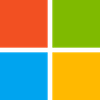
**ASP.NET Core gRPC and WPF on .NET Core with Azure AD Authorization/Authentication**

Microsoft

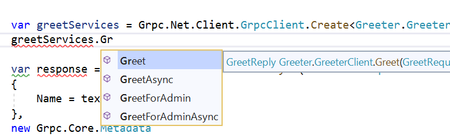
On ASP.NET Core 3.0(Still preview 9 now), there is a great RPC feature that is gRPC on ASP.NET Core!

Here is a document:

* [Introduction to gRPC on ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/grpc/?view=aspnetcore-3.0)

Its programming experience is really cool. Because it is type safe(I love it.)

So, we can use code complement feature to call gRPC services.



And .NET Core 3.0 supports developing desktop apps feature.

You can develop both of server side(with type safe by gRPC!) and client side(I love WPF! Type safe also!) using .NET Core 3.0.

gRPC on ASP.NET Core can use authentication/authorization features of ASP.NET Core.

It means you can protect gRPC using Azure AD and more. Authentication and authorization are important all of apps.

So, in this article, I will write about gRPC on ASP.NET Core and WPF on .NET Core, and protecting the apps using Azure AD Authentication/Authorization.

**Let's create a simple gRPC service**

gRPC services are defined as a .proto file.

Please read a following document about defails of .proto file:  
[gRPC services with C#](https://docs.microsoft.com/en-us/aspnet/core/grpc/basics?view=aspnetcore-3.0)

Create a solution, and create Sample.proto file at same location of solution folder.

syntax = "proto3";

option csharp\_namespace = "GrpcSampleApp";

import "google/protobuf/empty.proto";

package GrpcSample;

service Employees {

// for all users

rpc GetEmployees(google.protobuf.Empty) returns (GetEmployeeReply);

// for writer users

rpc AddEmployee(Employee) returns (AddEmployeeReply);

// for admin users

rpc DeleteEmployee(DeleteEmployeeRequest) returns (DeleteEmployeeReply);

}

message GetEmployeeReply {

repeated Employee employees = 1;

}

message AddEmployeeReply {

bool succeed = 1;

}

message DeleteEmployeeRequest {

string id = 1;

}

message DeleteEmployeeReply {

bool succeed = 1;

}

message Employee {

string id = 1;

string name = 2;

}

And create a project that type is gRPC Service, and then modify a Protobuf tag of the project file like below:

<Protobuf Include="..\Sample.proto" GrpcServices="Server" />

Delete Services/GreetService.cs file, and create Services/EmployeeService.cs to implement the service logic.

Ant then, write a code to the file like following link:

[EmployeeService.cs](https://github.com/runceel/gRPCSampleApps/blob/0a3d51da6f297cad0656a446c06b002716c14827/GrpcServiceApp/Services/EmployeeService.cs)

[Spoiler](https://techcommunity.microsoft.com/blog/modernworkappconsult/asp-net-core-grpc-and-wpf-on-net-core-with-azure-ad-authorizationauthentication/837781#community-837781-)

Final step! Change Configure method of Startup.cs to register the EmployeeService.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

// endpoints.MapGrpcService();

endpoints.MapGrpcService();

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Communication with gRPC endpoints must be made through a gRPC client. To learn how to create a client, visit: https://go.microsoft.com/fwlink/?linkid=2086909");

});

});

}

**Kestrel HTTP/2 Configuration**

gRPC is over HTTP/2. So, if we didn't create a project from gRPC Service project template, then we would have to configure appsettins.json to use HTTP/2.

{

"Logging": {

"LogLevel": {

"Default": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*",

"Kestrel": {

"EndpointDefaults": {

"Protocols": "Http2"

}

}

}

In this case, the configuration is already included by gRPC Service project template.

**Let's call the service from WPF on .NET Core**

Create a Class Library(.NET Core) project to generate client side code as GrpcServiceApp.ClientLib.

And add three packages from NuGet:

1. Google.Protobuf
2. Grpc.Net.Client (latest preview version)
3. Grpc.Tools

At next, add a Protobuf tag to GrpcServiceApp.ClientLib project file like below:

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<TargetFramework>netcoreapp3.0</TargetFramework>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Google.Protobuf" Version="3.9.1" />

<PackageReference Include="Grpc.Net.Client" Version="0.2.23-pre1" />

<PackageReference Include="Grpc.Tools" Version="2.23.0">

<PrivateAssets>all</PrivateAssets>

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

</PackageReference>

</ItemGroup>

<ItemGroup>

<Protobuf Include="..\Sample.proto" GrpcServices="Client" />

</ItemGroup>

</Project>

Important classes to call the service are GrpcChannel class and [Your service name].[Your service name]Client class.

In this case, I defined Employees service to Sample.proto file. So, the name is Employees.EmployeesClient.

At first, create GrpcChannel class from a server endpoint uri, and create Employees.EmployeesClient class using the GrpcChannel instance.

// create GrpcChannel from uri

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

// create a client using the channel instance.

var client = new Employees.EmployeesClient(channel);

// call some apis

var result = await client.GetEmployeesAsync(new Empty());

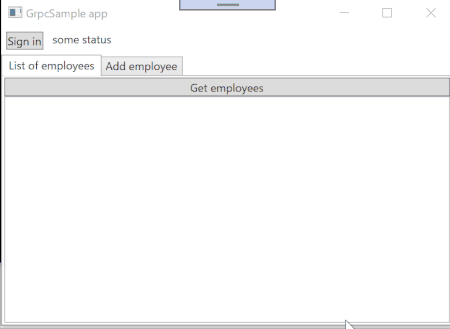
I implemented employees management client using the Employees.EmployeesClient class.

It is plane WPF app, for to easy to read/understand, I didn't use MVVM pattern, all logic are written at MainWindow.xaml.cs.

Source code can be looked from following link:

* [MainWindow.xaml](https://github.com/runceel/gRPCSampleApps/blob/2dc3b48ac1a136207036978bd5dcce7bdc5a7f66/GrpcServiceApp.Client/MainWindow.xaml)
* [MainWindow.xaml.cs](https://github.com/runceel/gRPCSampleApps/blob/2dc3b48ac1a136207036978bd5dcce7bdc5a7f66/GrpcServiceApp.Client/MainWindow.xaml.cs)

The app works like following:



It doesn't have authentication/authorization feature, yet.

**Let's add authentication/authorization feature!**

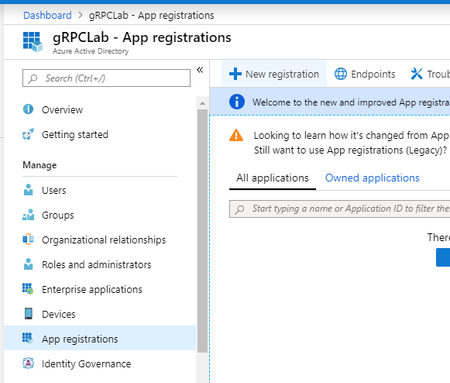
We learned how to create gRPC service and how to call gRPC service. At the next step,  we learn how to add authentication/authorization using Azure AD.

Create two apps to Azure AD tenant.

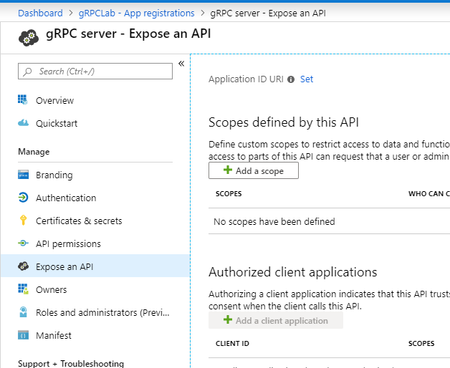
First one is for server app, second one is for client app.

**Create an app for server side**

Open Azure portal and then select Azure Active Directory. Select "App registrations" -> "New registration".

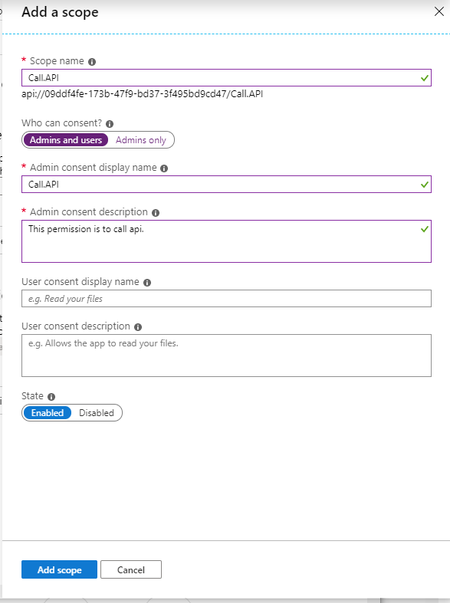


Enter an app name like "gRPC server", and then select a "Register" button. After created the app, would select "Expose an API" -> "Add a scope".



If you haven't create Applicatin ID URL, then would display a pane to create it. In this case, please select "Save and continue" button.

And input scope information, then select "Add scope" button.



And memo following items:

* Application ID
* Scope uri(api://application id/scope name)

At next, create roles to the app. Select "manifest" menu at left pane. So, you can see json.

There is "appRoles" property that is to define roles in an app.

The scheme of under the "appRoles" is as below:

{

"allowedMemberTypes": [

"User"

],

"description": "Administrators.",

"displayName": "Admins",

"id": "fc998089-b40c-40c1-af3c-161382ac6422",

"isEnabled": true,

"value": "Admins"

}

Please read a following document for details:

[How to: Add app roles in your application and receive them in the token](https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-add-app-roles-in-azure-ad-apps)

Added three roles for Users, Writers and Administrators.

"appRoles": [

{

"allowedMemberTypes": [

"User"

],

"description": "For users.",

"displayName": "Users",

"id": "927b2607-9226-4fbb-bc11-3ae7f90fdc84",

"isEnabled": true,

"value": "Users"

},

{

"allowedMemberTypes": [

"User"

],

"description": "For writers.",

"displayName": "Writer",

"id": "e0947d03-8281-4e91-b8c4-0b8a57e8211d",

"isEnabled": true,

"value": "Writer"

},

{

"allowedMemberTypes": [

"User"

],

"description": "For administrators.",

"displayName": "Administrators",

"id": "788389c8-0b68-4e90-b903-ea9add7ccd72",

"isEnabled": true,

"value": "Administrators"

}

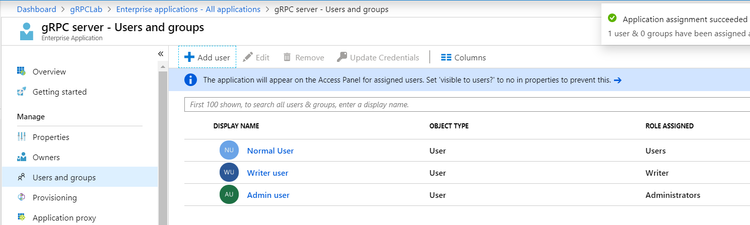
],

And select a "Save" button.

[Spoiler](https://techcommunity.microsoft.com/blog/modernworkappconsult/asp-net-core-grpc-and-wpf-on-net-core-with-azure-ad-authorizationauthentication/837781#community-837781-)

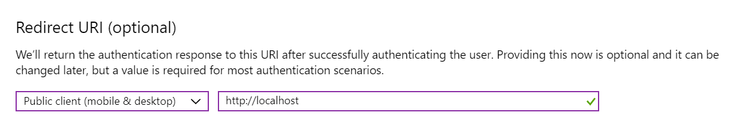
After that, assign roles to user using Azure AD enterprise applications setting page. Select "Azure Active Directory" -> "Enterprise applications" -> "Your server side app" -> "Users and groups".

In my case, I assigned three users like following:

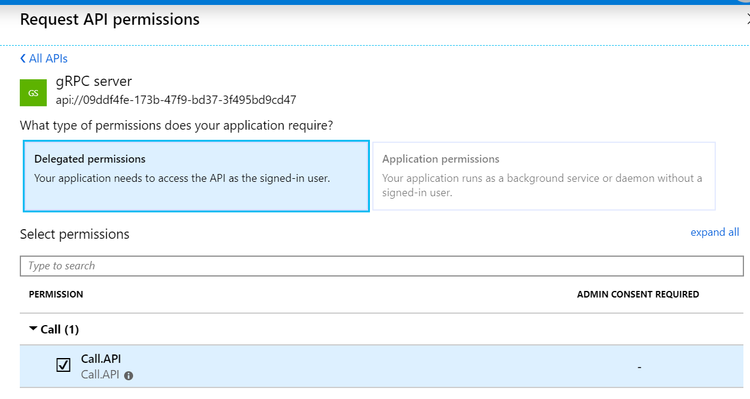


**Create an app for client side**

Add an app for client side with Redirect URI that category and value are "Public Client (mobile & desktop)" and "http://localhost".



After create the app, select "API permissions" -> "Add a permission". And select "Call.API" permission of "gRPC server" app on "APIs my organization uses" tab.



And copy Application ID of client app and Directory(tenant) ID at Overview page of the app.

**Implement Authentication and Authorization to the server side**

Add latest preview version's "Microsoft.AspNetCore.Authentication.JwtBearer" NuGet package to use authentication and authorization features using JWT of ASP.NET Core 3.0.

And add a few lines to Startup.cs to enable Authentication and Authorization like below:

public void ConfigureServices(IServiceCollection services)

{

services.AddGrpc();

// for auth

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

\_configuration.Bind("AzureAd", options);

});

services.AddAuthorization(options =>

{

// define policies

options.AddPolicy("Users", policy => policy.RequireClaim("http://schemas.microsoft.com/ws/2008/06/identity/claims/role", "Administrators", "Writer", "Users"));

options.AddPolicy("Writer", policy => policy.RequireClaim("http://schemas.microsoft.com/ws/2008/06/identity/claims/role", "Administrators", "Writer"));

options.AddPolicy("Admins", policy => policy.RequireClaim("http://schemas.microsoft.com/ws/2008/06/identity/claims/role", "Administrators"));

});

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

// for auth

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

// endpoints.MapGrpcService();

endpoints.MapGrpcService();

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Communication with gRPC endpoints must be made through a gRPC client. To learn how to create a client, visit: https://go.microsoft.com/fwlink/?linkid=2086909");

});

});

}

Add couple settings to appsettings.json for options of AddAuthentication method.

Its values are Authority(URI that included Azure AD Tenant ID) and Audience(Application ID URI) like following:

{

"Logging": {

"LogLevel": {

"Default": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*",

"Kestrel": {

"EndpointDefaults": {

"Protocols": "Http2"

}

},

"AzureAd": {

"Authority": "https://login.microsoftonline.com/{Your Azure AD Tenant ID}/",

"Audience": "Your server side Application ID URI(api://appid)"

}

}

If you want to know more details about those methods, please read following documents:

* [Azure Active Directory with ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/azure-active-directory/?view=aspnetcore-3.0)
* [Claims-based authorization in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-3.0)

So, let's add AuthrizeAttribute to the gRPC service class for adding authorization.

public class EmployeeService : GrpcSampleApp.Employees.EmployeesBase

{

private static List Employees { get; } = new List();

[Authorize("Users")]

public override Task GetEmployees(Empty request, ServerCallContext context)

{

// snip...

}

[Authorize("Writer")]

public override Task AddEmployee(Employee request, ServerCallContext context)

{

// snip...

}

[Authorize("Admins")]

public override Task DeleteEmployee(DeleteEmployeeRequest request, ServerCallContext context)

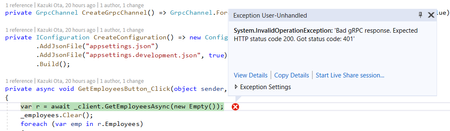
{

// snip...

}

}

After that, the client app will not able to call those APIs with 401 error.



**Implement sign in feature to the client side app**

Add MSAL.NET that package name is Microsoft.Identity.Client to the project of client side to implement sign in feature. And Microsoft.Extensions.Configuration.Binder(latest preview version) package to use Bind method of IConfiguration interface.

At first, we have to create an instance of PublicClientApplication. To create it, we have to prepare three information that are Client ID(The client side app id registered to Azure AD), RedirectUri(http://localhost) and Tenant ID. I write those values to appsettings.json at the client side project. And also, add a value of passing to as scopes when sign in, it is "api://you server app id/Call.API"(It was defined to the server side app on Azure AD.)

{

"ServerEndpoint": "https://localhost:5001",

"AzureAd": {

"ClientId": "Azure AD client app id.",

"RedirectUri": "http://localhost",

"TenantId": "Your Azure AD tenant id."

},

"Scopes": [

"api://your server app id/Call.API"

]

}

To create an instance of PublicClientApplication class that provided from MSAL.NET, I create CreatePublicClientApplication method to MainWindow.xaml.cs.

private IPublicClientApplication CreatePublicClientApplication()

{

var options = new PublicClientApplicationOptions();

\_configuration.Bind("AzureAd", options);

return PublicClientApplicationBuilder.CreateWithApplicationOptions(options)

.Build();

}

The instance is stored a \_publicClientApplication field of MainWindow class. Using it, implements a click event handler of "Sign In" button! There are important below methods of MSAL.NET. Please check [the documentation](https://docs.microsoft.com/bs-latn-ba/azure/active-directory/develop/msal-net-acquire-token-silently) to understand how to use those methods.

* GetAccountsAsync
* AcquireTokenSilent
* AcquireTokenInteractive

Following code snippet is a typical implementation to get access token to call APIs protected by Azure AD.

private async Task<AuthenticationResult> AcquireTokenAsync()

{

var scopes = \_configuration.GetSection("Scopes").Get<string[]>();

var account = (await \_publicClientApplication.GetAccountsAsync()).FirstOrDefault();

try

{

return await \_publicClientApplication.AcquireTokenSilent(scopes, account)

.ExecuteAsync();

}

catch (MsalUiRequiredException ex)

{

Debug.WriteLine(ex);

try

{

using (var cancellationTokenSource = new CancellationTokenSource())

{

var signInTask = \_publicClientApplication.AcquireTokenInteractive(scopes)

.WithSystemWebViewOptions(new SystemWebViewOptions

{

OpenBrowserAsync = SystemWebViewOptions.OpenWithChromeEdgeBrowserAsync,

})

.ExecuteAsync(cancellationTokenSource.Token);

if (signInTask != await Task.WhenAny(signInTask, Task.Delay(TimeSpan.FromMinutes(2))))

{

cancellationTokenSource.Cancel();

MessageBox.Show("Timeout.");

return null;

}

return await signInTask;

}

}

catch (MsalException msalEx)

{

Debug.WriteLine(msalEx);

return null;

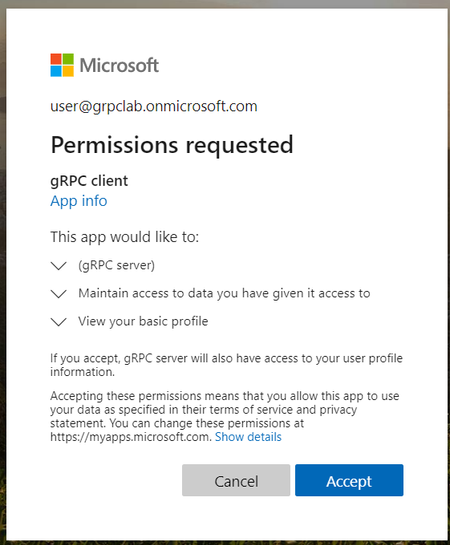
}

}

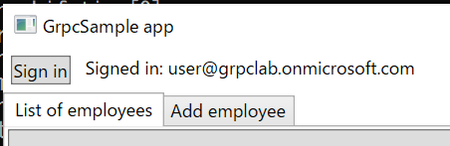
}

[Spoiler](https://techcommunity.microsoft.com/blog/modernworkappconsult/asp-net-core-grpc-and-wpf-on-net-core-with-azure-ad-authorizationauthentication/837781#community-837781-)

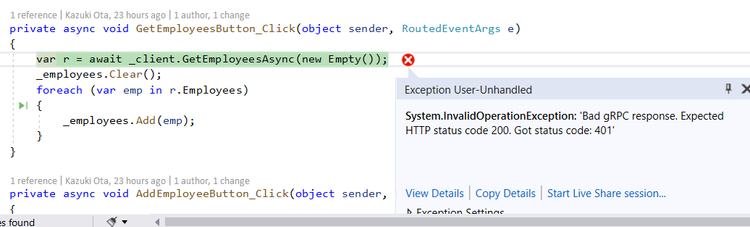
Run the app! So, the "Sign in" button is work.

Accepting permissions page

After signed in, the user name is shown.



We can get sign in feature, however we can't call APIs with 401 error yet.



Need Authorization HTTP header, it can be add using a metadata argument of every gRPC service methods.

Define a method to create a Metadata instance:

private async Task<Metadata> CreateAuthMetadataAsync()

{

var r = await AcquireTokenAsync();

if (r == null)

{

return null;

}

return new Metadata

{

{ "Authorization", $"Bearer {r.AccessToken}" },

};

}

And add code to use it before call APIs like below:

private async void GetEmployeesButton\_Click(object sender, RoutedEventArgs e)

{

// Get and check accessToken

var headers = await CreateAuthMetadataAsync();

if (headers == null)

{

MessageBox.Show("Couldn't get accessToken.");

return;

}

var r = await \_client.GetEmployeesAsync(new Empty(), headers); // add headers argument

\_employees.Clear();

foreach (var emp in r.Employees)

{

\_employees.Add(emp);

}

}

Add same code snippet to all location where call APIs.

After that, be able to call gRPC services from client app with access token. Of couse, not allowed users can't call APIs. For example: Users role user can't call API for Admins user.

Final step!! Add check role in client side feature. It is really easy with System.IdentityModel.Token.Jwt NuGet package.

Add the package to the client side app, it is really easy to use, just pass JWT access token to constructor of JwtSecurityToken class. After creating an instance, access claims by Claims property, like following:

var claims = new JwtSecurityToken(r.AccessToken);

var roleName = claims.Claims.FirstOrDefault(x => x.Type == "roles")?.Value ?? "Users"; // default is users role

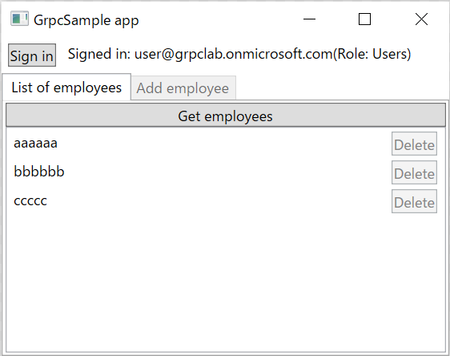
textBlockSiginStatus.Text = $"Signed in: {r.Account.Username}(Role: {roleName})";

tabItemAddEmployee.IsEnabled = roleName == "Writer" || roleName == "Administrators";

IsInAdminsRole = roleName == "Administrators";

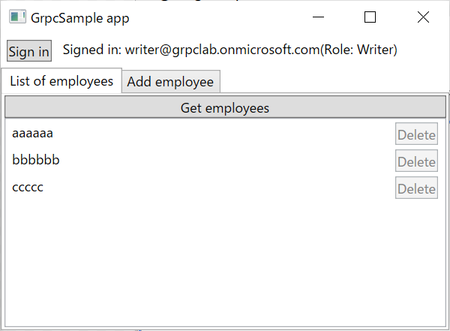
In this case, There is a role name at "roles" of type of Claim. In client side, we can control visible/colappse, enabled/disabled, etc...

Following picture is sign in as Users role:



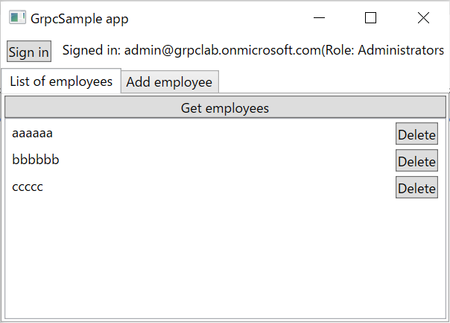
Disable a "Add employee" tab and "Delete" buttons.

Sign in as Writer role:



Enable a "Add employee" tab, however "Delete" buttons are disabled.

Sign in as Administrators role:



All features are available!