

Rafael Cruz



Home About Contato

[Updated] Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API – (ASP.NET Identity + Identity Server v4 + EntityFrameworkCore)

⌚ 15 de novembro de 2019 🙏 rbento 💬 2

Fala Galera,

Hoje venho neste post fazer atualização de uns dos posts mais lido do meu blog. O post no qual me refiro é este aqui [Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API](#).

Por isso, resolvi atualizá-lo já que ele foi feito em **.NET Framework** porém não vai ser uma simples atualização, vamos melhorar um pouco, além de usar o **.NET Core**, vamos também utilizar o **Identity Server v4 integrado com ASP.NET Identity** para nosso gestor de identidade.

Meu muito obrigado a todos que acompanham meu blog e que leem meus artigos.

O que é JSON Web Token (JWT) ?

Json Web Token é um **Token** de segurança que atua como container de informações sobre um determinado usuário. Ele pode ser transmitido facilmente entre o Servidor de Autorização (Token Issuer) e o Servidor de Recursos (Web Api).

Basicamente um **JSON Web Token** é uma cadeia de strings que é formado por 3 partes separadas por ponto (.): **Header**, **Payload**, **Signature**

A parte do **Header** é formado por um objeto **JSON** com duas propriedades são elas:

- **tip**: sempre tem o valor **JWT**
- **alg**: determina qual o algoritmo que foi usado para assinar o **Token**

A parte do **Payload** é um objeto **JSON** que contém informações do usuário e quais são seus perfis, veja o exemplo abaixo

```

1. {
2.     "unique_name": "rcruz",
3.     "sub": "rcruz",
4.     "role": [
5.         "Administrator"
6.     ],
7.     "iss": "http://mytokenissuer.com.br",
8.     "aud": "379ee8430c2d421380a713458c23ef74",
9.     "exp": 14142345602,
10.    "nbf": 1434241802
11. }

```

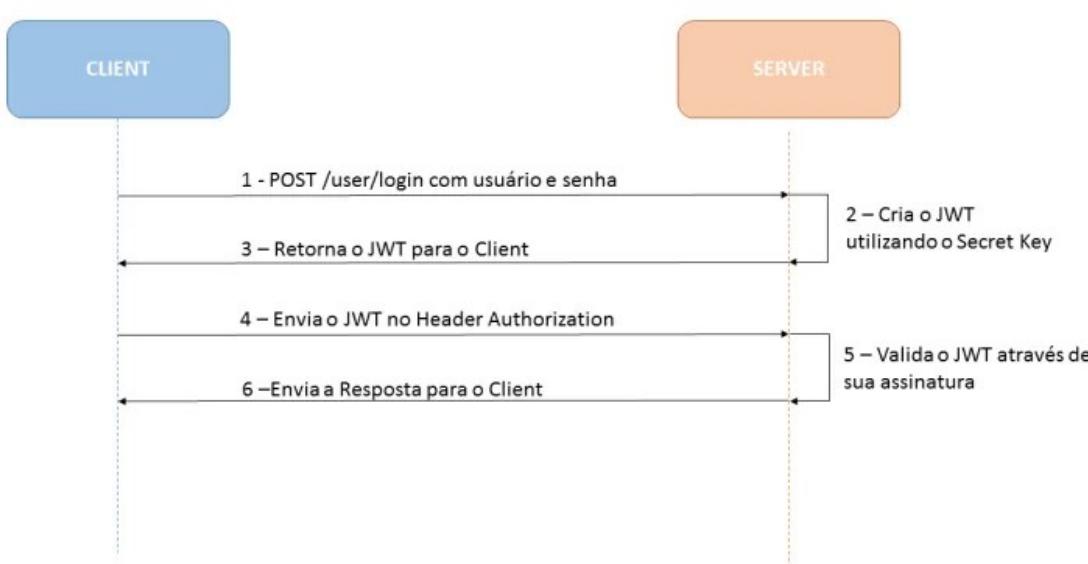
Nem todas as propriedades são obrigatórias porém no nosso exemplo vamos utilizar as propriedades acima, veja o que cada uma representa:

- **sub (subject)**: Representa o nome de usuário para qual o **Token** foi expedido
- **role** : Representa em quais perfis o usuário se encontra
- **iss (Token Issuer)**: Representa o servidor de autenticação que gerou o **Token**
- **aud (Audience)**: Representa o servidor de destino no qual este Token será usado
- **exp (Expiration)**: Representa o tempo de expiração do **Token** em formato **UNIX**
- **nbf (Not Before)**: Representa o tempo em formato **UNIX** que este **Token** não deve ser usado

A última parte do **Token** é uma junção do **Header** e do **Payload** em base 64 utilizando o algoritmo de criptografia definido no **Header** para gerar a assinatura do **Token** no nosso caso vamos utilizar o **HMAC-SHA256**.

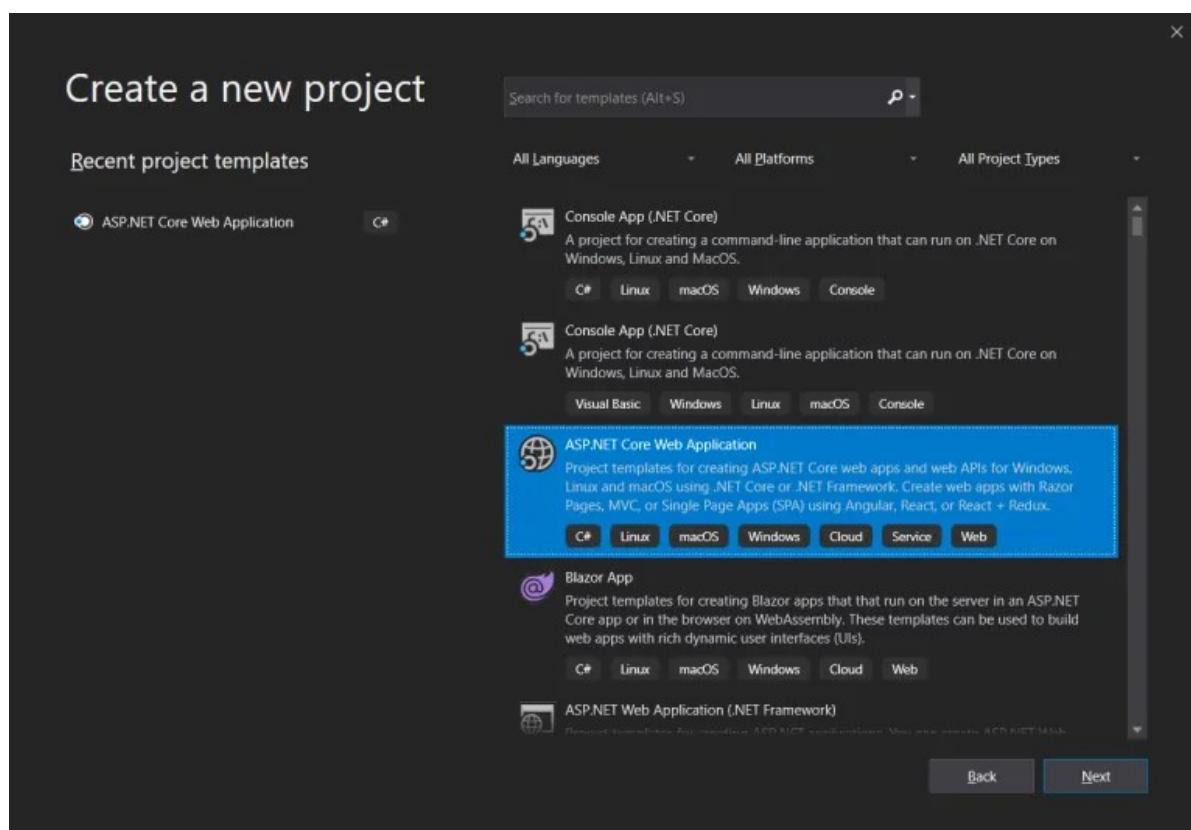
O Fluxo de Autenticação do JWT

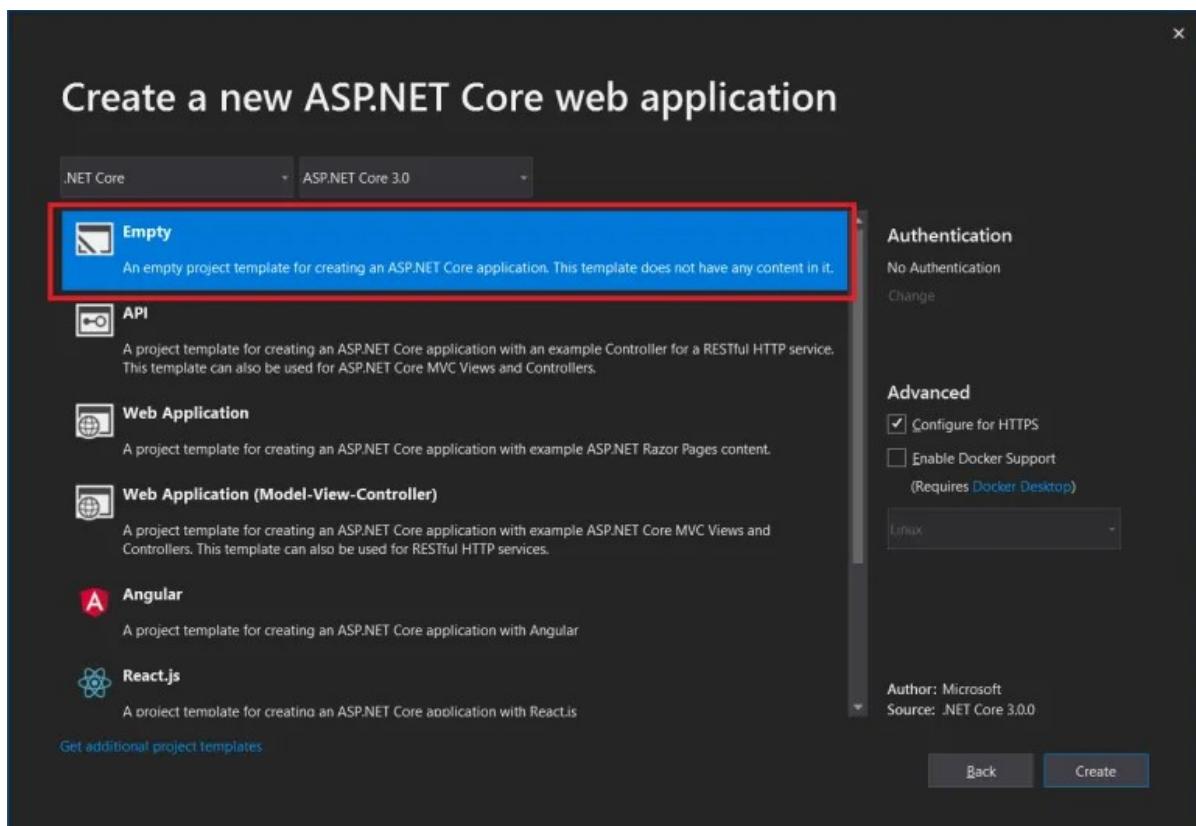
No fluxo abaixo detalha como é a criação e autenticação do **JWT**. Entendendo esse fluxo torna-se muito mais fácil utilizá-lo em nossas aplicações



Utilizando o Identity Server v4 como Gestor de Identidade

Para começar vamos criar o projeto do gestor de identidade utilizando o **Identity Server v4**. Crie o projeto conforme imagens abaixo:





Com o nosso projeto criado, vamos adicionar os pacotes necessários para fazer nosso gestor de identidade. Adicione os pacotes abaixo no projeto:

- **IdentityServer4**
- **IdentityServer4.AspNetIdentity**
- **IdentityServer4.EntityFramework**
- **Microsoft.AspNetCore.Identity**
- **Microsoft.AspNetCore.Identity.EntityFrameworkCore**
- **Microsoft.EntityFrameworkCore**
- **Microsoft.EntityFrameworkCore.SqlServer**

Estamos neste ponto preparados para configurar o **Identity Server v4**

Configurando o ASP.NET Identity com o Entity Framework

Neste passo vamos configurar o **ASP.NET Identity** integrado com o **Entity Framework**, o **ASP.NET Identity** será responsável pela gerência de usuário de sua aplicação.

Vamos criar uma nova classe chamada **ApplicationUser** e coloca-lá na raiz do nosso projeto.

A classe **ApplicationUser** deve ter o código conforme exemplo abaixo:

```
1. public class ApplicationUser : IdentityUser  
2. {
```

```
3.  
4. }
```

Com a classe **ApplicationUser** criada, vamos criar a classe que será responsável por gerir o banco do **ASP.NET Identity** . Crie uma classe chamada **ApplicationDbContext** e coloque na raiz do projeto.

A classe **ApplicationDbContext** deve ter o código conforme exemplo abaixo:

```
1. public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  
2. {  
3.     public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)  
4.         : base(options)  
5.     {  
6.     }  
7. }  
8.  
9. protected override void OnModelCreating(ModelBuilder builder)  
10. {  
11.     base.OnModelCreating(builder);  
12. }  
13. }
```

Agora que as classes estão criadas, vamos configura-las. Abra o **Startup.cs** e coloque o código conforme exemplo abaixo:

```
1. public void ConfigureServices(IServiceCollection services)  
2. {  
3.     services.AddDbContext<ApplicationDbContext>(options =>  
4.     options.UseSqlServer(Configuration.GetConnectionString("IdentityConnection")));  
5.  
6.     services.AddIdentity<ApplicationUser, IdentityRole>()  
7.         .AddEntityFrameworkStores<ApplicationDbContext>()  
8.         .AddDefaultTokenProviders();  
9.  
10. }  
11. }
```

Agora nossa aplicação está preparada para utilizar o **ASP.NET Identity** com o **Entity Framework** . Se você reparou estamos utilizando uma conexão com o banco de dados **SQL SERVER** e precisamos configurar o **appsettings.json** adicionando a chave de conexão com o banco de dados.

Abra o **appsettings.json** e adicione a seguinte estrutura conforme exemplo abaixo:

```
1. {  
2.     "ConnectionStrings": {  
3.         "IdentityConnection": "<SUAS CHAVES DE CONEXÃO COM O BANCO DE DADOS>"  
4.     },
```

```
5.  
6.    "Logging": {  
7.        "LogLevel": {  
8.            "Default": "Information",  
9.            "Microsoft": "Warning",  
10.           "Microsoft.Hosting.Lifetime": "Information"  
11.        }  
12.    },  
13.    "AllowedHosts": "*"  
14. }
```

Integrando o Identity Server v4 com o ASP.NET Identity

Com a integração do **ASP.NET Identity** concluída devemos integrar o **Identity Server v4** no projeto. Abra o **Startup.cs** e adicione as configurações necessárias para o **Identity Server v4** trabalhar junto com o **ASP.NET Identity**.

Adicione as linhas conforme código abaixo:

```
1. public void ConfigureServices(IServiceCollection services)  
2. {  
3.     services.AddDbContext<ApplicationContext>(options =>  
4.         options.UseSqlServer(Configuration.GetConnectionString("IdentityConnection")));  
5.     services.AddIdentity<ApplicationUser, IdentityRole>()  
6.         .AddEntityFrameworkStores<ApplicationContext>()  
7.         .AddDefaultTokenProviders();  
8.     var migrationsAssembly =  
9.     typeof(Startup).GetTypeInfo().Assembly.GetName().Name;  
10.    services.AddIdentityServer()  
11.        .AddDeveloperSigningCredential()  
12.        .AddAspNetIdentity<ApplicationUser>()  
13.        .AddConfigurationStore(options => {  
14.            options.ConfigureDbContext = builder =>  
15.                builder.UseSqlServer(Configuration.GetConnectionString("IdentityConnection"), db  
16.                    => db.MigrationsAssembly(migrationsAssembly)); })  
17.        .AddOperationalStore(options => { options.ConfigureDbContext  
18.            = builder =>  
19.                builder.UseSqlServer(Configuration.GetConnectionString("IdentityConnection"), db  
20.                    => db.MigrationsAssembly(migrationsAssembly)); });
```

Por fim devemos agora ativar o serviço, ative o serviço do **Identity Server v4** conforme código abaixo:

```
1. public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
2. {
3.     if (env.IsDevelopment())
4.     {
5.         app.UseDeveloperExceptionPage();
6.     }
7.
8.     app.UseRouting();
9.
10.    app.UseIdentityServer();
11.
12.    app.UseEndpoints(endpoints =>
13.    {
14.        endpoints.MapGet("/", async context =>
15.        {
16.            await context.Response.WriteAsync("Hello World!");
17.        });
18.    });
19. }
```

Configurando o Identity Server v4

Com a integração concluída devemos configurar os **Clients**, **API Resources**, **Identity Resource** que o **Identity Server v4** irá utilizar.

Vamos criar uma nova classe chamada **IdentityServerConfiguration**, nela estará contida toda a configuração necessária do **Identity Server**.

Crie a classe na raiz do projeto e insira o código abaixo:

```
1. public class IdentityServerConfiguration
2. {
3.     public static IEnumerable<IdentityResource> GetIdentityResources()
4.     {
5.         return new List<IdentityResource>()
6.         {
7.             new IdentityResources.OpenId(),
8.             new IdentityResources.Profile()
9.         };
10.    }
11.
12.    public static IEnumerable<ApiResource> GetApiResources()
13.    {
14.        return new List<ApiResource>()
15.        {
16.            new ApiResource("API", "Api Resources")
17.        };
18.    }
19. }
```

```
18.     }
19.
20.    public static IEnumerable<Client> GetClientScope()
21.    {
22.        return new List<Client>()
23.        {
24.            new Client()
25.            {
26.                ClientId = "79E0C2E2-D750-45BC-8FA3-1A9D5F9F82B5",
27.                ClientName = "Web API Acess Token",
28.                AllowedGrantTypes =
29.                    GrantTypes.ResourceOwnerPasswordAndClientCredentials,
30.                    AllowOfflineAccess = true,
31.                    ClientSecrets =
32.                    {
33.                        new Secret("1234567890".Sha256())
34.                    },
35.                    AllowedScopes =
36.                    {
37.                        IdentityServerConstants.StandardScopes.OpenId,
38.                        IdentityServerConstants.StandardScopes.Profile,
39.                        "API"
40.                    },
41.            }
42.        };
43.    }
44. }
```

Essa classe será usada para configuração do **Identity Server v4** no banco de dados. Vamos criar os **Migrations** necessários para criar a base de dados.

Criando os Migrations necessários do ASP.NET Identity & Identity Server v4

Estamos com tudo pronto para utilizar o **Identity Server v4** integrado com o **ASP.NET Identity**, porém ainda falta um detalhe, criar a base de dados e para isso usaremos os **Migrations** do **EntityFrameworkCore** nesta tarefa.

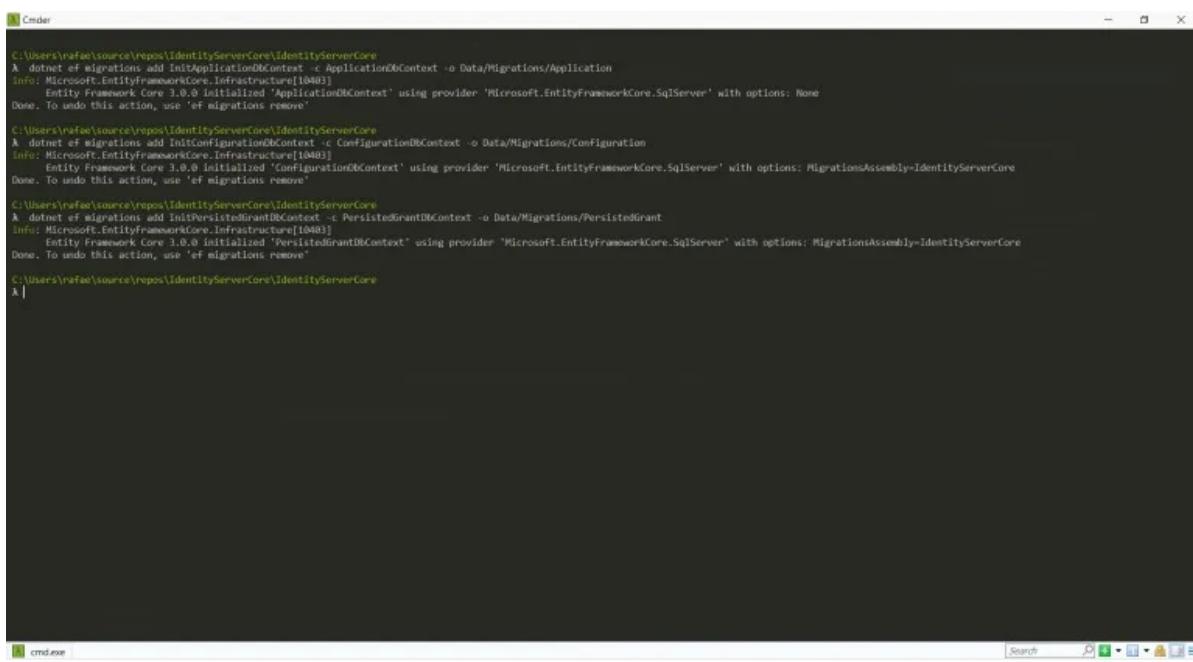
Iremos utilizar literalmente 3 contextos para criar nosso banco, o **ApplicationDbContext** que é do **ASP.NET Identity**, o **ConfigurationDbContext** e o **PersistedGrantDbContext** que são do **Identity Server v4**. Com eles temos todas a estrutura de banco de dados necessária para o **Identity Server v4**.

No prompt de comando digite:

```
■ dotnet ef migrations add InitApplicationDbContext -c ApplicationDbContext -o Data/Migrations/Application
```

- **dotnet ef migrations add InitConfigurationDbContext -c ConfigurationDbContext -o Data/Migrations/Configuration**
- **dotnet ef migrations add InitPersistedGrantDbContext -c PersistedGrantDbContext -o Data/Migrations/PersistedGrant**

Veja a imagem para um exemplo melhor:



```
C:\Users\rafael\source\repos\IdentityServerCore\IdentityServerCore
A: dotnet ef migrations add InitApplicationDbContext -c ApplicationDbContext -o Data/Migrations/Application
Info: Microsoft.EntityFrameworkCore.Infrastructure[1040]
      Entity Framework Core 3.0.0 initialized 'ApplicationDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Done. To undo this action, use 'ef migrations remove'

C:\Users\rafael\source\repos\IdentityServerCore\IdentityServerCore
A: dotnet ef migrations add InitConfigurationDbContext -c ConfigurationDbContext -o Data/Migrations/Configuration
Info: Microsoft.EntityFrameworkCore.Infrastructure[1040]
      Entity Framework Core 3.0.0 initialized 'ConfigurationDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: MigrationsAssembly=IdentityServerCore
Done. To undo this action, use 'ef migrations remove'

C:\Users\rafael\source\repos\IdentityServerCore\IdentityServerCore
A: dotnet ef migrations add InitPersistedGrantDbContext -c PersistedGrantDbContext -o Data/Migrations/PersistedGrant
Info: Microsoft.EntityFrameworkCore.Infrastructure[1040]
      Entity Framework Core 3.0.0 initialized 'PersistedGrantDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: MigrationsAssembly=IdentityServerCore
Done. To undo this action, use 'ef migration remove'

C:\Users\rafael\source\repos\IdentityServerCore\IdentityServerCore
A |
```

Aplique a migrações utilizando o comando “**dotnet ef database update**” para cada contexto e sua base de dados deverá está semelhante a imagem abaixo:

■ Tabelas
■ Tabelas do Sistema
■ FileTables
■ Tabelas Externas
■ Tabelas de Grafo
■ dbo._EFMigrationsHistory
■ dbo.ApiClaims
■ dbo.ApiProperties
■ dbo.ApiResources
■ dbo.ApiScopeClaims
■ dbo.ApiScopes
■ dbo.ApiSecrets
■ dbo.AspNetRoleClaims
■ dbo.AspNetRoles
■ dbo.AspNetUserClaims
■ dbo.AspNetUserLogins
■ dbo.AspNetUserRoles
■ dbo.AspNetUsers
■ dbo.AspNetUserTokens
■ dbo.ClientClaims
■ dbo.ClientCorsOrigins
■ dbo.ClientGrantTypes
■ dbo.ClientIdPRestrictions
■ dbo.ClientPostLogoutRed
■ dbo.ClientProperties
■ dbo.ClientRedirectUris
■ dbo.Clients
■ dbo.ClientScopes
■ dbo.ClientSecrets
■ dbo.DeviceCodes
■ dbo.IdentityClaims
■ dbo.IdentityProperties
■ dbo.IdentityResources
■ dbo.PersistedGrants
■ Exibições

Configurando o Seed do Identity Server v4

Precisamos criar uma classe que será responsável pela a carga em nossas tabelas. Ela irá utilizar a classe **IdentityServerConfiguration** para as inserções necessárias para o **Identity Server v4** funcionar corretamente.

Cria uma pasta chamada **Seed** e dentro desta pasta crie uma classe com o nome **SeedData**.

Coloque o código abaixo dentro da classe:

```
1. public class SeedData
2. {
3.     public static void EnsureSeedData(IServiceProvider serviceProvider)
4.     {
5.         PerformMigrations(serviceProvider);
6.
7.         EnsureSeedData(serviceProvider.GetRequiredService<ConfigurationDbContext>());
8.
9.         private static void PerformMigrations(IServiceProvider serviceProvider)
10.        {
11.            serviceProvider.GetRequiredService<ApplicationDbContext>
```

```
11.    () .Database.Migrate () ;
12.    serviceProvider.GetService<ConfigurationDbContext>
13.    () .Database.Migrate () ;
14.    serviceProvider.GetService<PersistedGrantDbContext>
15.    () .Database.Migrate () ;
16.    }
17.    private static void EnsureSeedData(ConfigurationDbContext context)
18.    {
19.        if (!context.Clients.Any())
20.        {
21.            foreach (var client in
22.                IdentityServerConfiguration.GetClientScope().ToList())
23.            {
24.                context.Clients.Add(client.ToEntity());
25.            }
26.        }
27.        if (!context.IdentityResources.Any())
28.        {
29.            foreach (var resource in
30.                IdentityServerConfiguration.GetIdentityResources().ToList())
31.            {
32.                context.IdentityResources.Add(resource.ToEntity());
33.            }
34.        }
35.    }
36.    if (!context.ApiResources.Any())
37.    {
38.        foreach (var resource in
39.            IdentityServerConfiguration.GetApiResources().ToList())
40.        {
41.            context.ApiResources.Add(resource.ToEntity());
42.        }
43.    }
44.}
45.
46.}
47.}
```

Feito isso, ainda temos mais um passo, vamos alterar o **Program.cs** para garantir que os dados estarão na base de dados.

Abra o **Program.cs** e adicione as linhas de código conforme exemplo abaixo:

```
1. static void Main(string[] args)
```

```

2.     {
3.         var host = CreateHostBuilder(args).Build();
4.
5.         using (var scope = host.Services.CreateScope())
6.         {
7.             var services = scope.ServiceProvider;
8.
9.             try
10.            {
11.                SeedData.EnsureSeedData(services);
12.            }
13.            catch (Exception ex)
14.            {
15.                var logger = services.GetRequiredService<ILogger<Program>>();
16.                logger.LogError(ex.ToString());
17.            }
18.        }
19.
20.        host.Run();
21.    }

```

Rodando o Identity Server v4

Com tudo configurado, vamos executar o **Identity Server v4**, se tudo ocorreu bem, você irá conseguir fazer uma requisição e ver resultado da resposta do **Identity Server v4** conforme imagem abaixo:

The screenshot shows the Postman application interface. The URL in the address bar is `https://localhost:44362/.well-known/openid-configuration`. The response status is 200 OK, and the response body is a JSON object representing the OpenID Configuration document.

```

{
  "issuer": "https://localhost:44362",
  "jwks_uri": "https://localhost:44362/.well-known/openid-configuration/jwks",
  "authorization_endpoint": "https://localhost:44362/connect/authorize",
  "token_endpoint": "https://localhost:44362/connect/token",
  "userinfo_endpoint": "https://localhost:44362/connect/userinfo",
  "end_session_endpoint": "https://localhost:44362/connect/endSession",
  "check_session_iframe": "https://localhost:44362/connect/checkSession",
  "revocation_endpoint": "https://localhost:44362/connect/revocation",
  "introspection_endpoint": "https://localhost:44362/connect/introspect",
  "device_authorization_endpoint": "https://localhost:44362/connect/deviceAuthorization",
  "frontchannel_logout_supported": true,
  "frontchannel_logout_session_supported": true,
  "backchannel_logout_supported": true,
  "backchannel_logout_session_supported": true,
  "scopes_supported": [
    "profile",
    "openid",
    "api",
    "offline_access"
  ]
}

```

E se chamarmos a API de Token devemos ter uma resposta conforme imagem abaixo:

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', 'Help', and a 'Logout' button. Below the navigation is a header bar with 'My Workspace' and 'Invite' buttons. The main workspace shows a 'POST https://localhost:44362/connect/token' request. The 'Body' tab is selected, showing a JSON payload with fields: grant_type (client_credentials), scope (API), client_id (7980CCE2-0750-45BC-BFA3-A9D5F9B2B5), and client_secret (1234567890). The response status is 200 OK, with a timestamp of 2023-04-11T11:36:44.948Z, a duration of 26ms, and a size of 940 B. The response body contains a JSON object with an access_token, expires_in, token_type, and scope fields.

Nosso **Identity Server v4** está pronto.

Criando um Client Web API e utilizando o Identity Server como Gerador de Token

Agora vamos criar uma aplicação cliente por exemplo um Web API que precisará ter uma API segura. Para isso crie um novo projeto **ASP.NET Core Web Application**. No template padrão do **ASP.NET Core Web Application** é criada uma API chamada **WeatherForecastController**.

Vamos configurar nosso cliente para utilizar o **Identity Server v4** como Autenticador/Autorizador para isso abra o arquivo **Startup.cs** e adicione o código abaixo:

```
1. public class Startup
2. {
3.     public Startup(IConfiguration configuration)
4.     {
5.         Configuration = configuration;
6.     }
7.
8.     public IConfiguration Configuration { get; }
9.
10.    // This method gets called by the runtime. Use this method to add
11.    // services to the container.
12.    public void ConfigureServices(IServiceCollection services)
13.    {
14.        services.AddControllers();
15.
16.        services.AddAuthentication("Bearer")
17.            .AddJwtBearer("Bearer", o =>
18.            {
19.                o.Authority = "https://localhost:44387";
20.                o.RequireHttpsMetadata = true;
21.            });
22.
23.        services.AddControllersWithViews();
24.    }
25.
```

```
20.             o.Audience = "API";
21.         });
22.
23.     }
24.
25.     // This method gets called by the runtime. Use this method to configure
26.     // the HTTP request pipeline.
27.     public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
28.     {
29.         if (env.IsDevelopment())
30.         {
31.             app.UseDeveloperExceptionPage();
32.
33.             app.UseHttpsRedirection();
34.
35.             app.UseRouting();
36.
37.             app.UseAuthentication();
38.             app.UseAuthorization();
39.
40.             app.UseEndpoints(endpoints =>
41.             {
42.                 endpoints.MapControllers();
43.             });
44.         }
45.     }
}
```

Adicione o pacote **Microsoft.AspNetCore.Authentication.JwtBearer** para que possamos utilizar **JWT Bearer Token**

Altere o código adicionando o atributo “**Authorize**” no controller **WeatherForecastController** e ao fazer uma requisição deverá retornar um **401 – Unauthorized** conforme imagem abaixo:

The screenshot shows the Postman application interface. A GET request is made to `https://localhost:44332/weatherforecast`. The response status is **401 Unauthorized**, with a time of 330ms and a size of 171 B. The response body is empty.

Nossa API está segura, agora para podermos utilizá-la devemos criar um token gerado pelo **Identity Server v4** e passar no Header de Autenticação da nossa API.

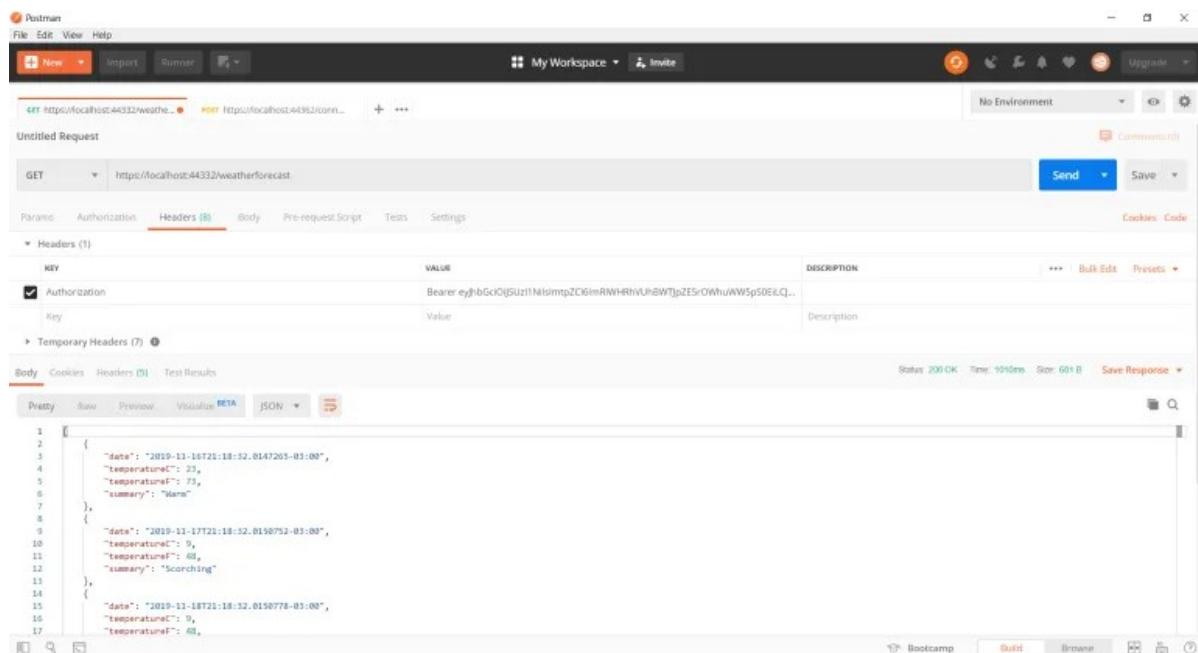
Veja nas imagens abaixo:

The screenshot shows the Postman application interface. A POST request is made to `https://localhost:44362/connect/token`. The response status is **200 OK**, with a time of 151ms and a size of 940 B. The response body is a JSON object containing an access token, token type (Bearer), and scope (API).

```

1 {
2   "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1bW1hL21pZC11IiwiYXpwIjoiMjQwMjE1LC30eXA1DlJ3eKtgf3Q1fQ.wy3uYmY1OjE1Izdm4Nj90fzI+5wW4c16MTU3NcgJNzAdMiniaXHt2joiaHR0cHM6Ly9vZ2hlbGhvcc3Q80Qdf311C3hd02iD1JBUEkicC3jg6L11rFfmQ20113
3   "token_type": "Bearer",
4   "scope": "API"
5 }

```



Conclusão

O **JSON Web Token (JWT)** facilita e padroniza a forma de separar o servidor de autorização e o servidor de recursos. O **JWT** é amplamente utilizado para garantir a segurança de uma **API REST** pois todo o token é assinado digitalmente.

Chegamos ao final deste post no qual é uma atualização deste post [Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API](#) que fiz sobre **JWT**. Resovi atualizar abordando tecnologias recentes como o **ASP.NET Identity Core**, **EntityFramework Core** e o **Identity Server v4**.

Espero que gostem desta atualização também deixo meu muito obrigado a vocês que leem meus artigos, espero que eles estejam ajudando vocês =]

O código fonte deste artigo está no meu GitHub através deste [link](#)

Abs e até o próximo

Compartilhe:



Relacionado

[Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API](#)
13 de maio de 2016

[Validando CSRF Token em chamada Ajax com MVC](#)
28 de janeiro de 2016
Em "Angular JS"

[Seja bem vindo, ASP.NET CORE 1.0](#)
14 de fevereiro de 2016
Em "ASP.NET MVC"

Em "Arquitetura"

 Arquitetura, ASP.NET MVC, C#, Entity Framework

 .NET Core, C#, Entity Framework, IdentityServer

« Coders in Rio Summit 2019 – O evento que
marcou o Rio de Janeiro

Compactação de Resposta no ASP.NET Core Web
API com Brotli »

2 Replies to “[Updated] Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API – (ASP.NET Identity + Identity Server v4 + EntityFrameworkCore)”



Adriano says:

18 de fevereiro de 2020 at 17:59

Rafael, muito bom esse tutorial, bem explicado e objetivo.

Estou montando um projeto e quero utilizar nele o ASPNET IDENTITY isolado do restante, para isso pensei numa estrutura dessa forma:

- 1 – PROJ.ASPNETIDENTITY
- 2 – PROJ.MVC
- 3 – PROJ.API
- 4 – PROJ.DOMAIN
- 5 – PROJ.DATA
- 6 – PROJ.CROSSCUTTING

Futuramente irei implementar o XAMARIN para criar aplicacao MOBILE , voce acha que estou complicando muito ou é um padrão existente ? Caso seja, voce tem algum artigo publicado ou algum documento que chegue próximo a essa estrutura ?

Forte abraço

[Responder](#)



rbento says:

19 de fevereiro de 2020 at 14:50

Fala ai Adriano.

Não está complicando não, acho que está no caminho correto.

No meu canal do youtube, existe um video explicando como vc faz com o ASP.NET Core

Identity.

Procura no youtube: <https://www.youtube.com/c/canalcodersinrio>

Abs.,

Rafael Cruz

[Responder](#)

Deixe uma resposta

O seu endereço de e-mail não será publicado. Campos obrigatórios são marcados com *

Comentário

Nome *

E-mail *

Site

PUBLICAR COMENTÁRIO

Notifique-me sobre novos comentários por e-mail.

Notifique-me sobre novas publicações por e-mail.

Esse site utiliza o Akismet para reduzir spam. [Aprenda como seus dados de comentários são processados.](#)



Search



Assine o blog

Digite seu endereço de e-mail para receber notificações de novas publicações por e-mail.

Endereço de e-mail

Assinar

Arquivos

- > abril 2020
- > fevereiro 2020
- > janeiro 2020
- > dezembro 2019
- > novembro 2019
- > outubro 2019
- > setembro 2019
- > junho 2019
- > maio 2019
- > abril 2019
- > março 2019
- > fevereiro 2019
- > outubro 2018
- > agosto 2018
- > julho 2018
- > junho 2018
- > maio 2018
- > abril 2018
- > março 2018
- > fevereiro 2018
- > janeiro 2018
- > novembro 2017
- > setembro 2017
- > julho 2017
- > maio 2017
- > abril 2017
- > março 2017
- > fevereiro 2017

- > janeiro 2017
- > dezembro 2016
- > novembro 2016
- > outubro 2016
- > setembro 2016
- > agosto 2016
- > julho 2016
- > junho 2016
- > maio 2016
- > abril 2016
- > março 2016
- > fevereiro 2016
- > janeiro 2016

Categorias

- > ALM
- > Angular JS
- > Arquitetura
- > ASP.NET MVC
- > C#
- > Cloud Service
- > Cursos & Palestras
- > Dicas
- > Entity Framework
- > Ferramentas
- > Javascript
- > Mobile

- > Outros
- > ReactJs
- > Sem categoria
- > Visual Studio
- > Windows Azure

Tópicos recentes

- > Construindo um Windows Service ou Linux Daemon com Worker Service & .NET Core – Parte 1
- > Boas práticas em construção de API
- > Criando padrões de resposta em suas APIs com ASP.NET Core
- > Compactação de Resposta no ASP.NET Core Web API com Brotli
- > [Updated] Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API – (ASP.NET Identity + Identity Server v4 + EntityFrameworkCore)

Tags

.NET Core .NET Core 1 AJAX Akka.NET ALM Angular ASP.NET Azure BDD
Bot C# Design Patterns Entity Framework Ferramentas IdentityServer Javascript
JSON JWT Linux NBench NoSQL ORM OWIN Parallel Programação React
RequireJS Rx.NET SaaS SaaSKit Segurança SelfHost Task TDD Tenant TFS
Thread TypeScript Ubuntu Unit Test Visual Studio Visual Studio 2017 VS2015
VS2017 WebApi

© 2020 Rafael Cruz

Powered by WordPress / Theme by Design Lab