**Autenticação via Azure AD utilizando a biblioteca “@azure/msal-angular” versão 2.5.5**

O primeiro passo para poder desenvolver uma autenticação via Azure AD é fazer a configuração do Azure no portal Azure, abaixo vou colocar as configurações que utilizei para funcionar.

Acessar <https://portal.azure.com/>

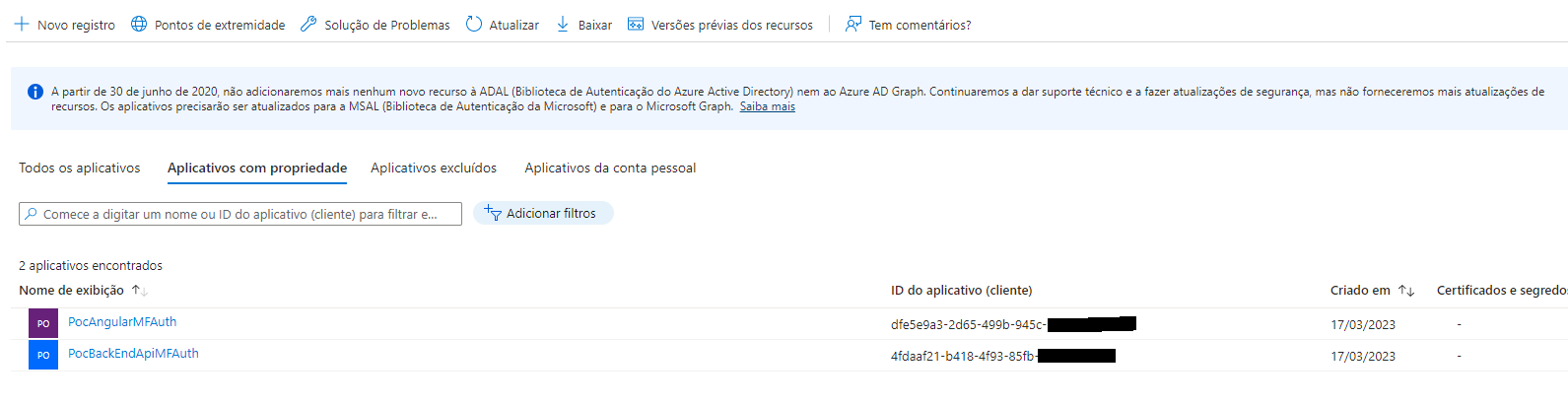
Ao acessar o portal do Azure, acessar o menu “Azure Active Directory”



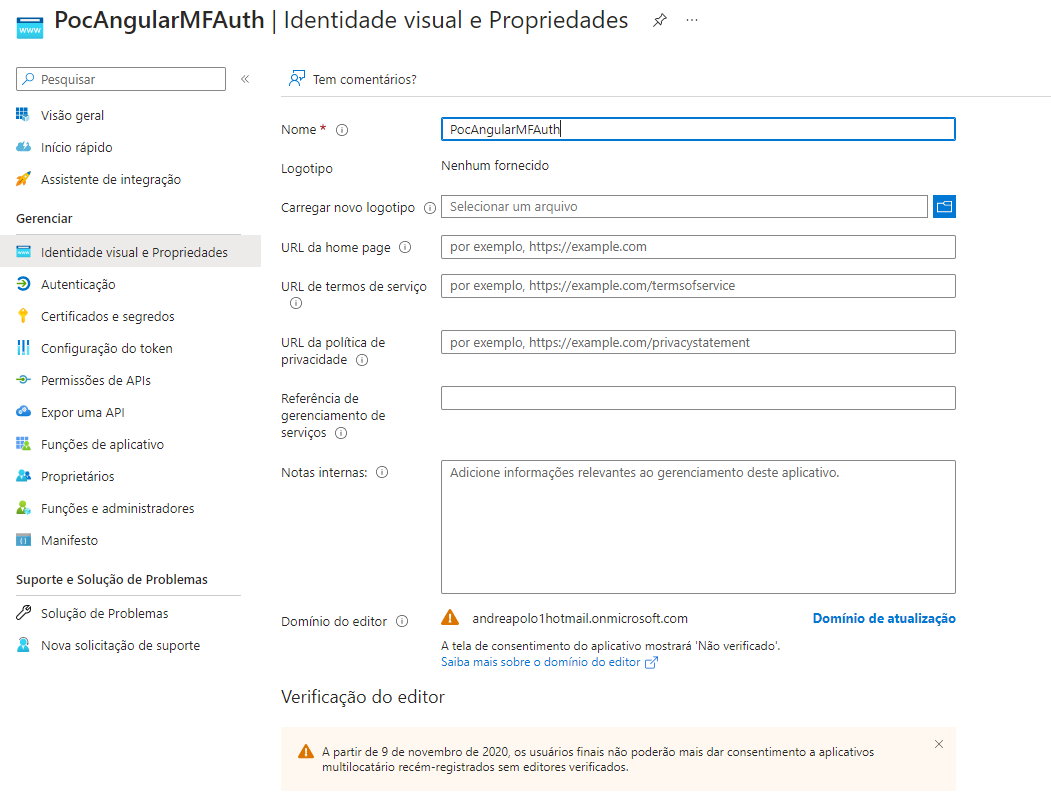
Entrar no menu “Registros de Aplicativo”



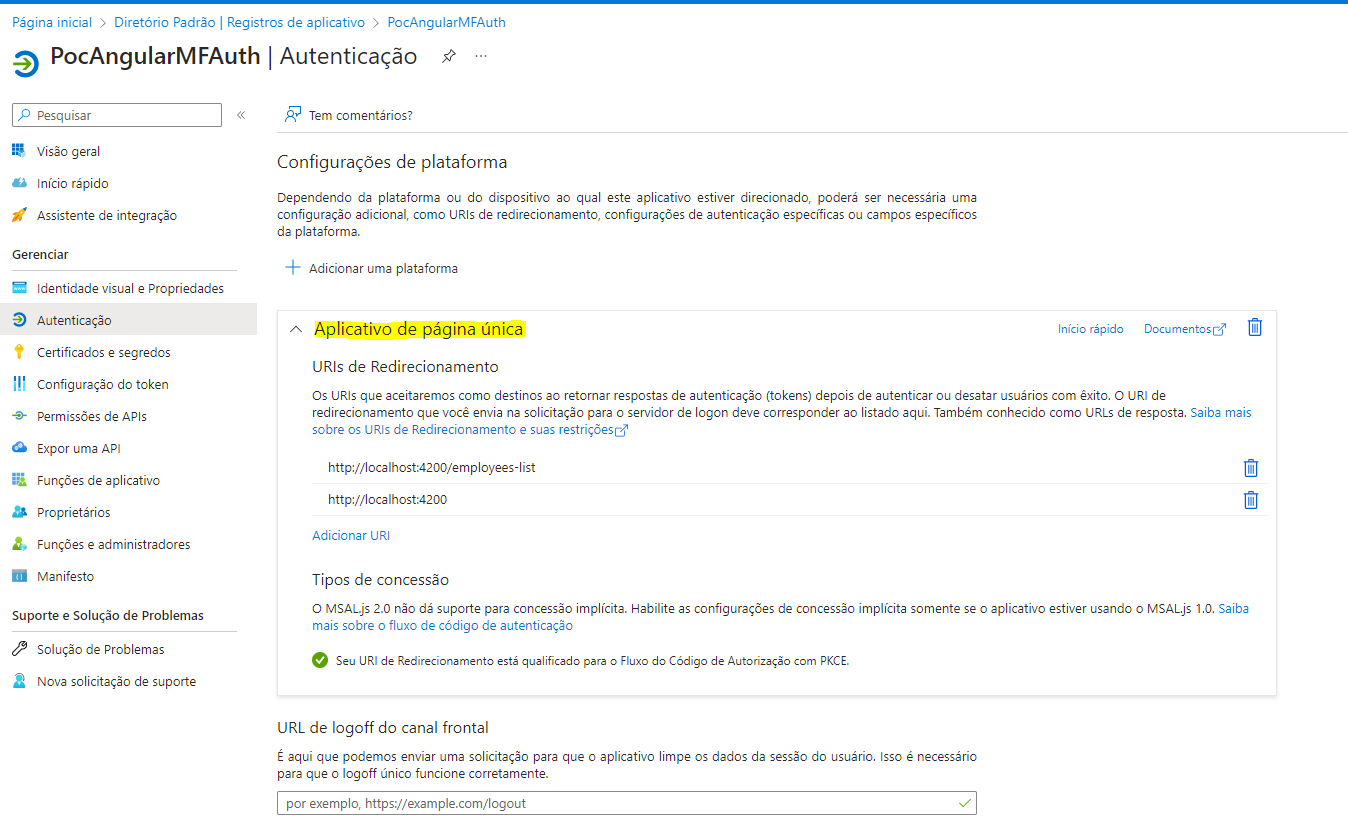
Clicar em “novo registro” e cadastrar a aplicação angular conforme na tela abaixo.

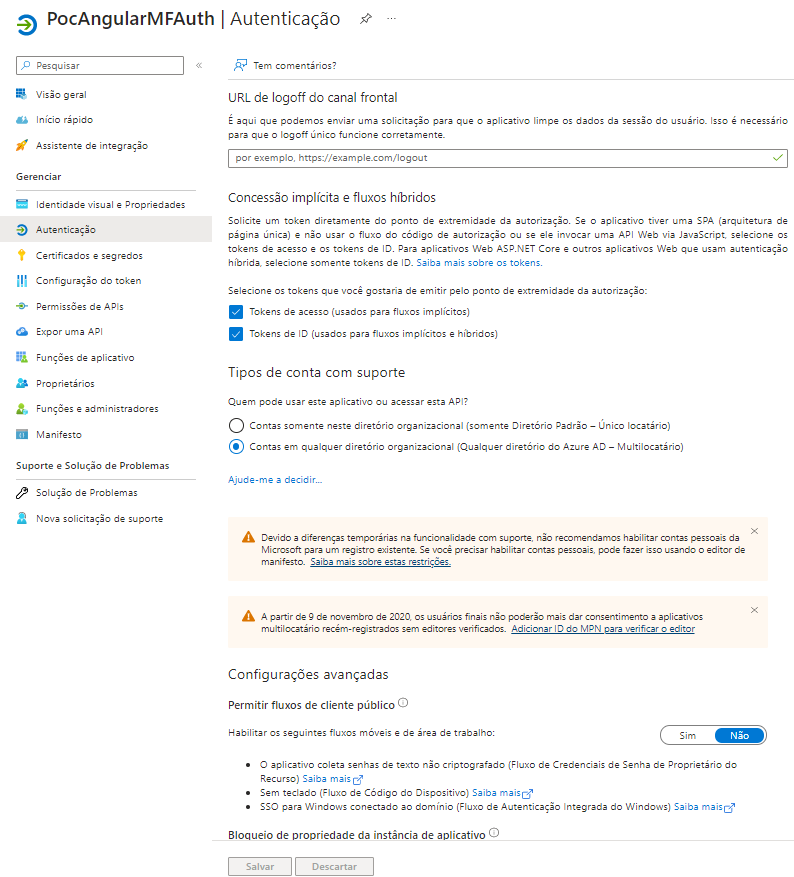


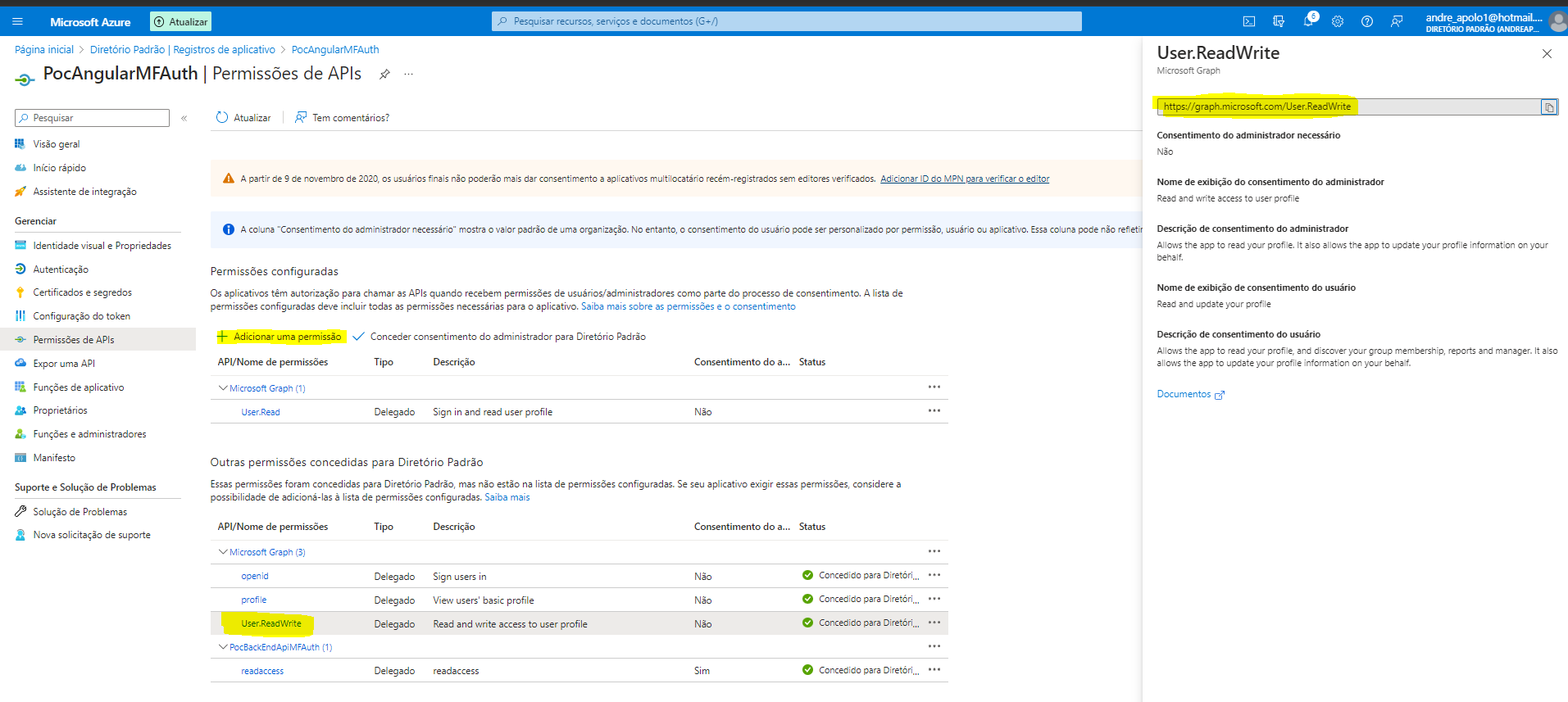
No caso do angular com a biblioteca @azure/msal-azure versão 2.5.5 a configuração deve ser feita conforme o print abaixo, colocando como SPA (Single Page Application)



Cadastrar as URLs da aplicação Angular que o Azure pode fazer redirecionamento.





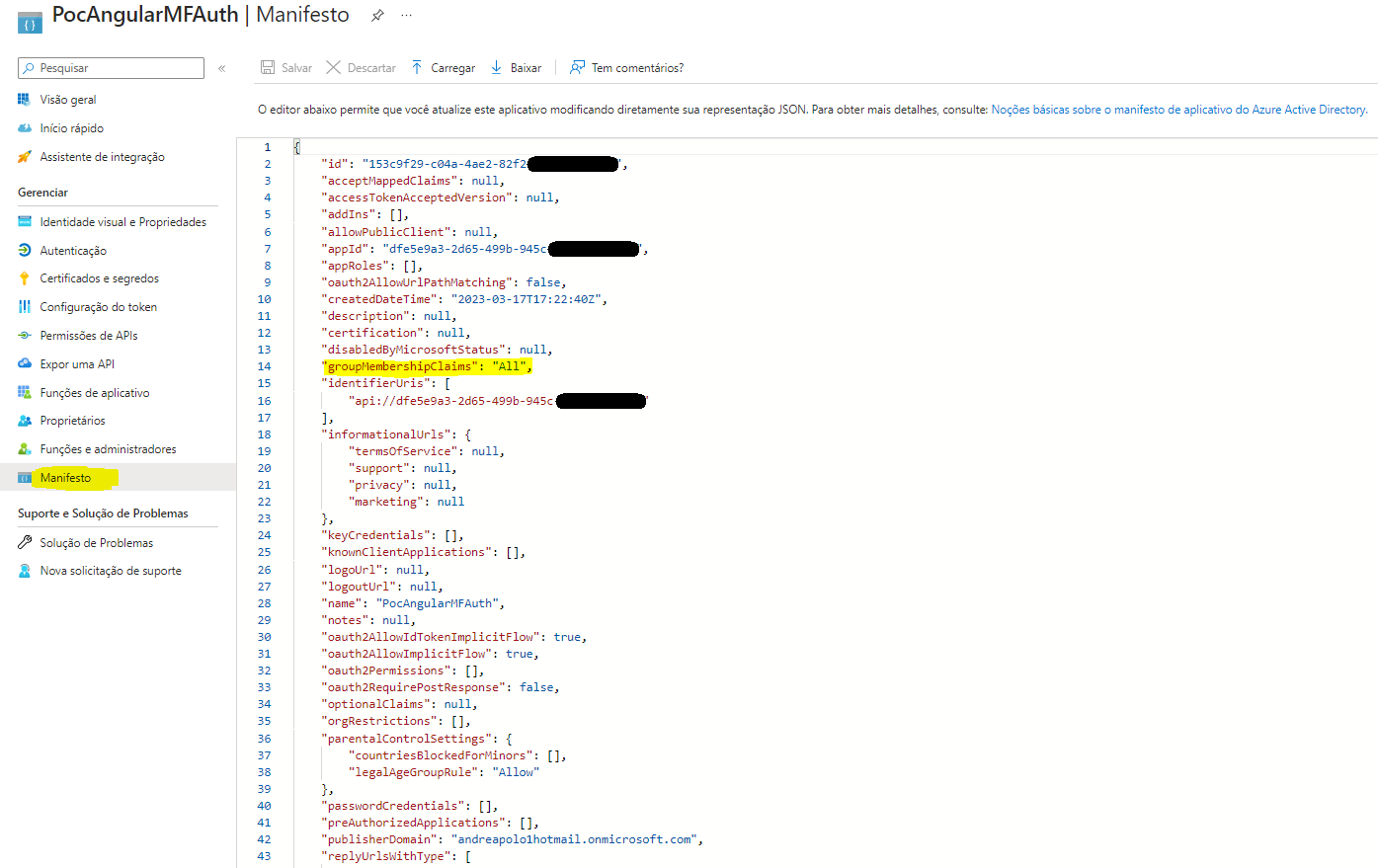


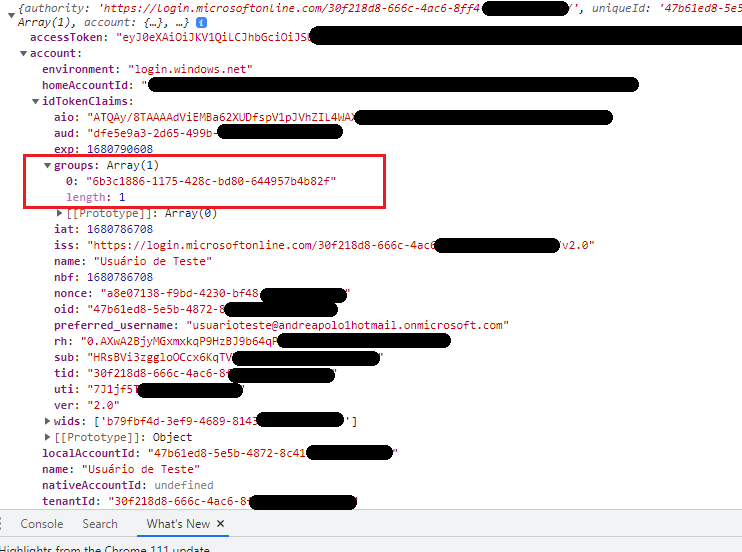
Em Manifesto, configurar a propriedade “groupMembershipClaims” com o valor “All”, isto também será necessário fazer na configuração dao API Backend para que o token gerado para o front-end seja também válido para o backend.

Os valores possíveis para esta propriedade são as seguintes:

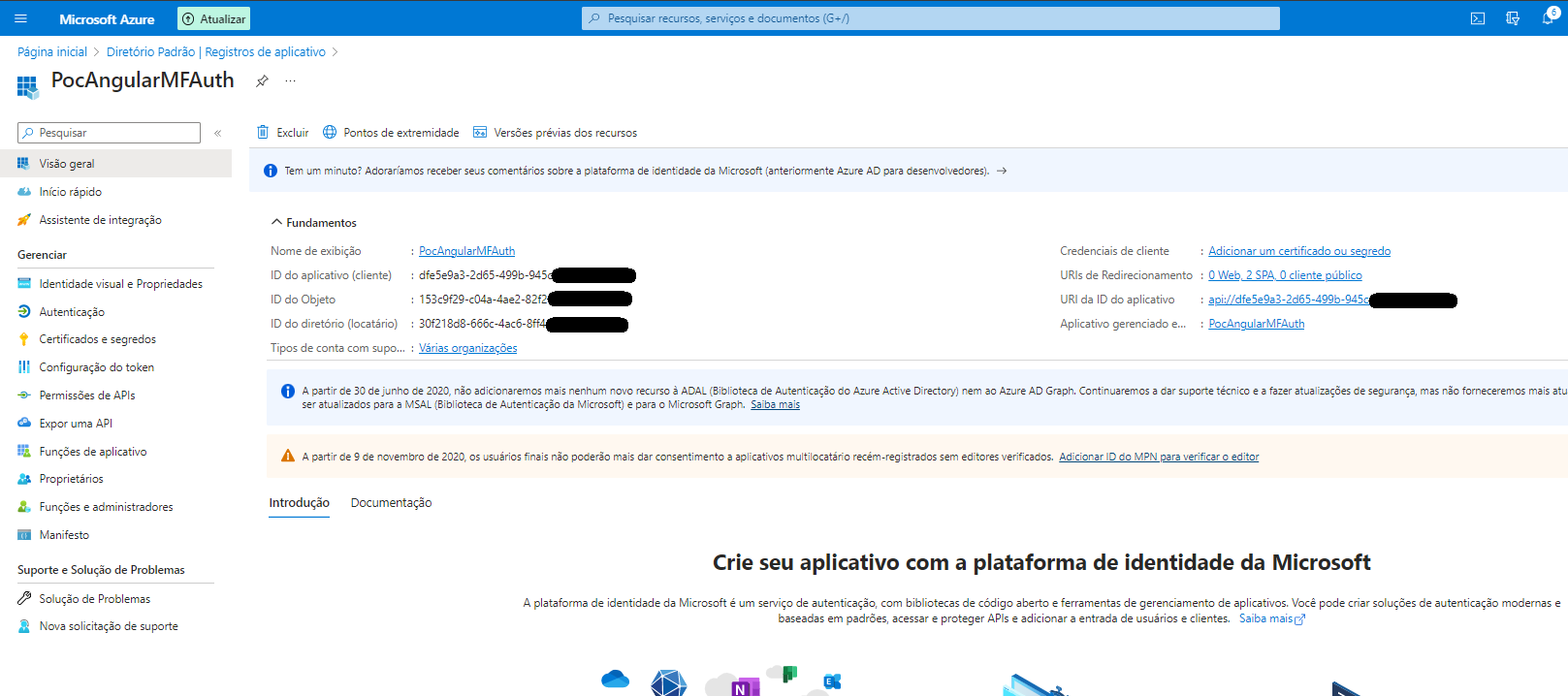
|  |
| --- |
| The groupMembershipClaims property of the [application manifest](https://learn.microsoft.com/en-us/azure/active-directory/develop/reference-app-manifest) configures the groups claim on a per-application basis. A value of null excludes all groups, a value of SecurityGroup includes only Active Directory Security Group memberships, and a value of All includes both Security Groups and Microsoft 365 Distribution Lists. |

Fonte: <https://learn.microsoft.com/en-us/azure/active-directory/develop/access-tokens#claims-in-access-tokens>

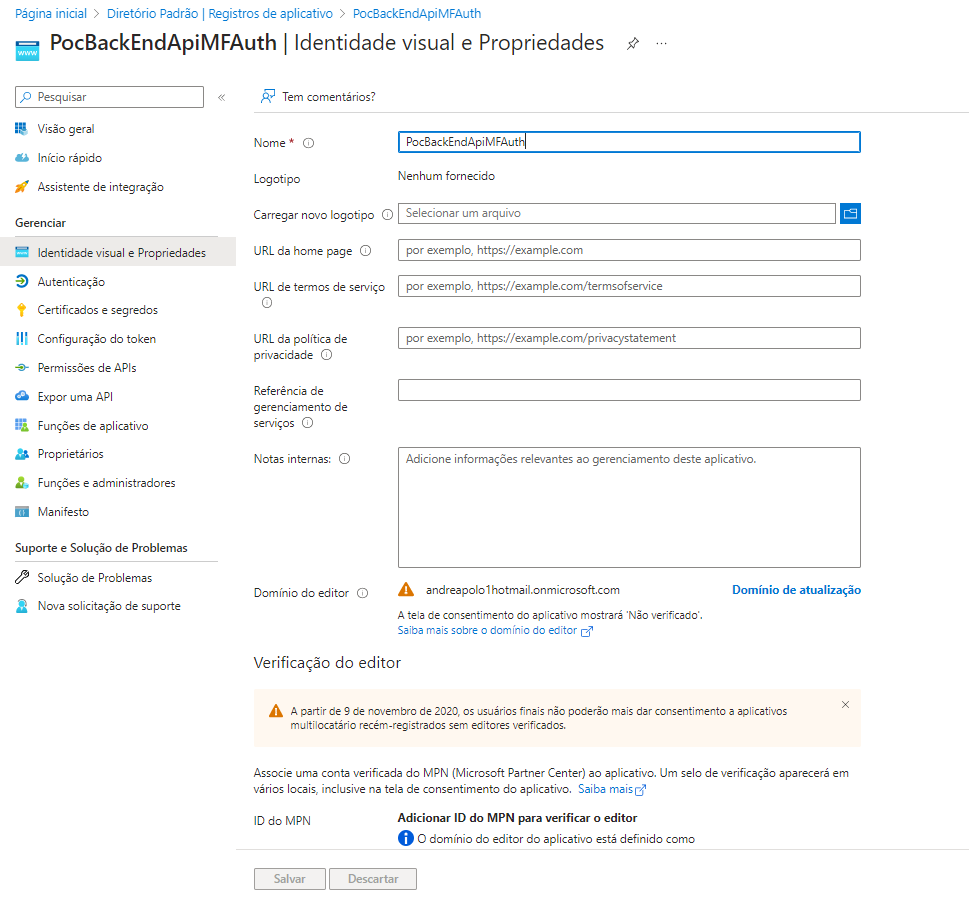


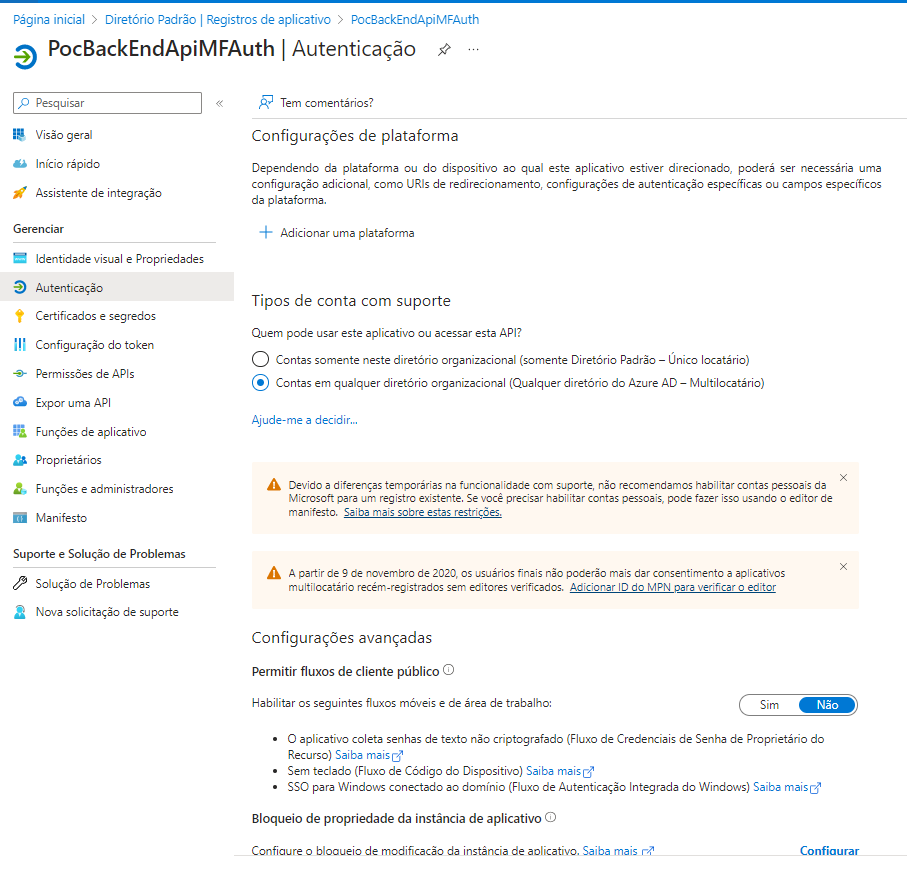
Desta forma o “AuthenticationResult” retornado para o angular irá conter o ID dos grupos que o usuário possui permissão, porém até onde eu pesquisei o Azure retorna o ID e não o nome do grupo, existe também uma outra rota que pode buscar os grupos que o usuário possui, [https://graph.microsoft.com/v1.0/users/{userId}/getMemberObjects](https://graph.microsoft.com/v1.0/users/%7buserId%7d/getMemberObjects) porém não fiz teste/uso da mesma para saber qual seria o retorno desta.

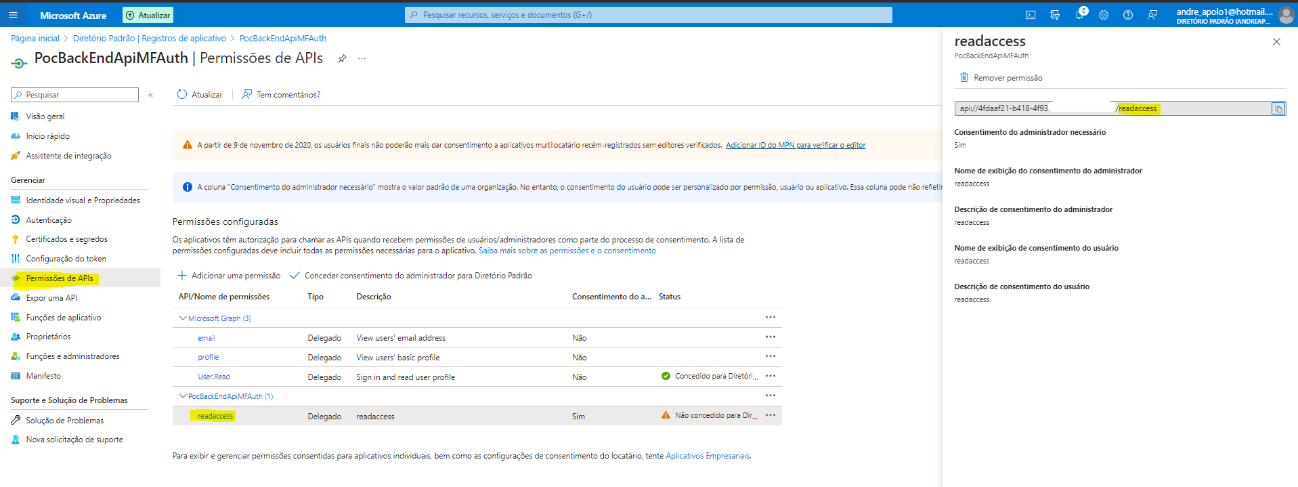
Na parte de “Visão Geral” podemos ver os ids gerados, precisaremos utilizar estes Ids na aplicação para que tudo funcione corretamente, precisaremos do “Client ID” (id do aplicativo (cliente)) e também do TenentId (ID do diretório do locatário).



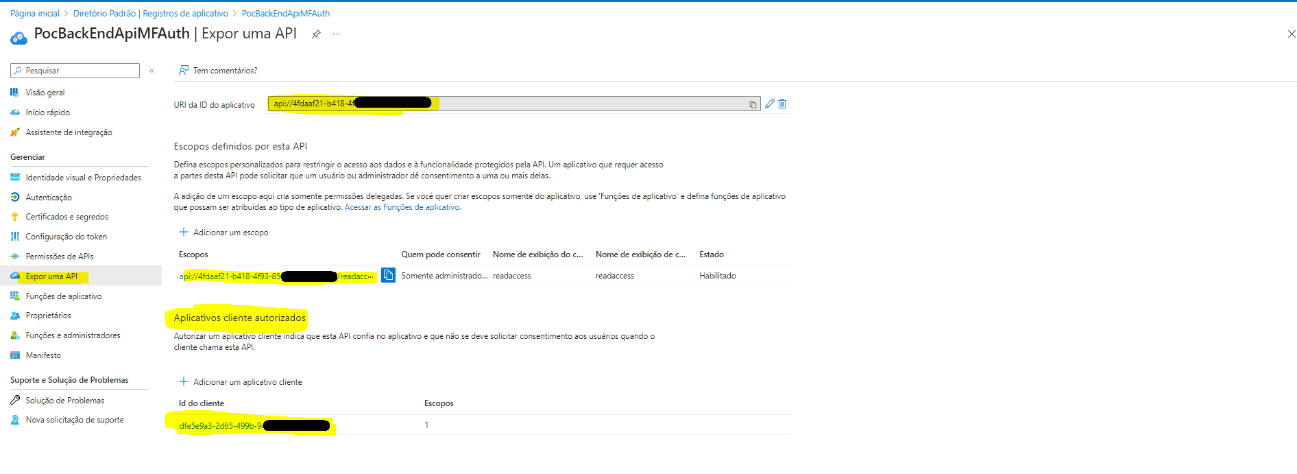
Agora devemos fazer também a configuração da API do backend:



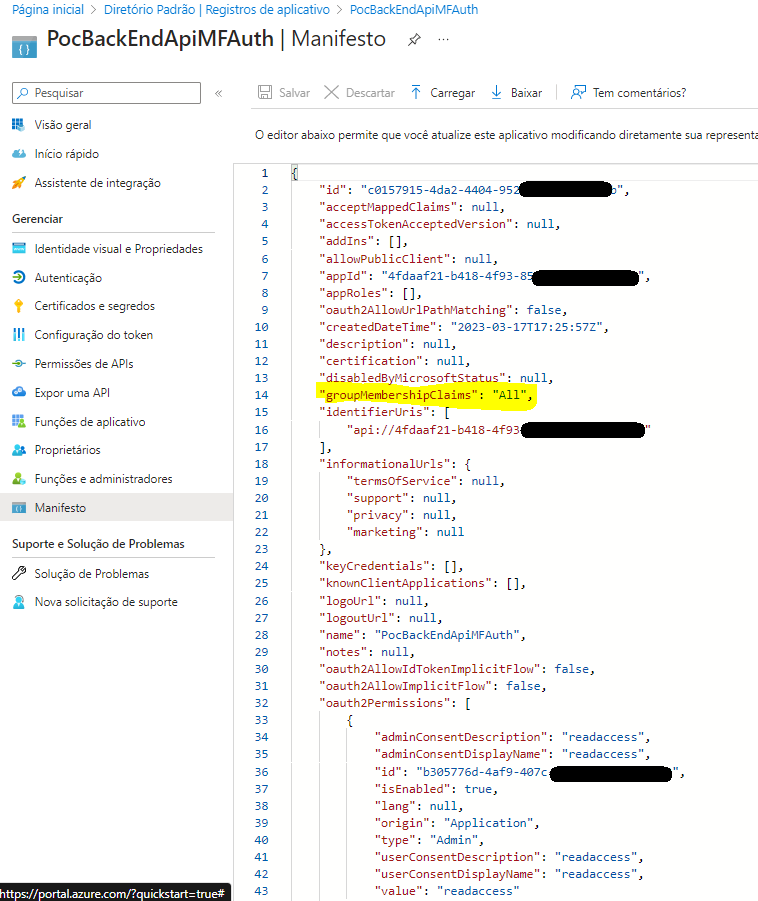




Este próximo passo é importante, devemos expor a API do backend, adicionando o escopo “readaccess” que criamos no passo anterior e adicionando acesso “permissão” para o aplicativo front-end nesta API, note que no final deste print a sessão “id do cliente” é o (ClientId) da aplicação Angular (front-end)



Configurar a propriedade “groupMembershipClaims” também com o valor “All”tal como foi feito no manifest da aplicação Angular.



**Após as devidas configurações realizadas no Portal Azure**

- Instalar no projeto o pacote @azure/msal-angular

- Intalar no projeto o pacote @azure/msal-browser

OBS: Caso a instalação não funcione com o comando npm i @azure/msal-angular , utilizar o comando @azure/msal-angular --force

O projeto de exemplo front-end encontra-se no git no endereço:

Verificar os códigos que encontram-se em: <https://github.com/githubandreluz/MFAuthAzureAD-Frontend-LoginPopup.git>

app.module.ts

auth-interceptor.service.ts

msaluser.service.ts

app.component.ts

app-routing.module.ts

index.html

O projeto de exemplo do back-end encontra-se no git no endereço: <https://github.com/githubandreluz/MFAuthAzureAD-APIBackend.git>

Verificar os códigos que encontram-se em:

Program.cs

appsettings.json

|  |
| --- |
| // Program.cs  using Microsoft.Identity.Web;  var builder = WebApplication.CreateBuilder(args);  ConfigurationManager configuration = builder.Configuration;  // Add services to the container.  builder.Services.AddCors(options => options.AddPolicy("AllowAll", builder => { builder.WithOrigins("\*", "https://localhost", "http://localhost", "http://homologacao.gsw.com.br").AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin(); }));  builder.Services.AddControllers()  .AddJsonOptions(options => options.JsonSerializerOptions.PropertyNameCaseInsensitive = true);  //Este comando já esta "descontinuado" portanto deixei aqui comentado, utilizar o "AddMicrosoftIdentityWebApiAuthentication" no lugar  //builder.Services.AddAuthentication(AzureADDefaults.BearerAuthenticationScheme).AddAzureADBearer(options => configuration.Bind("AzureActiveDirectory", options));  // Sign-in users with the Microsoft identity platform  builder.Services.AddMicrosoftIdentityWebApiAuthentication(configuration, "AzureActiveDirectory");  // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle  builder.Services.AddEndpointsApiExplorer();  builder.Services.AddSwaggerGen();  var app = builder.Build();  // Configure the HTTP request pipeline.  if (app.Environment.IsDevelopment())  {  app.UseDeveloperExceptionPage();  app.UseSwagger();  app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "POC MULTI FACTOR AUTHENTICATION V1"));  }  app.UseRouting();  app.UseCors(x => x  .AllowAnyOrigin()  .AllowAnyMethod()  .AllowAnyHeader());  app.UseCors("AllowAll");  app.UseHttpsRedirection();  app.UseAuthentication();  app.UseRouting();  app.UseAuthorization();  app.UseEndpoints(endpoints =>  {  endpoints.MapControllers();  });  app.Run(); |