**Secure gRPC service with .NET Core and Azure Active Directory**

what it would take to secure a gRPC solution with the new Microsoft.Identity.Web on the server. [Microsoft.Identity.Web](https://github.com/AzureAD/microsoft-identity-web" \t "_blank) is a new library for .NET Core (3.1 or later) that makes authentication and token management a breeze. The library is going to hit GA soon so now is a good time to experiment and see how to get our ASP.NET Core apps working with it.

**Prerequisites**

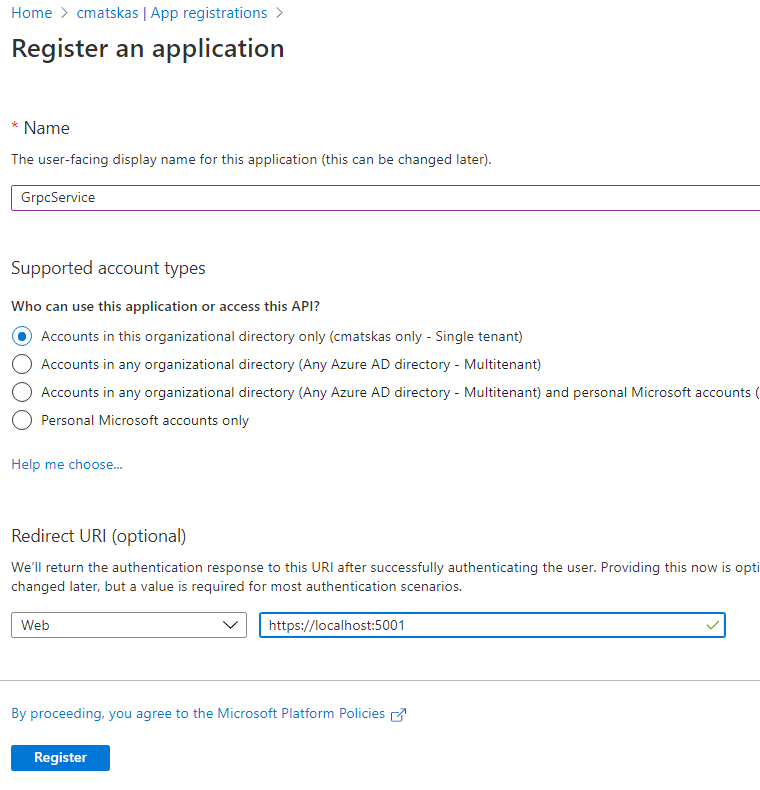
* .NET Core 3.1
* Your favorite IDE (I'll be using VS Code)
* Azure AD ([get a FREE M365 dev tenant](https://developer.microsoft.com/en-us/microsoft-365/dev-program))

**Create the Azure AD App registrations**

gRPC services, although they act as APIs, can't be access by the browser. Therefore you need a client app that can be used to call our gRPC service. To be able to authenticate the user in the client app and validate the access tokens in the gRPC service, we will need 2 app registrations.

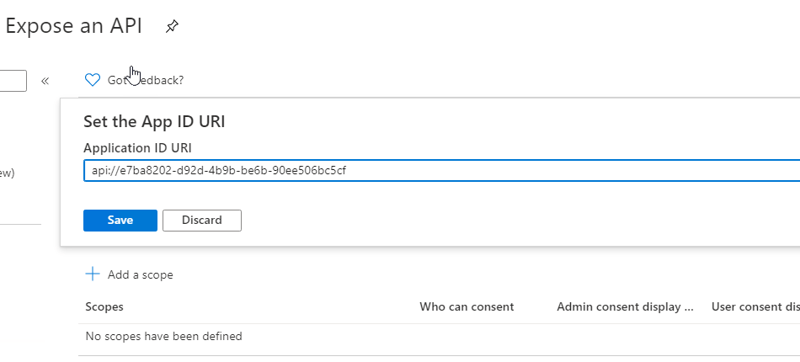
**gRPC Service app registration**

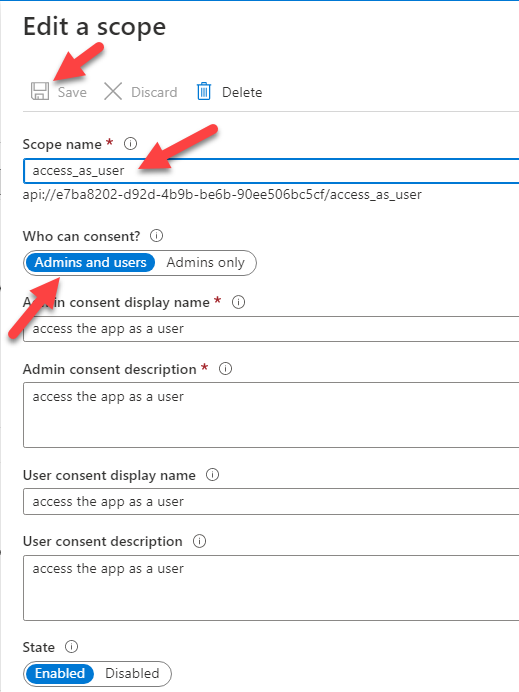
Head to the Azure AD portal and select the **App registration** tab. Click the **New registration** button and use the following settings to create the server app:

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fa31tn6og6j8jp195v4c3.png)

The Redirect URI assumes that we will be using the default port for Kestrel to run our gRPC service.

Our client app needs to know about the server app. Therefore, we need navigate to the **Expose an API** tab, set the App ID URI (leave the defaults) and then add a new scope:

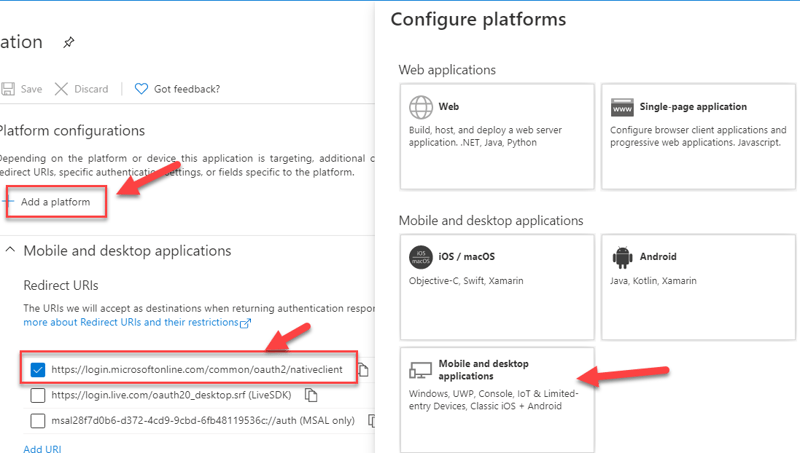
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2F4lszrdqbh4afoiu1n4wg.png)

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Ffdc1cm7kzqkl99t12e19.png)

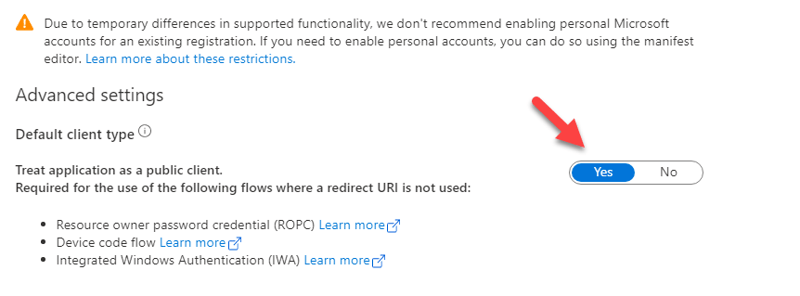
That's all we need in our servcie app

**gRPC client app registration**

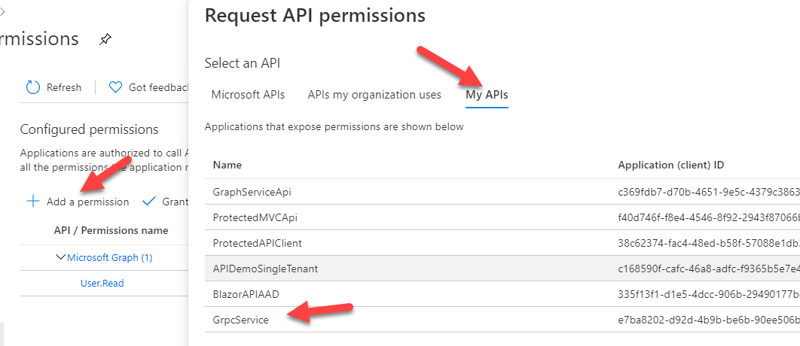
Back to the **App Registrations** tab, we need to create a new app registration. Select **Single org** and leave the Redirect URi blank. Once the app is created, head to the **Authentication** tab to **Add a platform**. Since our client app will be a console .NET Core app, we should choose the **Mobile and desktop applications** and use the https://..../nativeclient for the **Redirect URI**.

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2F3soo7uom4n7pgtn94kjj.png)

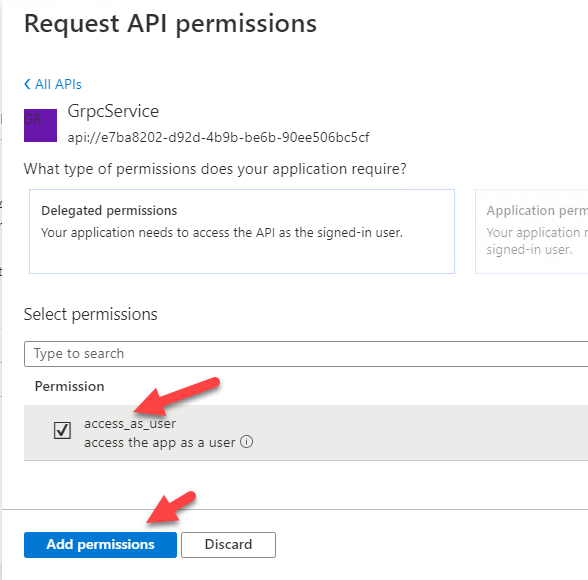
In the **Advanced settings** on the same tab, update the **Default client type** as per the image below:

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fe4pu2xjsx8kpxi4rtqb8.png)

Next, we need to add the API permissions for our gRPC service. Open the **API Permissions** tab and click the **Add a permission**. Select the **My APIs** and click on the **GrpcService** app (this could be different for you depending on the app name you used to register your service app).

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fpoagysrcgt70upz45e1l.png)

Select the right permission and press **Add permission**

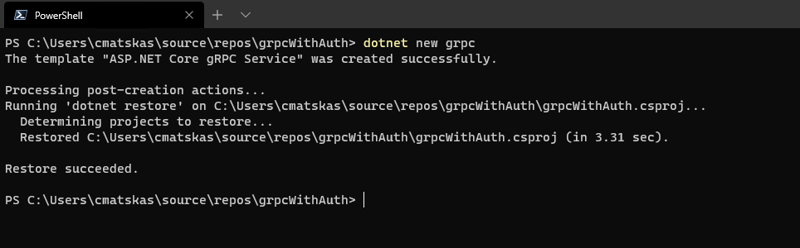
[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2F7hkcmo0d2mqiv0orn0mr.png)

This is all we need to register the client app

**Create the gRPC Service in .NET Core**

We will be using the .NET CLI to create our gRPC service

dotnet new grpc

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fudsio7mh7kfz4sr86zox.png)

We now need to add a couple of NuGet packages for enabling authentication. Open the \*.csproj file and add the following package reference

<PackageReference Include="Microsoft.Identity.Web" Version="0.2.3-preview" />

The gRPC service needs to know how to authenticate incoming access token so we have to update the appsettings.json file to include the grpc service app registration details as per the example below:

"AzureAd": {

"Instance": "https://login.microsoftonline.com/",

"Domain": "cmatskas.onmicrosoft.com",

"TenantId": "b55f0c51-0000-0000-0000-33569b247796",

"ClientId": "e7ba8202-0000-0000-0000-90ee506bc5cf"

}

Next, we will edit the Startup.cs file to add the authentication code. Add a constructor to get access to the IConfiguration object.

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

The authentication middleware goes into the ConfigureServices() method. Update it with the following code:

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddMicrosoftWebApi(Configuration);

services.AddAuthorization();

Finally, let's not forget the Configure() method which needs to be update to include authN and authZ

app.UseAuthentication();

app.UseAuthorization();

The final step is to update the GreetService.cs class to ensure that any incoming requests are authenticated and have the right access token.

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

[Authorize]

public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)

{

var httpContext = context.GetHttpContext();

httpContext.VerifyUserHasAnyAcceptedScope(scopeRequiredByAPI);

return Task.FromResult(new HelloReply

{

Message = "Hello " + request.Name

});

}

The action above has an [Authorize] attribute and then we use the VerifyUserHasAnyAcceptedScope(<scopeArray>) method that is part of the new Microsoft.Identity.Web to ensure that the request has the appropriate scopes.

This is all we need to do to create an authenticate gRPC service. As you can see, the new Microsoft.Identity.Web provides a very powerful and simplified API to validate tokens.

**Create the gRPC client (console) app**

To keep things simple, we will create a .NET Core console app to speak to our .NET Core gRPC service. On a new directory, run

dotnet new console

We now need to add a number of NuGet packages, and a reference to our .proto file, which currently resides in our gRPC service directory.

NOTE: I've placed both projects (service and client) side by side so the path is relative to the location of my projects. This could be different for you

Open the \*.csproj file and add the following xml

<ItemGroup>

<PackageReference Include="Google.Protobuf" Version="3.13.0" />

<PackageReference Include="Grpc.Net.ClientFactory" Version="2.31.0" />

<PackageReference Include="Grpc.Tools" Version="2.31.0">

<PrivateAssets>all</PrivateAssets>

<IncludeAssets>runtime; build; native; contentfiles; analyzers</IncludeAssets>

</PackageReference>

<PackageReference Include="Microsoft.Extensions.Configuration.CommandLine" Version="3.1.7" />

<PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="3.1.7" />

<PackageReference Include="Microsoft.Identity.Client" Version="4.17.1" />

</ItemGroup>

<ItemGroup>

<Protobuf Include="../grpcWithAuth/Protos/greet.proto" GrpcServices="Client" Link="Protos/greet.proto" />

</ItemGroup>

The first ItemGroup contains all the references we need to work with gRPC and the Microsoft.Identity.Client library that will be used to authenticate users.

We also need to add a new appsettings.json file where we can store the Azure AD settings for our client app, including the scopes we need for our gRPC service. The file should have the following json

{

"AzureAd":{

"ClientId": "28f7d0b6-d372-4cd9-9cbd-6fb48119536c",

"TenantId": "b55f0c51-61a7-45c3-84df-33569b247796"

},

"scope": "api://e7ba8202-d92d-4b9b-be6b-90ee506bc5cf/access\_as\_user"

}

To keep our code clean, we'll add a new class DeviceCodeAuthProvider.cs to deal with the user authentication. The code need for that class is attached below:

using System;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Threading.Tasks;

using Microsoft.Extensions.Configuration;

using Microsoft.Identity.Client;

public class DeviceCodeAuthProvider

{

private IPublicClientApplication msalClient;

private IAccount userAccount;

public DeviceCodeAuthProvider(IConfiguration config)

{

msalClient = PublicClientApplicationBuilder

.Create(config["AzureAd:ClientId"])

.WithAuthority(AadAuthorityAudience.AzureAdMyOrg, true)

.WithTenantId(config["AzureAd:TenantId"])

.Build();

}

public async Task<string> GetAccessToken(string[] scopes)

{

if (userAccount == null)

{

try

{

var result = await msalClient.AcquireTokenWithDeviceCode(scopes, callback => {

Console.WriteLine(callback.Message);

return Task.FromResult(0);

}).ExecuteAsync();

userAccount = result.Account;

return result.AccessToken;

}

catch (Exception exception)

{

Console.WriteLine($"Error getting access token: {exception.Message}");

return null;

}

}

else

{

var result = await msalClient

.AcquireTokenSilent(scopes, userAccount)

.ExecuteAsync();

return result.AccessToken;

}

}

}

Finally, we need to update the Program.cs to autneticate the user and call the gRPC service:

using System;

using System.Threading.Tasks;

using Grpc.Core;

using Grpc.Net.Client;

using grpcWithAuth;

using Microsoft.Extensions.Configuration;

namespace grpcClient

{

class Program

{

private static IConfiguration configuration;

static async Task Main(string[] args)

{

LoadAppSettings();

var authProvider = new DeviceCodeAuthProvider(configuration);

var token = await authProvider.GetAccessToken(new string[] {configuration["scope"]});

var channel = GrpcChannel.ForAddress("https://localhost:5001");

var client = new Greeter.GreeterClient(channel);

var headers = new Metadata();

headers.Add("Authorization", $"Bearer {token}");

var request = new HelloRequest()

{

Name = "SpongeBob"

};

var reply = await client.SayHelloAsync(request, headers);

Console.WriteLine(reply.Message);

}

static void LoadAppSettings()

{

configuration = new ConfigurationBuilder()

.SetBasePath(System.IO.Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json")

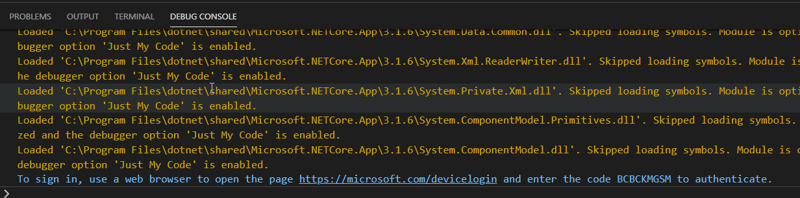
.Build();

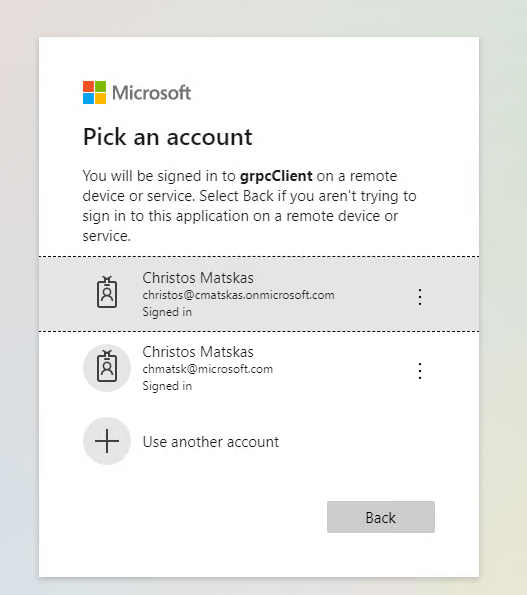
}

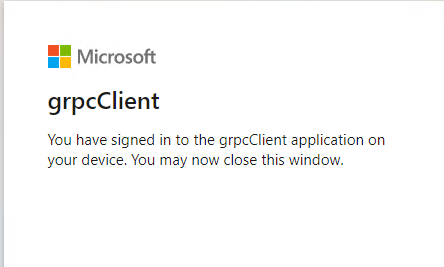
}

}

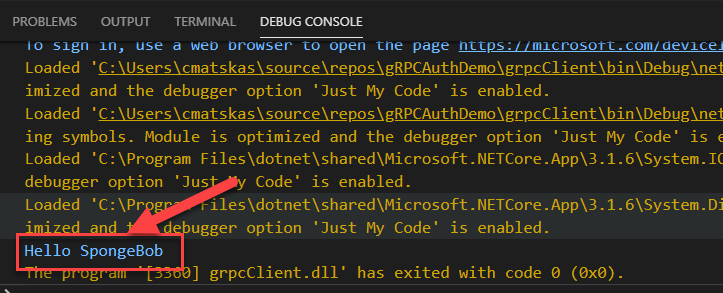
Let's run both projects and see what happens :). First the gRPC service and then the client

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fjo43uo06lx5o5ttd2xcw.png)

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fm27rl2ybv4cnvfjpe0ep.png)

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2F4myw5wxw2w6pd5wqvsrz.png)

After successfully authenticating, we can see the call to the gRPC service executing and the right result returned to our console app:

[](https://media2.dev.to/dynamic/image/width=800%2Cheight=%2Cfit=scale-down%2Cgravity=auto%2Cformat=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fi%2Fcja02kwh9a4le9rjaqcm.png)

Sweet sauce!!! Everything is working as expected

**Source code**

You can get the full source code for this project on [GitHub](https://github.com/cmatskas/gRPCAuthWithAzureAD)

**\**

**Summary**

This was a pretty straightforward setup. The authentication code was less than 8 lines for our gRPC service and we had to do a bit of work on the client app but nothing extravagant. All in all, the end-to-end implementation took little effort so I hope this shows how you can benefit from the .NET Core ecosystem to create powerful and fast gRPC services secured by Azure AD.