url de sign in y sign ut



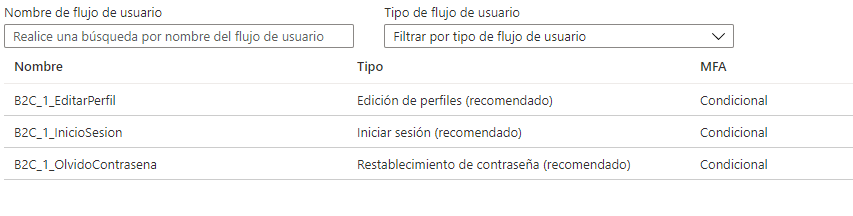
Habilitar los tokens de acceso y token id en la opción de “Autenticación”



Copiar los ID de cliente (aplicación) e inquilino (directorio) para configurarlos en la aplicación



Copiar los nombres de los flujos de usuario o políticas del ADB2C para configurarlos en la aplicación. (Cómo crear una política de usuario: https://www.youtube.com/watch?v=nWJ3m82536A)



**Configurar la aplicación ASP.NET Core**

En el archivo appsettings.json agregar la siguiente configuración con la info de AADB2C

"AzureAd": {

"Instance": "https://{tenant}.b2clogin.com",

"Domain": "{tenant}.onmicrosoft.com",

"ClientId": "...0fe347",

"TenantId": "...16dc876",

"CallbackPath": "/signin-oidc",

"SignedOutCallbackPath ": "/signout-callback-oidc",

"SignUpSignInPolicyId": "B2C\_1\_...",

"ResetPasswordPolicyId": "B2C\_1\_...",

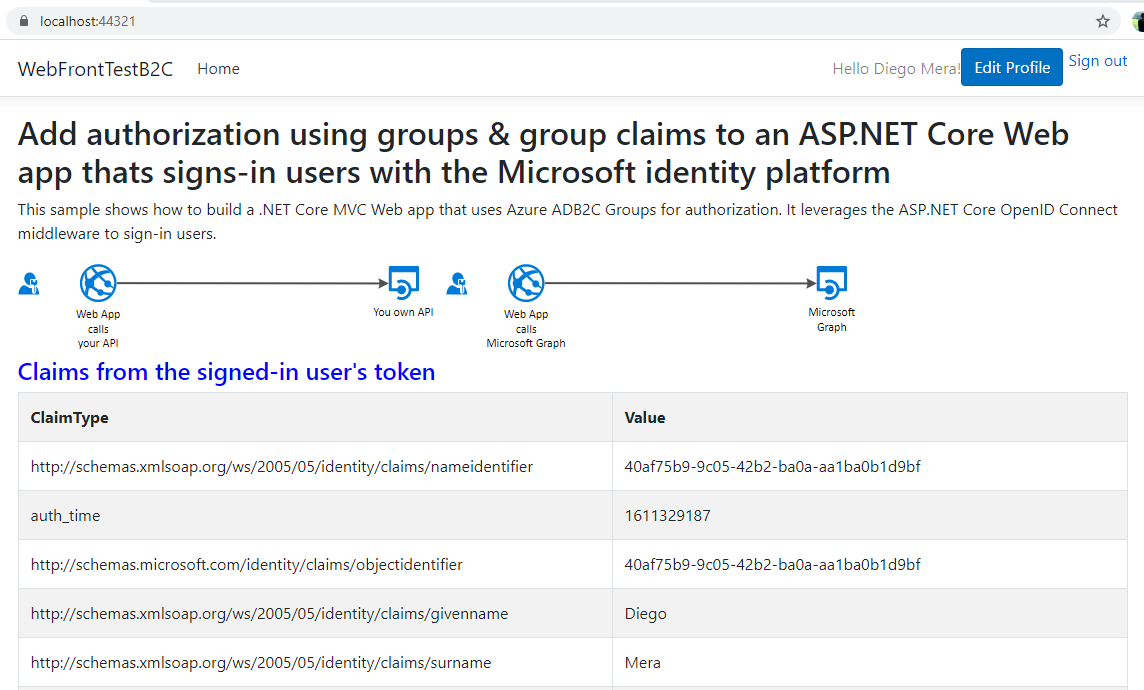
"EditProfilePolicyId": "B2C\_1\_...",

// To call an API

"ClientSecret": ""

}

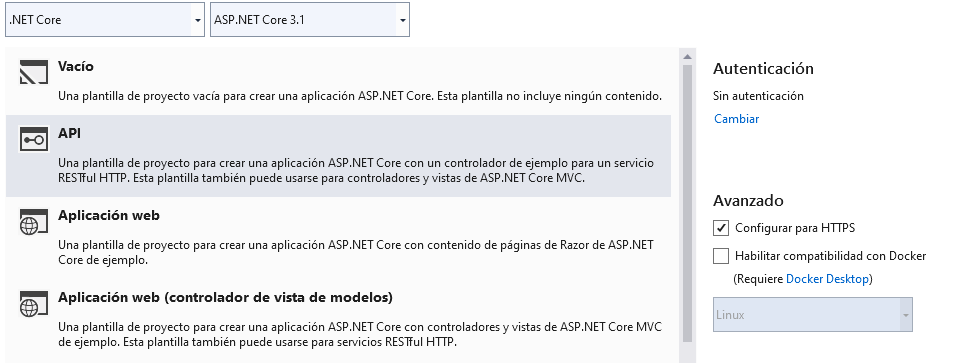
**Ejecutar la aplicación**



# **AUTENTICACIÓN DE UNA API REST CON ASP.NET CORE Y AADB2C**

## **CREACIÓN DE UNA APLICACIÓN WEB API CON ASP.NET CORE 3.1**

**Crear una aplicación Web API (MVC), .NET Core 3.1 o superior, sin autenticación y configurar para HTPS.**



**Instalar los paquetes Nuget necesarios:**

Microsoft.Identity.Web

Microsoft.Identity.Web.UI

**Configurar el startup.cs para agregar todas las dependencias necesarias.**

Agregar autenticación de API con Azure Active Directory B2C

// Adds Microsoft Identity platform (AAD v2.0) support to protect this Api

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddMicrosoftIdentityWebApi(options =>

{

Configuration.Bind("AzureAdB2C", options);

options.TokenValidationParameters.NameClaimType = "name";

},

options => { Configuration.Bind("AzureAdB2C", options); });

En el método Configure() agregar las siguientes configuraciones

app.UseAuthentication();

app.UseAuthorization();

**Modificar el WeatherForecastController (API Rest creado por la plantilla) para agregar la autenticación**

En el HomeController agregar la notación Authorize

[Authorize]

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

Agregar la siguiente propiedad a la clase

// The Web API will only accept tokens 1) for users, and 2) having the "access\_as\_user" scope for this API

static readonly string[] scopeRequiredByApi = new string[] { "access\_as\_user" };

En el método Get agregar la siguiente línea para permitir invocaciones solo para los ámbitos (scope) definidos en la variable scopeRequiredByApi

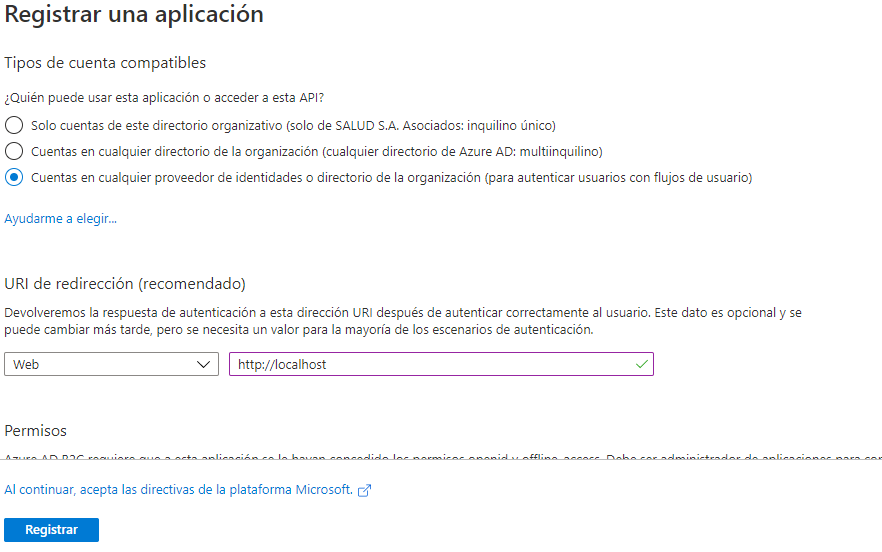
public IEnumerable<WeatherForecast> Get()

{

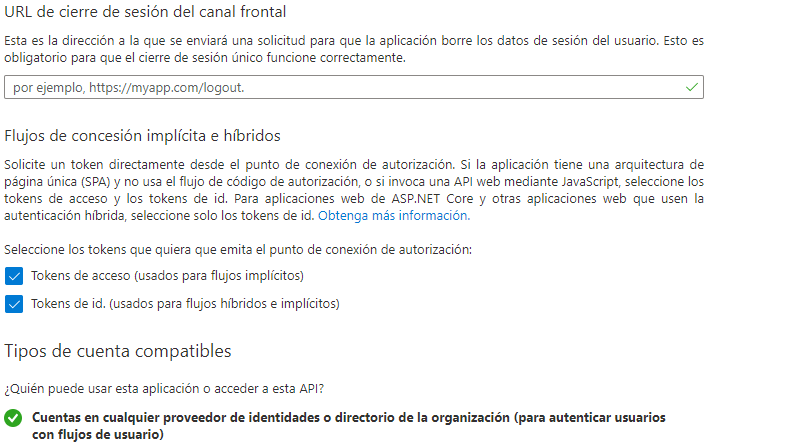
HttpContext.VerifyUserHasAnyAcceptedScope(scopeRequiredByApi);

**Registrar la aplicación API (backend) en Azure Active Directory B2C**

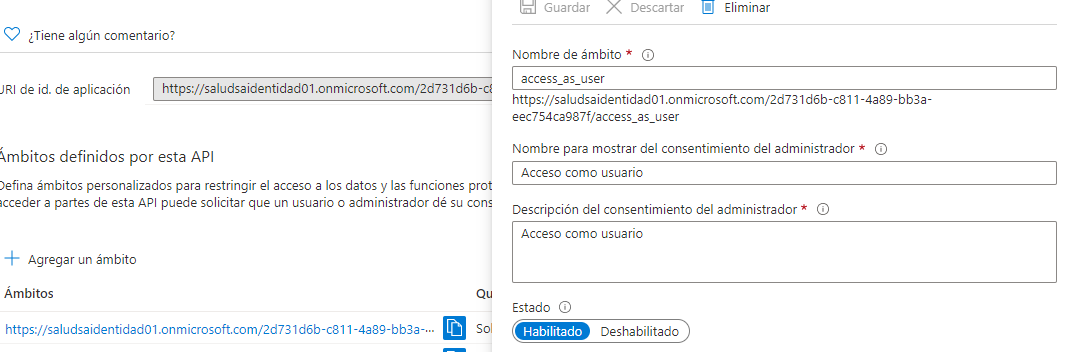
Registrar la aplicación



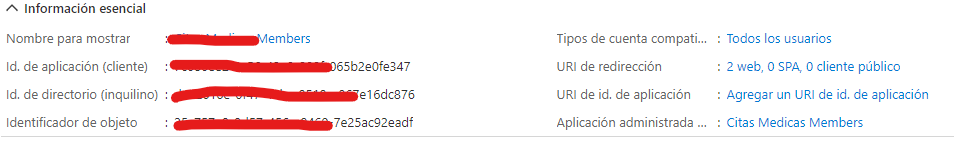
Habilitar los tokens de acceso y token id en la opción de “Autenticación”



Exponer una API y un ámbito en el registro de la aplicación en la opción de “Exponer API”



Copiar los ID de cliente (aplicación) e inquilino (directorio) para configurarlos en la aplicación



**Configurar la aplicación ASP.NET Core**

En el archivo appsettings.json agregar la siguiente configuración con la info de AADB2C

"AzureAdB2C": {

"Instance": "https://{tenant}.b2clogin.com",

"Domain": "{tenant}.onmicrosoft.com",

"ClientId": "...eec754ca987f",

"TenantId": "...a867e16dc876",

"SignedOutCallbackPath ": "/signout/B2C\_1\_InicioSesion",

"SignUpSignInPolicyId": "B2C\_1\_InicioSesion",

"ResetPasswordPolicyId": "B2C\_1\_OlvidoContrasena",

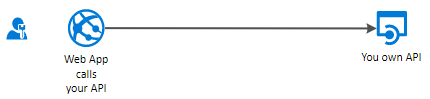
"EditProfilePolicyId": "B2C\_1\_EditarPerfil", // Optional profile editing policy

//"CallbackPath": "/signin/B2C\_1\_sign\_up\_in" // defaults to /signin-oidc

"ClientSecret": ""

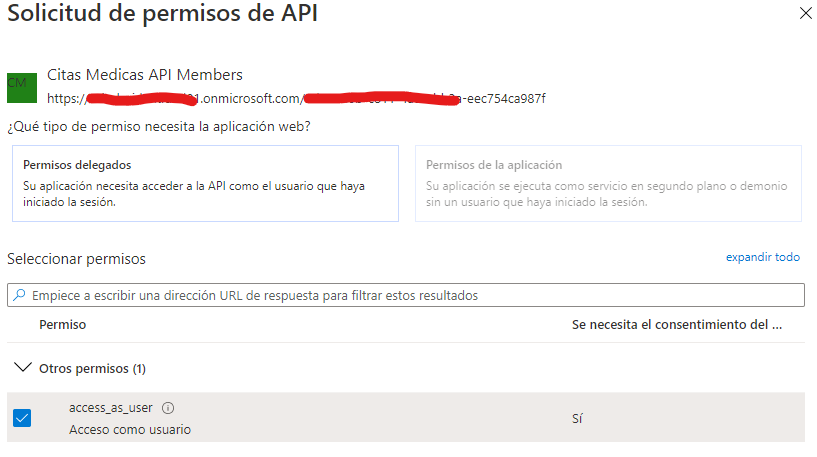
}

## **CONSUMO DE LA API DESDE LA APP WEB CLIENTE (FRONT)**



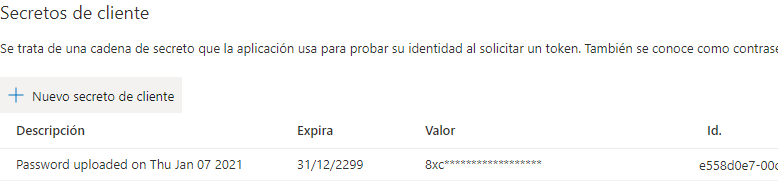
**En el registro de la aplicación cliente (front) agregar los permisos a la API expuesta**

En el registro de la aplicación cliente agregar la API y el ámbito (scope) en la opción “Permisos de API”



Conceder permisos con la opción “Conceder consentimiento de administrador para {tenant}

Agregar un secreto en la opción “Certificados y Secretos” y copiar el valor generado



**Configurar la aplicación cliente para que pueda consumir la API**

En el archivo appsettings.json agregar el secreto generado y copiado anteriormente en la sección con los valores de Active Directory

"ClientSecret": "8xcP.KsnfG3l5HUXRU\_r~.Bc~XXXXXXX"

Agregar la configuración con la URL de la API y el ámbito (scope) previamente agregado en el registro de la API en Active Directory B2C

"TestApi": {

"Scope": "https://{tenan}.onmicrosoft.com/{id}/access\_as\_user",

"TestApiBaseAddress": "http://localhost:49157"

},

En el controlador HomeController agregar el siguiente método PrepareAuthenticatedClient() para crear un objeto HttpClient y agregar a la cabecera un token generado para el consumo de la API

private async Task<HttpClient> PrepareAuthenticatedClient()

{

try

{

var accessToken = await \_tokenAcquisition.GetAccessTokenForUserAsync(new[] { Configuration["TestApi:Scope"] });

Debug.WriteLine($"access token-{accessToken}");

HttpClient \_httpClient = new HttpClient();

\_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);

\_httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

return \_httpClient;

}

catch (Exception ex)

{

return new HttpClient();

}

}

Agregar el siguiente método GetTest() para realizar la llamada a la API y devolver el resultado

private async Task<string> GetTest(HttpClient httpClient)

{

try

{

string testApiBaseAdress = Configuration["TestApi:TestApiBaseAddress"];

var response = await httpClient.GetAsync($"{ testApiBaseAdress}/weatherforecast");

if (response.StatusCode == HttpStatusCode.OK)

{

var content = await response.Content.ReadAsStringAsync();

return content;

}

else

{

return response.StatusCode.ToString();

}

}

catch (Exception ex)

{

return ex.Message;

}

}

En el método Index() agregar el siguiente código para el llamado a la API

//CALL API

var httpClient = PrepareAuthenticatedClient().GetAwaiter().GetResult();

var resultApi = GetTest(httpClient).GetAwaiter().GetResult();

ViewData.Add("ResultApi", resultApi);

Modificar el archivo Index.cshtml para mostrar el resultado del llamado a la API en pantalla

<table class="table table-striped table-bordered table-condensed table-hover">

<tr>

<td colspan="2">Test API</td>

<td colspan="2"></td>

</tr>

<tr>

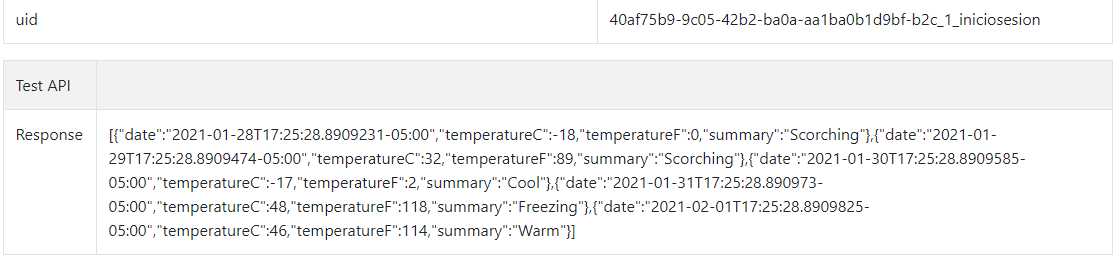
<td colspan="2">Response</td>

<td colspan="2">@ViewData["ResultApi"]</td>

</tr>

</table>

**Ejecutar las aplicaciones Web y Web API para probar la invocación a la API**



# **CONSULTAR DATOS DEL USUARIO AUTENTICADO USANDO GRAPH DESDE LA WEB API**

## **CONSUMO DE UN METODO DE GRAPH DESDE LA WEB API**

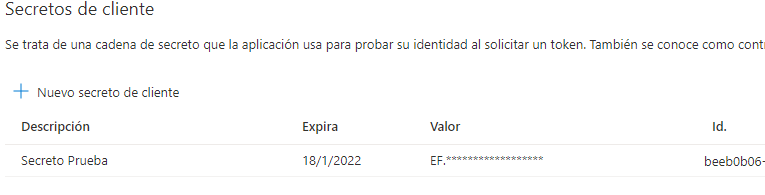
**Agregar los permisos al consumo de Graph en el registro de la API en Active Directory B2C**

En el registro de la API, en la sección “Permisos de API” agregar permisos al consumo de Graph de la siguiente forma:



Agregar permisos para los ámbitos GroupMember.Read.All y User.Read.All y conceder consentimiento de administrador con la opción “Conceder Consentimiento de Administrador para {tenant}”

Agregar un secreto en la opción “Certificados y Secretos” y copiar el valor generado



**Agregar el consumo de Graph en la aplicación Web API**

En la aplicación Web API agregar los siguientes paquetes Nuget

Microsoft.Graph

Microsoft.Graph.Auth

**Agregar una clase GraphHelper para colocar los métodos para obtener información del usuario autenticado.**

Agregar el siguiente método estático para consultar los grupos a los que pertenece el usuario autenticado

private static async Task<Dictionary<string, string>> ProcessUserGroupsB2C(GraphServiceClient graphClient, string userId)

{

Dictionary<string, string> groupClaims = new Dictionary<string, string>();

try

{

// Before instatntiating GraphServiceClient, the app should have granted admin consent for 'GroupMember.Read.All' permission.

//var graphClient = context.HttpContext.RequestServices.GetService<GraphServiceClient>();

if (graphClient == null)

{

Console.WriteLine("No service for type 'Microsoft.Graph.GraphServiceClient' has been registered in the Startup.");

}

else

{

// The properties that we want to retrieve from MemberOf endpoint.

string select = "id,displayName,onPremisesNetBiosName,onPremisesDomainName,onPremisesSamAccountNameonPremisesSecurityIdentifier";

IUserMemberOfCollectionWithReferencesPage memberPage = new UserMemberOfCollectionWithReferencesPage();

try

{

//Request to get groups and directory roles that the user is a direct member of.

memberPage = await graphClient.Users[userId].MemberOf.Request().Select(select).GetAsync().ConfigureAwait(false);

}

catch (Exception graphEx)

{

var exMsg = graphEx.InnerException != null ? graphEx.InnerException.Message : graphEx.Message;

Console.WriteLine("Call to Microsoft Graph failed: " + exMsg);

}

if (memberPage?.Count > 0)

{

// There is a limit to number of groups returned, below method make calls to Microsoft graph to get all the groups.

var allgroups = ProcessIGraphServiceMemberOfCollectionPage(memberPage);

if (allgroups?.Count > 0)

{

// Re-populate the `groups` claim with the complete list of groups fetched from MS Graph

foreach (Group group in allgroups)

{

// The following code adds group ids to the 'groups' claim. But depending upon your reequirement and the format of the 'groups' claim selected in

// the app registration, you might want to add other attributes than id to the `groups` claim, examples being;

// For instance if the required format is 'NetBIOSDomain\sAMAccountName' then the code is as commented below:

// groupClaims.Add(group.OnPremisesNetBiosName+"\\"+group.OnPremisesSamAccountName));

groupClaims.Add(group.Id, group.DisplayName);

}

}

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return groupClaims;

}

Agregar el siguiente método para obtener la información de trabajo del usuario

private static async Task<Dictionary<string, string>> ProcessUserJobInfoB2C(GraphServiceClient graphClient, string userId)

{

Dictionary<string, string> Jobinfo = new Dictionary<string, string>();

try

{

if (graphClient == null)

{

Console.WriteLine("No service for type 'Microsoft.Graph.GraphServiceClient' has been registered in the Startup.");

}

else

{

Microsoft.Graph.User result = new User();

try

{

result = await graphClient.Users[userId]

.Request().Select(e => new

{

e.DisplayName,

e.Id,

e.Identities,

e.JobTitle,

e.CompanyName,

e.Department

}).GetAsync();

}

catch (Exception graphEx)

{

var exMsg = graphEx.InnerException != null ? graphEx.InnerException.Message : graphEx.Message;

Console.WriteLine("Call to Microsoft Graph failed: " + exMsg);

}

Jobinfo.Add("CompanyName", result.CompanyName);

Jobinfo.Add("Department", result.Department);

Jobinfo.Add("JobTitle", result.JobTitle);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return Jobinfo;

}

Agregar el siguiente método que obtiene la info de los grupos y del trabajo del usuario y los retorna en un solo diccionario.

public static async Task<Dictionary<string, string>> GetSignedInUsersGroupsB2C(GraphServiceClient graphClient, string userId)

{

Dictionary<string, string> groupClaims = new Dictionary<string, string>();

groupClaims = await ProcessUserGroupsB2C(graphClient, userId);

var jobInfo = await ProcessUserJobInfoB2C(graphClient, userId);

foreach (var item in jobInfo)

{

groupClaims.Add(item.Key, item.Value);

}

return groupClaims;

}

En el controlador WeatherForecastController agregar el siguiente método para hacer el llamado al Graph y devolver la info de los grupos y del trabajo del usuario

[HttpGet("GetGraph")]

public Dictionary<string,string> GetGraph()

{

//HttpContext.VerifyUserHasAnyAcceptedScope(scopeRequiredByApi);

var user = HttpContext.User;

var configAz = Configuration.GetSection("AzureAdB2C");

IConfidentialClientApplication confidentialClientApplication = ConfidentialClientApplicationBuilder

.Create(configAz.GetSection("ClientId").Value)

.WithTenantId(configAz.GetSection("TenantId").Value)

.WithClientSecret(configAz.GetSection("ClientSecret").Value)

.Build();

ClientCredentialProvider authProvider = new ClientCredentialProvider(confidentialClientApplication);

// Set up the Microsoft Graph service client with client credentials

GraphServiceClient graphClient = new GraphServiceClient(authProvider);

var userId = user.Claims.Where(x => x.Type == "http://schemas.microsoft.com/identity/claims/objectidentifier").Select(x => x).FirstOrDefault().Value;

var grupos = GraphHelper.GetSignedInUsersGroupsB2C(graphClient, userId).GetAwaiter().GetResult();

return grupos;

}

En el archivo appsettings.json agregar el secreto generado y copiado anteriormente en la sección con la configuración de Active Directory B2C

"ClientSecret": "EF.6.6A09\_Bq3AgEaXXXXXXXX"

## **CONSUMO DE LA API DESDE LA APLICACIÓN WEB (FRONT)**

Agregar el llamado al nuevo método en la API desde la aplicación Web

En el HomeController agregar el siguiente método GetTestGraph para hacer el llamado al nuevo método en la API

private async Task<Dictionary<string, string>> GetTestGraph(HttpClient httpClient)

{

try

{

string testApiBaseAdress = Configuration["TestApi:TestApiBaseAddress"];

var response = await httpClient.GetAsync($"{ testApiBaseAdress}/weatherforecast/GetGraph");

if (response.StatusCode == HttpStatusCode.OK)

{

var content = await response.Content.ReadAsStringAsync();

var result = JsonConvert.DeserializeObject<Dictionary<string, string>>(content);

return result;

}

else

{

return new Dictionary<string, string>();

}

}

catch (Exception ex)

{

return new Dictionary<string, string>();

}

}

En el método Index() agregar el llamado al método GetTestGraph()

//CALL API GRAPH

var resultApiGraph = GetTestGraph(httpClient).GetAwaiter().GetResult();

ViewData.Add("ResultApiGraph", resultApiGraph);

Modificar la página Index.cshtml para mostrar los resultados devueltos por la API

@{

Dictionary<string, string> resultApiGraph = new Dictionary<string, string>();

if (ViewData.ContainsKey("ResultApiGraph"))

{

resultApiGraph = ViewData["ResultApiGraph"] as Dictionary<string, string>;

}

}

<table class="table table-striped table-bordered table-condensed table-hover">

<tr>

<td colspan="2">API Graph test</td>

<td colspan="2"></td>

</tr>

@if (resultApiGraph != null)

{

@foreach (var group in resultApiGraph)

{

<tr>

<td colspan="2">@group.Key</td>

<td colspan="2">@group.Value</td>

</tr>

}

}

</table>

**Probar la aplicación**

Ejecutar tanto la aplicación Web (Front) como la aplicación Web API (backnd)

