

# Rafael Cruz



Home    About    Contato

## Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API

⌚ 13 de maio de 2016 🙋 rbento 💬 48

Fala Galera,

Hoje vamos falar sobre como podemos melhorar a segurança do nosso **Web API** utilizando **OAuth JSON Web Token (JWT)**. Porém primeiro precisamos entender o que é o **JSON Web Token (JWT)** e porque necessitamos de utilizar o **JWT**.

Gostaria de fazer uma implementação do JWT com o .NET Core 3.0 utilizando Identity Server v4 e EntityFramework Core? Não perca a versão mais recente deste conteúdo, [clique aqui e leia o post Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API – \(ASP.NET Identity + Identity Server v4\)](#)

### 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 JSON Web Token no ASP.NET Web API e OWIN Middleware

Não há um suporte direto para a criação de **JSON Web Token** no ASP.NET Web API sendo assim para que possamos criar nossos **Tokens** precisamos implementar a interface **ISecureDataFormat**.

Para consumir um **JSON Web Token** porém já existe um pacote chamado "Microsoft.Owin.Security.Jwt" que entende e deserializa os **Tokens** com pouca linhas de código, sendo assim a maior parte do trabalho é a codificação do servidor de autenticação.

Então vamos ao código!!!

### Criando um Servidor de Autenticação Web Api

Vamos criar um novo Projeto Web API eu coloquei com o nome de "JWT" mas sinta-se a vontade para usar um nome de sua preferencia =]

Para se criar um servidor de autenticação precisamos adicionar alguns pacotes, eles são encontrados no **NuGet** então nossa vida fica muito mais fácil.

Com o projeto criado vamos ao **Package Manager Console** e adicione os seguintes pacotes:

- Install-Package Microsoft.Owin.Cors
- Install-Package System.IdentityModel.Tokens.Jwt -Version 4.0.2.206221351
- Install-Package Thinktecture.IdentityModel.Core

Cada pacote tem sua responsabilidade dentro do processo de criação do **Token JWT**. O pacote **System.IdentityModel.Tokens.Jwt** ele é responsável por validar, parsear e gerar os tokens enquanto o **Thinktecture.IdentityModel.Core** será útil quando formos precisar assinar os **Tokens**

Vamos abrir o arquivo **Startup.Auth.cs** e adicionar o código abaixo:

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using Microsoft.AspNet.Identity;
5.  using Microsoft.AspNet.Identity.EntityFramework;
6.  using Microsoft.Owin;
7.  using Microsoft.Owin.Security.Cookies;
8.  using Microsoft.Owin.Security.Google;
9.  using Microsoft.Owin.Security.OAuth;
10. using Owin;
11. using JsonWebToken.Providers;
12. using JsonWebToken.Models;
13.
```

```
14. namespace JsonWebToken
15. {
16.     public partial class Startup
17.     {
18.         public void ConfigureAuth(IAppBuilder app)
19.         {
20.             OAuthAuthorizationServerOptions authServerOptions = new
21.             OAuthAuthorizationServerOptions()
22.             {
23.                 //Em produção se atentar que devemos usar HTTPS
24.                 AllowInsecureHttp = true,
25.                 TokenEndpointPath = new PathString("/oauth2/token"),
26.                 AccessTokenExpireTimeSpan = TimeSpan.FromMinutes(30),
27.                 Provider = new CustomOAuthProvider(),
28.                 AccessTokenFormat = new CustomJwtFormat("http://localhost")
29.             };
30.             app.UseOAuthAuthorizationServer(authServerOptions);
31.         }
32.     }
33. }
```

Pronto configuramos nosso servidor de autenticação JWT, porém nosso projeto ainda não está compilando precisamos implementar as classes **CustomOAuthProviderJwt** e **CustomJwtFormat**.

Então vamos ao trabalho.

Vamos primeiro criar a classe **CustomOAuthProviderJwt**, com a classe criada adicione o código abaixo:

```
1. using Microsoft.Owin.Security;
2. using Microsoft.Owin.Security.OAuth;
3. using System.Collections.Generic;
4. using System.Security.Claims;
5. using System.Threading.Tasks;
6.
7. namespace JsonWebToken
8. {
9.     internal class CustomOAuthProviderJwt : OAuthAuthorizationServerProvider
10.    {
11.        public override Task
12.        ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
13.        {
14.            string clientId = string.Empty;
15.            string clientSecret = string.Empty;
16.            string symmetricKeyAsBase64 = string.Empty;
17.            if (!context.TryGetBasicCredentials(out clientId, out clientSecret))
18.        }
19.    }
20. }
```

```
18.          {
19.              context.TryGetFormCredentials(out clientId, out clientSecret);
20.          }
21.
22.          if (context.ClientId == null)
23.          {
24.              context.SetError("invalid_clientId", "client_Id não pode ser
nulo");
25.              return Task.FromResult<object>(null);
26.          }
27.
28.          context.Validated();
29.          return Task.FromResult<object>(null);
30.      }
31.
32.      public override Task
GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
33.      {
34.
35.          context.OwinContext.Response.Headers.Add("Access-Control-Allow-
Origin", new[] { "*" });
36.
37.          //FAKE FAZER A VALIDAÇÃO NO BANCO DE DADOS
38.          if (context.UserName != context.Password)
39.          {
40.              context.SetError("invalid_grant", "Usuário ou senha invalidos");
41.              return Task.FromResult<object>(null);
42.          }
43.
44.          var identity = new ClaimsIdentity("JWT");
45.
46.          identity.AddClaim(new Claim(ClaimTypes.Name, context.UserName));
47.          identity.AddClaim(new Claim("sub", context.UserName));
48.          identity.AddClaim(new Claim(ClaimTypes.Role, "Administrator"));
//PEGAR AS ROLES CORRETAS
49.
50.          var props = new AuthenticationProperties(new Dictionary<string,
string>
51.          {
52.              {
53.                  "audience", (context.ClientId == null) ? string.Empty :
context.ClientId
54.              }
55.          });
56.
57.          var ticket = new AuthenticationTicket(identity, props);
58.          context.Validated(ticket);
59.          return Task.FromResult<object>(null);
60.      }
61.
62.  }
63. }
```

Ainda falta implementar a classe **CustomJwtFormat**, essa classe será responsável por gerar os **Tokens JWT** então vamos adicionar o código abaixo na classe.

```
1.  using Microsoft.Owin.Security;
2.  using Microsoft.Owin.Security.DataHandler.Encoder;
3.  using System;
4.  using System.Collections.Generic;
5.  using System.IdentityModel.Tokens;
6.  using System.Linq;
7.  using System.Web;
8.  using Thinktecture.IdentityModel.Tokens;
9.
10. namespace JsonWebToken
11. {
12.     public class CustomJwtFormat : ISecureDataFormat<AuthenticationTicket>
13.     {
14.         private readonly string _issuer = string.Empty;
15.         private readonly string Base64Secret =
16.             "IxraJDoa2FqElO7IhrSrUJELhUckePEPVpaePlS_Xaw";
17.         public CustomJwtFormat(string issuer)
18.         {
19.             _issuer = issuer;
20.         }
21.         public string Protect(AuthenticationTicket data)
22.         {
23.             if (data == null)
24.                 throw new ArgumentNullException("data");
25.
26.             string symmetricKeyAsBase64 = Base64Secret;
27.
28.             var keyByteArray =
29.                 TextEncodings.Base64Url.Decode(symmetricKeyAsBase64);
30.
31.             var signingKey = new HmacSigningCredentials(keyByteArray);
32.
33.             var issued = data.Properties.IssuedUtc;
34.
35.             var expires = data.Properties.ExpiresUtc;
36.
37.             var token = new JwtSecurityToken(_issuer, null,
38.                 data.Identity.Claims, issued.Value.UtcDateTime, expires.Value.UtcDateTime,
39.                 signingKey);
40.
41.             var handler = new JwtSecurityTokenHandler();
42.             var jwt = handler.WriteToken(token);
43.
44.             return jwt;
45.         }
46.     }
47. }
```

```

43.         }
44.
45.         public AuthenticationTicket Unprotect(string protectedText)
46.         {
47.             throw new NotImplementedException();
48.         }
49.     }
50. }

```

Tudo pronto na nossa implementação de **JWT** para testar usarei o **Postman** para enviar as requisições e gerar um **Token** de acesso.

Veja na imagem abaixo:

The screenshot shows a Postman interface with a red box highlighting the request body. The request method is POST, and the URL is `http://localhost:1741/oauth2/token`. The request body is set to `form-data` and contains the following fields:

Key	Value
username	admin
password	admin
grant_type	password
client_id	099153c2625149bc8ecb3e85e03f0022

The response section shows a `200 OK` status with a response time of `37420 ms`. The response body is displayed in JSON format:

```

{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmldWVfbmFtZSI6ImFkbWluIiwic3ViIjoiYWRtaW4iLCJyb2xlijo1QWRtaW5pc3RyYXRvcicIsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6MTc0MS8iLCJ0dHA6Ly9sb2NhbGhvc3Q6MTc0MS8iLCJleHAiOjE0NjMxMDQ3MjMsIm5iZiI6MTQ2MzEwMjkyM30.kvO-1tZg7iB5WdowjD_YiilHVBDPiU6RM1a8RqYnb6c",
  "token_type": "bearer",
  "expires_in": 1799
}

```

Geramos nosso **Token** como podem ver na resposta abaixo:

```

1. eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmldWVfbmFtZSI6ImFkbWluIiwic3ViIjoiYWRtaW4iLCJyb2xlijo1QWRtaW5pc3RyYXRvcicIsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6MTc0MS8iLCJ0dHA6Ly9sb2NhbGhvc3Q6MTc0MS8iLCJleHAiOjE0NjMxMDQ3MjMsIm5iZiI6MTQ2MzEwMjkyM30.kvO-1tZg7iB5WdowjD_YiilHVBDPiU6RM1a8RqYnb6c

```

Notem que existem um parâmetro chamado **client\_id**, esse parâmetro é um **Hash Code** aleatório. A principal função dele é validar o acesso ao recursos, ou seja, mesmo que o usuário e senha estejam corretos, temos que validar se a origem da requisição é válida. Então resumindo, sempre que alguém quiser nossa **Api** deverá ser cadastrada na base de dados e geraremos uma chave de acesso para

ele. Essa chave de acesso é o **client\_id**.

Vamos validar se nosso **Token** realmente foi gerado de forma correta.

Existe um programa online chamado [jwt.io](http://jwt.io), ele faz a verificação online de **Token JWT**.

Veja na imagem abaixo a validação do nosso **Token**

The screenshot shows the jwt.io interface. On the left, under 'Encoded', the token is displayed as a long string of characters: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmxdWVfbmFtZSI6ImFkbWluIiwic3ViIjoiYWRtaW4iLCJyb2xlijoiQWRtaW5pc3RyYXRvcIIsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6MTc0MS8iLCJleHAiOjE0NjMxMDQ3MjMsIm5iZiI6MTQ2MzEwMjkyM30.kv0-1tZg7iB5WdowjD\_YiilHVBDPiU6RM1a8RqYnb6c. The right side, under 'Decoded', shows the token's structure:

```

HEADER: ALGORITHM & TOKEN TYPE
{
  "typ": "JWT",
  "alg": "HS256"
}

PAYLOAD: DATA
{
  "unique_name": "admin",
  "sub": "admin",
  "role": "Administrator",
  "iss": "http://localhost:1741/",
  "exp": 1463104723,
  "nbf": 1463102923
}

VERIFY SIGNATURE
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) secret base64 encoded
  
```

A red banner at the bottom states: **⊗ Invalid Signature**.

Como podem ver, nosso **Token** está com a assinatura inválida. Para corrigir vamos pegar a chave de assinatura que usamos para gerar o **Token** e colocar na caixa de texto com o nome **Secret**. Agora nosso **Token** está com a assinatura válida, veja na imagem abaixo

The screenshot shows the jwt.io interface again. The token remains the same as before. The right side, under 'Decoded', shows the token's structure:

```

HEADER: ALGORITHM & TOKEN TYPE
{
  "typ": "JWT",
  "alg": "HS256"
}

PAYLOAD: DATA
{
  "unique_name": "admin",
  "sub": "admin",
  "role": "Administrator",
  "iss": "http://localhost:1741/",
  "exp": 1463104723,
  "nbf": 1463102923
}

VERIFY SIGNATURE
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) secret base64 encoded
  
```

This time, the 'VERIFY SIGNATURE' section shows the correct signature calculation, indicating a valid signature.



## Gerando AccessKey e ClientId dinamicamente.

Para gerar nosso **client\_id**, ou seja, a chave que dará acesso a nossa **API** devemos criar implementação de geração de **Token**, ai vem a pergunta, porque deveríamos gerar **ClientId** e um **AccessKey** se o usuário e senha me basta ?

A resposta é simples, como vamos garantir que somente dispositivos ou programas autorizados podem ter acesso a nossa **API** ? Então imagine o cenário no qual nossa **API** é acessado por um **APP** que construirmos, temos que garantir que somente nosso **APP** seja autorizado a acessar esses recursos e é ai que entra o **AccessKey** e **ClientId**.

Vamos criar nossa classe que controlará a geração do **AccessKey** e **ClientId**. Adicione uma nova classe chamada **WebApplicationAccess** e adicione o código abaixo. Notem que no seu construtor estático já estou adicionando um acesso para facilitar nossos testes.

```
1. public class WebApplicationAccess
2. {
3.     public String ClientId { get; set; }
4.     public String AccessKey { get; set; }
5.     public String SecretKey { get; set; }
6.     public String ApplicationName { get; set; }
7.
8.     public static ConcurrentDictionary<string, WebApplicationAccess>
9.         WebApplicationAccessList = new ConcurrentDictionary<string,
10.            WebApplicationAccess>();
11.
12.
13.     static WebApplicationAccess()
14.     {
15.         WebApplicationAccessList.TryAdd("e84a2d13704647d18277966ec839d39e",
16.             new WebApplicationAccess()
17.             {
18.                 AccessKey = "CgP7NyLXtaGmyOgjj3sUMwmAlrSKqa5JyZ4P1OlQeM",
19.                 SecretKey = "UTboSKRUb13ZmztLtyB0W4BN37ndx_aK8__ry9sxfv8",
20.                 ApplicationName = "ApplicationTests",
21.                 ClientId = "e84a2d13704647d18277966ec839d39e"
22.             });
23.     }
24.
25.     public static WebApplicationAccess GrantApplication(string name)
```

```
23.         {
24.             var clientId = Guid.NewGuid().ToString("N");
25.
26.             var key = new byte[32];
27.             RNGCryptoServiceProvider.Create().GetBytes(key);
28.             var base64Secret = TextEncodings.Base64Url.Encode(key);
29.
30.
31.             var accessKey = new byte[32];
32.             RNGCryptoServiceProvider.Create().GetBytes(key);
33.             var accessKeyText = TextEncodings.Base64Url.Encode(key);
34.
35.             WebApplicationAccess newWebApplication = new WebApplicationAccess {
36.                 ClientId = clientId, SecretKey = base64Secret, AccessKey = accessKeyText,
37.                 ApplicationName = name };
38.             WebApplicationAccessList.TryAdd(clientId, newWebApplication);
39.             return newWebApplication;
40.         }
41.
42.         public static WebApplicationAccess Find(string clientId)
43.         {
44.             WebApplicationAccess webApplication = null;
45.             if (WebApplicationAccessList.TryGetValue(clientId, out
46.             webApplication))
47.             {
48.                 return webApplication;
49.             }
50.         }
```

Nossa classe que irá gerenciar os acessos a nossa API está criada.

Vamos adicionar um **WebApi** chamado **WebApplicationAccessController** para criar nosso **Token**. Esse **WebApi Controller** só tem um recurso que é para criar as chaves de acesso para a nossa API, obviamente por motivos de exemplo ele não irá gravar no banco de dados mas na vida real iríamos gravar na base de dados e assim consultar posteriormente.

Vamos ao código.

```
1.  public class WebApplicationAccessController : ApiController
2.  {
3.      // POST api/<controller>
4.      public IHttpActionResult Post([FromBody] string applicationName)
5.      {
6.          var webApplication =
7.              WebApplicationAccess.GrantApplication(applicationName);
8.          return Ok<WebApplicationAccess>(webApplication);
9.      }
```

9. }

Com tudo criado, devemos agora alterar nossa classe chamada **CustomOAuthProviderJwt** para validar o **ClientId** e o **AccessKey**.

## Validando o ClientId e AccessKey no CustomOAuthProviderJwt

Vamos alterar o **CustomOAuthProviderJwt** para validar o **ClientId** e o **AccessKey**. Devemos alterar o trecho no qual validamos o **ClientId**. Esse método é o **ValidateClientAuthentication**. Vamos ao código do **CustomOAuthProviderJwt**.

```
1. internal class CustomOAuthProviderJwt : OAuthAuthorizationServerProvider
2. {
3.     public override Task
4.     ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
5.     {
6.         string clientId = string.Empty;
7.         string clientSecret = string.Empty;
8.         string symmetricKeyAsBase64 = string.Empty;
9.
10.        if (!context.TryGetBasicCredentials(out clientId, out clientSecret))
11.        {
12.            context.TryGetFormCredentials(out clientId, out clientSecret);
13.        }
14.
15.        if (context.ClientId == null)
16.        {
17.            context.SetError("invalid_clientId", "client_Id não pode ser
nulo");
18.            return Task.FromResult<object>(null);
19.        }
20.
21.        //Procurando pelo Client Id
22.        var token = context.ClientId.Split(':');
23.
24.        var client_id = token.First();
25.        var accessKey = token.Last();
26.
27.        var applicationAccess = WebApplicationAccess.Find(client_id);
28.
29.        if (applicationAccess == null)
30.        {
31.            context.SetError("invalid_clientId", "client_Id não encontrado");
32.            return Task.FromResult<object>(null);
33.        }
34.
35.        if (applicationAccess.AccessKey != accessKey)
36.        {
37.            context.SetError("invalid_clientId", "access key não encontrado
ou inválido");
38.        }
39.    }
40. }
```

```

37.             return Task.FromResult<object>(null);
38.         }
39.
40.         context.Validated();
41.         return Task.FromResult<object>(null);
42.     }
43.
44.     public override Task
45.     GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
46.     {
47.
48.         context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { "*" });
49.
50.         //FAKE FAZER A VALIDAÇÃO NO BANCO DE DADOS
51.         if (context.UserName != context.Password)
52.         {
53.             context.SetError("invalid_grant", "Usuário ou senha invalidos");
54.             return Task.FromResult<object>(null);
55.         }
56.
57.         var identity = new ClaimsIdentity("JWT");
58.
59.         identity.AddClaim(new Claim(ClaimTypes.Name, context.UserName));
60.         identity.AddClaim(new Claim("sub", context.UserName));
61.         identity.AddClaim(new Claim(ClaimTypes.Role, "Administrator"));
62.         //PEGAR AS ROLES CORRETAS
63.
64.         var props = new AuthenticationProperties(new Dictionary<string,
65.                                                 string>
66.         {
67.             {
68.                 "audience", (context.ClientId == null) ? string.Empty :
69.                 context.ClientId
70.             }
71.         });
72.
73.         var ticket = new AuthenticationTicket(identity, props);
74.         context.Validated(ticket);
75.         return Task.FromResult<object>(null);
76.     }

```

Nossa classe **CustomOAuthProviderJwt** agora valida o **ClientId** e o **AccessKey**. Porém ainda temos mais trabalho a fazer, devemos alterar o **CustomJwtFormat** para assinar o **Token por ClientId**. Vamos alterar o método **Protect**, adicione o código abaixo:

```

1. public class CustomJwtFormat : ISecureDataFormat<AuthenticationTicket>
2. {
3.     private readonly string _issuer = string.Empty;
4.     public CustomJwtFormat(string issuer)
5.     {

```

```
6.             _issuer = issuer;
7.         }
8.
9.         public string Protect(AuthenticationTicket data)
10.        {
11.            if (data == null)
12.                throw new ArgumentNullException("data");
13.
14.
15.            string audience = data.Properties.Dictionary["audience"];
16.
17.            if (string.IsNullOrWhiteSpace(audience)) throw new
18. InvalidOperationException("ClientId e AccessKey não foi encontrado");
19.
20.            var keys = audience.Split(':');
21.
22.            var client_id = keys.First();
23.            var accessKey = keys.Last();
24.
25.
26.            var applicationAccess = WebApplicationAccess.Find(client_id);
27.
28.            var keyByteArray =
29. TextEncodings.Base64Url.Decode(applicationAccess.SecretKey);
30.
31.            var signingKey = new HmacSigningCredentials(keyByteArray);
32.
33.            var issued = data.Properties.IssuedUtc;
34.
35.            var expires = data.Properties.ExpiresUtc;
36.
37.            var token = new JwtSecurityToken(_issuer, client_id,
38. data.Identity.Claims, issued.Value.UtcDateTime, expires.Value.UtcDateTime,
39. signingKey);
40.
41.            var handler = new JwtSecurityTokenHandler();
42.
43.            var jwt = handler.WriteToken(token);
44.
45.            return jwt;
46.        }
47.
```

Agora vamos testar novamente. Notem que agora temos que passar no parâmetro **client\_id**, o **ClientId** e o **AccessKey** separado por dois pontos (:). Vamos voltar ao **Postman**.

The screenshot shows the Postman interface with a successful API call to `http://localhost:1741/oauth2/token`. The request method is POST, and the body contains the following form-data:

- username: admin
- password: admin
- grant\_type: password
- client\_id: e84a2d13704647d18277966ec839d

The response status is 200 OK, and the JSON payload is:

```

1 {
2   "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmJxdWVfbmFtZSI6ImFkbWluIiwic3ViIjoiYWRtaW4iLCJyb2x1IjoiQWRtaW5pc3RyYXRvcilsImlzcyI6Imh0dHA6Ly9sb2Nhbgv3lMTc0MS8iLCJhdWQ1OjI2ODRMNmQxMzcmNDY0N2QxODI3Nzk2NmVjODM5ZDM5ZS1sImV4cCI6MTQ2IzExMDA2NywibmJmIjoxNDYzMTA4MjY3fQ.IDFI-2PHEaeL12ikSGUSH8Vi00v0U1QFLCTaggTbSsI",
3   "token_type": "bearer",
4   "expires_in": 1799
5 }

```

Pronto nosso **Token** foi gerado com sucesso. Vamos valida-ló no [jwt.io](https://jwt.io)

The jwt.io interface shows the decoded token details. The token is a JWT with the following structure:

**Encoded**: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1bmJxdWVfbmFtZSI6ImFkbWluIiwic3ViIjoiYWRtaW4iLCJyb2x1IjoiQWRtaW5pc3RyYXRvcilsImlzcyI6Imh0dHA6Ly9sb2Nhbgv3lMTc0MS8iLCJhdWQ1OjI2ODRMNmQxMzcmNDY0N2QxODI3Nzk2NmVjODM5ZDM5ZS1sImV4cCI6MTQ2IzExMDA2NywibmJmIjoxNDYzMTA4MjY3fQ.IDFI-2PHEaeL12ikSGUSH8Vi00v0U1QFLCTaggTbSsI

**Decoded**:

```

HEADER: ALGORITHM & TOKEN TYPE
{
  "typ": "JWT",
  "alg": "HS256"
}

PAYLOAD: DATA
{
  "unique_name": "admin",
  "sub": "admin",
  "role": "Administrator",
  "iss": "http://localhost:1741/",
  "aud": "e84a2d13704647d18277966ec839d39e",
  "exp": 1463110067,
  "nbf": 1463108267
}

VERIFY SIGNATURE
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  [REDACTED]
) secret base64 encoded

```

**Signature Verified**

Notem agora que no **Payload** agora temos a propriedade **aud** essa propriedade está preenchida com o **ClientId** que está obtendo o **Token** também temos mais uma informação o **Token** foi assinado pelo **Secret Key** do **ClientId**, assim, cada **ClientId** tem seu próprio **SecretKey** tornando assim nosso **Token**

mais seguro.

## Configurando nossa Aplicação MVC para usar Bearer Token.

Com nossa implementação de **Token** toda feita, devemos configurar nossa aplicação usá-lo **Token** de **Autenticação**.

Adicione o seguinte pacote através do **Package Manager Console**

- Install-Package Microsoft.Owin.Security.Jwt

Depois do pacote adicionado, vamos abrir **Startup.Auth.cs** e adicione o código abaixo.

```
1. public partial class Startup
2. {
3.     // For more information on configuring authentication, please visit
4.     // http://go.microsoft.com/fwlink/?LinkId=301864
5.     public void ConfigureAuth(IAppBuilder app)
6.     {
7.         OAuthAuthorizationServerOptions authServerOptions = new
8.             OAuthAuthorizationServerOptions()
9.         {
10.             AllowInsecureHttp = true,
11.             TokenEndpointPath = new PathString("/oauth2/token"),
12.             AccessTokenExpireTimeSpan = TimeSpan.FromMinutes(30),
13.             Provider = new CustomOAuthProviderJwt(),
14.             AccessTokenFormat = new
15.                 CustomJwtFormat("http://localhost:1741/")
16.             };
17.
18.
19.         var issuer = "http://localhost:1741/";
20.         var audience =
21.             WebApplicationAccess.WebApplicationAccessList.Select(x =>
22.                 x.Value.ClientId).AsEnumerable();
23.         var secretsSymmetricKey = (from x in
24.             WebApplicationAccess.WebApplicationAccessList
25.                 select new
26.                     SymmetricKeyIssuerSecurityTokenProvider(issuer,
27.                         TextEncodings.Base64Url.Decode(x.Value.SecretKey))).ToArray();
28.
29.
30.         app.UseJwtBearerAuthentication(
31.             new JwtBearerAuthenticationOptions
32.             {
33.                 AuthenticationMode = AuthenticationMode.Active,
34.                 AllowedAudiences = audience,
35.                 IssuerSecurityTokenProviders = secretsSymmetricKey
36.             });
37.     }
38. }
```

```
31.         });
32.     }
33. }
```

No código acima estamos pegando todos os **ClientId** registrados com seus **Secret Keys** assim o servidor de Api será capaz de validar o **Token** gerado.

Com essa alteração podemos utilizar a anotação **[Authorize]** nos nossos **WebApi**, caso o **Token** não seja válido o acesso ao recurso será negado.

Vamos criar um **WebApi** básico para fazer nossos testes, vamos ao código:

```
1. [Authorize]
2. public class ValuesController : ApiController
3. {
4.     // GET api/values
5.     public IEnumerable<string> Get()
6.     {
7.         return new string[] { "value1", "value2" };
8.     }
9.
10.    // GET api/values/5
11.    public string Get(int id)
12.    {
13.        return "value";
14.    }
15.
16.    // POST api/values
17.    public void Post([FromBody]string value)
18.    {
19.    }
20.
21.    // PUT api/values/5
22.    public void Put(int id, [FromBody]string value)
23.    {
24.    }
25.
26.    // DELETE api/values/5
27.    public void Delete(int id)
28.    {
29.    }
30. }
```

Agora vamos voltar no **Postman** para fazer as requisições de testes. Primeiro devemos gerar o **Token**.



password: admin  
grant\_type: password  
client\_id: e84a2d13704647d18277966ec839d

```

1 {
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJibm1xdW9fbmFtZSI6ImFkbWluIiwic3ViIjoiYWRtaW4iLCJyb2x1IjoiQURtaW5pc3RvYXRvciiIsIm1zcyI6Imh0dHA6Ly9sb2NhbgHvc3Q6MTc0MS81LCJhdWQzIjI0DRhNQxIzcwIDY0h2QxODI3Nzk2MmV5ODM5ZDIsImV4cCI6MTQ2MzExMTgxNCiibmJmIjoxNDYzMTEwMDewfQ.I38c_a1EBnjQuz3xUeiyQQxna_ECFvFIW4_vmtj2Ygt4"
  "token_type": "bearer",
  "expires_in": 1799
}
  
```

Com o **Token** gerado, devemos passar ele no **Header** do **Request** para chamar nosso recurso. No cabeçalho **Header** o nome da chave que iremos utilizar será o **Authorization**.

Accept: application/json  
Content-Type: application/json  
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJibm1xdW9fbmFtZSI6ImFkbWluIiwic3ViIjoiYWRtaW4iLCJyb2x1IjoiQURtaW5pc3RvYXRvciiIsIm1zcyI6Imh0dHA6Ly9sb2NhbgHvc3Q6MTc0MS81LCJhdWQzIjI0DRhNQxIzcwIDY0h2QxODI3Nzk2MmV5ODM5ZDIsImV4cCI6MTQ2MzExMTgxNCiibmJmIjoxNDYzMTEwMDewfQ.I38c\_a1EBnjQuz3xUeiyQQxna\_ECFvFIW4\_vmtj2Ygt4

```

1 [
  "value1",
  "value2"
]
  
```

Passando o **Token** correto conseguimos acessar o recurso tranquilamente, agora vamos testar o caso de exceção, iremos passar um **Token** inválido. O servidor deverá retorna um **erro 403** de acesso não autorizado.

Accept: application/json  
Content-Type: application/json  
Authorization: Bearer 1832840dcjdhv0

The screenshot shows the Postman interface with the following details:

- Header**: A table with columns "Header" and "Value".
- Send**, **Preview**, **Add to collection** buttons.
- Reset** button.
- Body**: Cookies (2), Headers (11). STATUS: 401 Unauthorized, TIME: 145 ms.
- Pretty**, **Raw**, **Preview**, **X**, **JSON**, **XML** buttons.
- Code Preview** area:

```
1 {  
2     "Message": "Authorization has been denied for this request."  
3 }
```

Como podem ver, o servidor não aceitou a requisição ao recurso e retornou um erro de acesso não autorizado. Agora vamos testar não passando o **Token** de acesso, o servidor deverá retornar erro de acesso não autorizado.

The screenshot shows the Postman interface with the following details:

- Normal**, **Basic Auth**, **Digest Auth**, **OAuth 1.0**, **No environment**.
- http://localhost:1741/api/values**, **GET**, **URL params**, **Headers (2)**.
- Accept**: application/json, **Content-Type**: application/json, **Header**: Value.
- Send**, **Preview**, **Add to collection** buttons.
- Reset** button.
- Body**: Cookies (2), Headers (11). STATUS: 401 Unauthorized, TIME: 48 ms.
- Pretty**, **Raw**, **Preview**, **X**, **JSON**, **XML** buttons.
- Code Preview** area:

```
1 {  
2     "Message": "Authorization has been denied for this request."  
3 }
```

O servidor também não autorizou o acesso ao recurso pois não passamos o **Token** de acesso.

Assim chegamos ao fim do post, usando **JSON Web Token (JWT)** facilita e padroniza a forma de separar o servidor de autorização e o servidor de recursos.

Podemos usar o **JWT** para criar serviços **RestFul** seguros e compartilhados de formas seguras entre os clientes que usam os serviços.

O código fonte deste post se encontra no GitHub através deste [link](#).

Não deixem de comentar.

Abs e até o próximo.

---

Compartilhe:



---

#### Relacionado

[\[Updated\] Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API - \(ASP.NET Identity + Identity Server v4 + EntityFrameworkCore\)](#)  
15 de novembro de 2019  
Em "Arquitetura"

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

[Autenticação utilizando ASP.NET Core 2.0 Identity](#)  
13 de março de 2018  
Em "ASP.NET MVC"

---

Arquitetura, [ASP.NET MVC](#), [C#](#)

[ASP.NET](#), [C#](#), [JWT](#), [WebApi](#)

[« Team Foundation Server como ALM – Integração Implementando o Design Pattern Repository e Unit Contínua – Parte 3](#) [» Implementando o Design Pattern Repository e Unit Of Work com Entity Framework](#)

---

## 48 Replies to “Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API”



Eduardo says:

[13 de maio de 2016 at 18:53](#)

Muito bom, Rafael!

Vejo que, infelizmente, ainda existem muitos desenvolvedores ignorando a importância da segurança quando trabalham em uma aplicação. Acho que posts como esse podem ajudar por abordar da explicação teórica até a implementação.

Muito bom!

[Responder](#)



**rbento** says:

16 de maio de 2016 at 11:31

Valeu, Eduardo.

Espero realmente que meus posts ajudem de alguma forma a mudar esse pensamento.

Abs.,

[Responder](#)



**Vagner Mello** says:

15 de setembro de 2017 at 07:24

Parabéns pelo post! Vale a pena fazer um video. Muito obrigado 😊

[Responder](#)



**rbento** says:

21 de setembro de 2017 at 20:40

Realmente, estou considerando esta hipótese

Muito obrigado pela visita

Abs.,

[Responder](#)



**Paulo Eduardo Correia** says:

18 de maio de 2016 at 08:26

Rafael,

Muito bom o post eu implementei uma Aplicação com este tipo de configuração no inicio do mês, mas não utilizei apenas o Oauth com OWIN, pois não gostei do Json web token pois ele é apenas um json em base 64 que é facilmente convertido

Mas fiquei com uma dúvida neste modelo é possível ter um token de refresh ? pois na minha implementação é gerado um access\_token e refresh\_token evitando a necessidade de quem está consumindo enviar o usuário a senha quando o token expira

[Responder](#)



**rbento** says:

18 de maio de 2016 at 12:37

Oi Paulo,

Sim e possível gerar o refresh token sim, na configuração do JWT no Start.Auth.cs temos que configurar a geração de refresh\_token.

Depois temos que alterar a forma de geração do token no JWTCustomFormat.

Abs.,

[Responder](#)



**Paulo Rogério** says:

18 de maio de 2016 at 10:17

Seu artigo é ótimo, mas gostaria de deixar a dica de gerar uma vídeo aula, irá ser bem interessante também.

[Responder](#)



**rbento** says:

18 de maio de 2016 at 12:39

Muito Obrigado, Paulo.

Então estava com um canal no youtube mas por questão profissionais acabei não postando mais videos.

Tento no meu post passar o passo a passo de um forma clara.

Abs.,

[Responder](#)



**Mauricio** says:

10 de junho de 2016 at 16:54

Rafael,

JWT são diferentes dos tokens gerados pela próprio oAuth no .net ?

[Responder](#)



**rbento** says:

10 de junho de 2016 at 18:26

Na verdade não. Tokens JWT estão contido no modelo de autenticação do OAuth e não é exclusivo da linguagem .NET a geração do token e sim pelo modelo de autenticação OAuth. Existe uma RFC 7519 (<https://tools.ietf.org/html/rfc7519>) que trata justamente disso sobre o modelo de autenticação JWT com OAuth,

Abs e obrigado pela visita

[Responder](#)



**Alexandre** says:

11 de outubro de 2016 at 20:02

Muito bom o artigo. Uma observação, está ocorrendo um erro no CustomJwtFormat

Error CS1503 Argument 6: cannot convert from  
'Thinktecture.IdentityModel.Tokens.HmacSigningCredentials' to  
'Microsoft.IdentityModel.Tokens.SigningCredentials'

Se este erro está acontecendo pode ser pq vc está usando a ultima versão do nuget.

System.IdentityModel.Tokens.Jwt

Portanto o procedimento é desinstalar o System.IdentityModel.Tokens.Jwt e instalar o da versão 4.0.2.206221351

Segue abaixo:

Uninstall "System.IdentityModel.Tokens.Jwt"

Install-Package System.IdentityModel.Tokens.Jwt -Version 4.0.2.206221351

[Responder](#)



**rbento** says:

12 de outubro de 2016 at 23:49

Oi Alexandre,

Muito obrigado pela visita.

Realmente está acontecendo esse problema.

Vou verificar e corrigir.

Abs.,

[Responder](#)



**Adriano** says:

16 de novembro de 2016 at 20:34

Boa noite Rafael, gostaria de tirar uma dúvida, tem como controlar os tokens gerados por usuário por exemplo: cada usuário só poderá ter um token ativo, se ele gerar um segundo token o anterior é apagado. Não estou entendendo como ele mantém esses tokens ativos ou como posso acessar eles.

[Responder](#)



**rbento** says:

18 de novembro de 2016 at 16:24

Oi Adriano,

Cada token é gerado por usuário, ou seja, o usuário somente terá um token ativo, caso o token esteja expirado o próprio sistema gera um automaticamente.

Para melhor entendimento, o token é o como se fossem uma sessão do usuário no ASP.NET, aonde cada usuário somente tem uma sessão válida e caso ela se expire ele tem que se logar novamente.

Para acessar os token podemos usar dentro do WebAPI a propriedade User que fica dentro do ApiController. Lá contém as informações de acesso do usuário.

Abs e obrigado pela visita

[Responder](#)



**Greidimar** says:

16 de janeiro de 2017 at 10:01

Não comprehendi onde o servidor salva os tokens. Vejo a aplicação gerar os tokens mas não vejo eles serem salvos .

[Responder](#)



**rbento** says:

[16 de janeiro de 2017 at 16:46](#)

Oi Greidimar,

O Servidor não salva o token, o que ele faz é validar a sua assinatura baseados em algoritmos como HMAC256.

Abs.,

Rafael Cruz

[Responder](#)



**Ulysses Alves** says:

[21 de fevereiro de 2017 at 17:43](#)

"Então imagine o cenário no qual nossa API é acessado por um APP que construirmos, temos que garantir que somente nosso APP seja autorizado a acessar esses recursos e é ai que entra o AccessKey e ClientId."

De acordo com uma resposta no fórum "Engenharia de Software" (em inglês) do grupo StackExchange (<http://softwareengineering.stackexchange.com/a/219041>), isso que você tentou fazer para bloquear sua API de "acesso indevido por apps terceiros" é IMPOSSÍVEL de se conseguir, em resumo devido à possibilidade de realização de engenharia reversa do app, de modo que qualquer pessoa com suficiente habilidade técnica e suficiente interesse poderia descobrir exatamente o que você colocou no código e arquivos de configuração do seu app, e assim efetuar a mesma ação quando acessando sua API, que no final pensaria que está recebendo uma requisição do seu app, quando na verdade a requisição foi proveniente de uma "cópia" do seu app (ou pelo menos da requisição que seu app faz ao seu serviço ou web api).

Fora isso, gostei muito do artigo. Bastante abrangente e completo. Estarei usando como base para a criação de um serviço de autenticação/autorização em um sistema que estou desenvolvendo com ASP.NET Web Api. Obrigado por compartilhar.

[Responder](#)



**rbento** says:

[22 de fevereiro de 2017 at 10:24](#)

Entendo e concordo, a partir do momento que se descobre o secret key é impossível de evitar, e isso acontece com todas as API uma exemplo claro é a api do Facebook caso você descubra o Secret Key e Access Key você terá todas as informações necessárias para acessar a API.

Obrigado pela visita

Abs.,

[Responder](#)



**MARCOS DONI** says:

[6 de março de 2017 at 11:46](#)

"Então imagine o cenário no qual nossa API é acessado por um APP que construirmos, temos que garantir que somente nosso APP seja autorizado a acessar esses recursos e é ai que entra o AccessKey e ClientId."

Um invasor que faça um ataque Man in the middle, não poderia obter AccessKey e ClientId no momento do envio?

[Responder](#)



**rbento** says:

[7 de março de 2017 at 11:29](#)

Sim ele poderia descobrir o accessKey e o clientId, uma maneira de evitar isso seria criptografar essas dados de envio com um hash utilizando um salt para a variação do hash e no servidor web fazer a comparação do hash. Acho que isso ajudaria.

Abs.,

[Responder](#)



**Sérgio** says:

[7 de março de 2017 at 09:38](#)

Caro Rafael,

Você conseguiu corrigir o erro? Uninstall "System.IdentityModel.Tokens.Jwt"  
Install-Package System.IdentityModel.Tokens.Jwt -Version 4.0.2.206221351

[Responder](#)



**rbento** says:

[7 de março de 2017 at 11:38](#)

Oi Sergio,

Corrigi sim e muito obrigado pela informação também fiz a alteração no post para adicionar o pacote com a referencia a esta versão.

abs.,

[Responder](#)



**Sérgio** says:

[7 de março de 2017 at 09:46](#)

Rafael, abaixo o erro

Error CS1503 Argument 6: cannot convert from  
'Thinktecture.IdentityModel.Tokens.HmacSigningCredentials' to  
'Microsoft.IdentityModel.Tokens.SigningCredentials'

[Responder](#)



**rbento** says:

[7 de março de 2017 at 11:41](#)

Oi Sergio,

Para ajustar esse erro temos que instalar o pacote System.IdentityModel.Tokens.Jwt na sua versão 4.0.2.206221351, como você mesmo mencionou.

Também fiz uma correção no post para assim referenciar esse pacote na sua versão correta.

Abs.,

[Responder](#)



**Schmidt** says:

[22 de abril de 2017 at 10:04](#)

Bom dia Rafael.

Eu implementei o servidor de autenticação, e coloquei minha aplicação para se logar e consumir o Json... Pelo Postman consigo passar o "Authorization" para meu controller que esta com a anotação [Authorize], funciona tudo perfeitamente.... porém não encontrei uma forma de passar esse header em um redirecttoaction (um redirecionamento de pagina) chamando uma nova ação de um controller. Será que seria possível fazê-lo????

[Responder](#)



**rbento** says:

[24 de abril de 2017 at 23:48](#)

Oi Schmidt,

É possível sim fazer isso, você antes do chamar o redirecttoaction deve usar o objeto response e adicionar o Header Authorization e o seu valor.

Assim deve funcionar de boa =]

Abs e obrigado pela visita

[Responder](#)



**Schmidt** says:

[25 de abril de 2017 at 08:49](#)

Bom dia Rafael...

Então, eu tentei dessa forma, porém pelo que entendi, quando você redireciona uma action, ela recarrega, e o que foi passado nos headers se perdem... não consegui fazer funcionar, se não fosse pedir de mais, se desse para você testar no seu projeto... só para ver se é algo errado no meu... 😊

Fico grato!!!

[Responder](#)



**Alexandre** says:  
[7 de junho de 2017 at 20:34](#)

Bom dia Rafael,

Fiz esta implementação e funcionou tudo normal pelo postman. Porém, quando implementei no client usando jquery Ajax, não consegui carregar no context os dados de clientId, username e password.

Segue minha chamada no jquery:

```
$ajax({  
    global: false,  
    type: 'POST',  
    url: HelperJS.getBaseUrl('oauth2/token'),  
    cache: false,  
    headers: {  
        'Content-type': 'application/x-www-form-urlencoded'  
    },  
    data: 'username=' + $scope.userName + '&password=' + $scope.password,  
    success: function (data) {  
        fnSuccess(data);  
    },  
    error: function (jqXHR, textStatus, errorThrown) {  
        HelperJS.showError  
    }  
});
```

Se conseguir me ajudar eu agradeço.

[Responder](#)



**rbento** says:  
[9 de junho de 2017 at 13:08](#)

Oi Alexandre,

Você deve estar esquecendo de passar algum parâmetro, senão me engano o grant\_type=password

Acredito que esse seja o problema.

abs.,

[Responder](#)



**Wallace** says:  
[9 de junho de 2017 at 17:45](#)

Boa tarde, é possível realizar esse Oauth junto a autenticação do facebook?

[Responder](#)



**rbento** says:

▼ ▼ 13 de junho de 2017 at 00:29

Oi Wallace, não a própria autenticação do facebook é via oauth, seria como se tivesse sobreposição de autenticação.  
o próprio facebook já usa esse modelo de oauth para a sua autenticação via login e a geração de token seria também diferente já que você deveria saber qual foi a criptografia usada para gerar token iguais.

Abs

[Responder](#)



**Tiago** says:

19 de junho de 2017 at 09:50

Obrigado por compartilhar!!

Você teria um exemplo de como armazenar e recuperar o token do cookie?

[Responder](#)



**rbento** says:

20 de junho de 2017 at 16:19

Oi Tiago,

Obrigado pela visita! Não tenho um exemplo não, mas não seria tão complicado, basta no retorno gravar o token no cookie em javascript.

Talvez esse post do stackoverflow possa te dar um caminho.

<https://stackoverflow.com/questions/4825683/how-do-i-create-and-read-a-value-from-cookie>

Abs.,

[Responder](#)



**Andre** says:

7 de julho de 2017 at 18:12

Fala rafael!

Muito bom seu post. Estou com uma duvida, por que devo usar o formato x-www-form-urlencoded no Postman e se teria como eu passar no formato application/json?

[Responder](#)



**rbento** says:

12 de julho de 2017 at 01:42

Olá André, pelo meu exemplo neste post eu demonstrei utilizando x-www-form-urlencoded porém você pode utilizar o application/json,  
você deve modicar a forma que você lê os inputs para saber como buscar via json.

Abs.,

[Responder](#)**Pedro** says:

28 de julho de 2017 at 08:33

Caso meu 'authorization-server' e meu 'resource-server' não sejam os mesmos, eu teria que 'duplicar' a implementação em ambos?

[Responder](#)**rbento** says:

3 de setembro de 2017 at 21:51

Oi Pedro,

Não, cada um vai fazer o papel que lhe cabe, por exemplo no caso do authorization-server, você pegará a implementação de geração de token e validação do token no resource-server ele somente será capaz de enviar um token para o authorization-server e saber se está valido ou não.

Abs

[Responder](#)**Nilton Morais** says:

28 de julho de 2017 at 12:19

Muito bom e explicado!

Deu tudo certo aqui.

Porém agora quero fazer a verificação com o banco de dados. Tem como fazer um exemplo usando essa base que criamos, porém agora buscando os dados no banco e salvando o token no banco também?

Abraço!

[Responder](#)**rbento** says:

3 de setembro de 2017 at 21:46

Fala Nilton,

Sobre o banco de dados não tenho um tutorial para esse exemplo, porém não faz muito sentido gravar um token no banco de dados já que o mesmo expira.

Abs.,

[Responder](#)**Jonnison Lima Ferreira** says:

29 de agosto de 2019 at 19:16

Estou tentando seguir o tutorial, gero o token porém não consigo autenticar, é possível criar uma

solução híbrida entre autenticação por sessão e token?

[Responder](#)



**Jonnison Lima Ferreira** says:

[29 de agosto de 2019 at 19:17](#)

Estou tentando seguir o tutorial, gero o token porém não consigo autenticar, o token é gerado normalmente, consigo validar no jwt.io mas na aplicação sempre diz que nao tenho acesso, alguma ideia do q pode ser?

é possivel criar uma solução híbrida entre autenticação por sessão e token?

[Responder](#)



**rbento** says:

[13 de setembro de 2019 at 13:15](#)

Olá Jonnison, muito obrigado pela visita!

Sim, com a técnica do refresh token, é possível simular uma sessão.

Sobre o problema do acesso, pode ser o scope entre o client e o token podem estar diferente, o segredo entre os dois devem está iguais também.

Já verificou essas partes?

[Responder](#)



**Jonnison Lima Ferreira** says:

[19 de setembro de 2019 at 11:38](#)

Sou novo no C#, como verifco isso?

[Responder](#)



**Fabio Rattis** says:

[23 de setembro de 2019 at 16:39](#)

tem 2 lugares no seu codigo,

1 codigo q está gerando o TOKEN, e ele usa um SECRET para isso

e deve ter outro codigo para validar esse token, e deve ter o mesmo SECRET  
se nao nao funciona corretante.

[Responder](#)



**Alexandre** says:

[27 de setembro de 2019 at 08:20](#)

The solution for Severity cannot convert from  
'Thinktecture.IdentityModel.Tokens.HmacSigningCredentials' to  
'Microsoft.IdentityModel.Tokens.SigningCredentials'

```
var securityKey = new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey(keyByteArray);
var signingCredentials = new Microsoft.IdentityModel.Tokens.SigningCredentials(
```

```
securityKey, SecurityAlgorithms.HmacSha256Signature);
```

```
var token = new JwtSecurityToken(_issuer, client_id, data.Identity.Claims, issued.Value.UtcDateTime,  
expires.Value.UtcDateTime, signingCredentials);
```

[Responder](#)



**Pedro Henrique** says:

[30 de outubro de 2019 at 11:39](#)

Ótimo post sobre o assunto, realmente me salvou no momento de autenticar com json webtoken.

[Responder](#)



**rbento** says:

[30 de outubro de 2019 at 12:57](#)

Muito obrigado Pedro!

Estou escrevendo uma atualização com NET Core e Identity Server

Abs e obrigado pela visita

[Responder](#)

Pingback: [\[Updated\] Implementando OAuth JSON Web Token com OWIN no ASP.NET Web API - \(ASP.NET Identity + Identity Server v4\) - Rafael Cruz](#)

## 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.

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