To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

Faça login em medium.com com o Google

Fernando Passaia
fernandopassaia@futuradata.com.br

F
Fernando Passaia
fernando.passaia@futureofmedia.hu

ASP.NET Core 2.
APIs utilizando J
Tokens)

Renato Groffe Follow
Oct 16, 2017 · 8 min read

Controlar o acesso a APIs REST é uma necessidade comum aos mais variados tipos de projetos. Esta preocupação se torna ainda maior em APIs expostas na Internet, nas quais a utilização indevida de um recurso pode resultar em graves problemas (como perda de informações confidenciais, prejuízos financeiros e ações legais, por exemplo).

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

O
cc a
APIs. Esta abordagem se vale de **tokens** criptografados para assim liberar a utilização de recursos de uma API, sendo que tal técnica recebeu o nome de **Bearer Authentication.**

Um token baseado na especificação **JWT** é formado por 3 partes, cada uma delas separadas por ponto:

xxxxx.yyyyy.zzzzz

As diferentes partes envolvidas são

1. **Header**: cabeçalho contendo o tipo do token (**JWT**) e o mecanismo de criptografia utilizado (**HMAC** ou **RSA**);
2. **Payload**: conjunto de claims associadas ao usuário referenciado pelo token;
3. **Signature**: assinatura empregada na validação do token, dependendo para isto de uma chave controlada pela aplicação que faz uso deste mecanismo de autenticação.

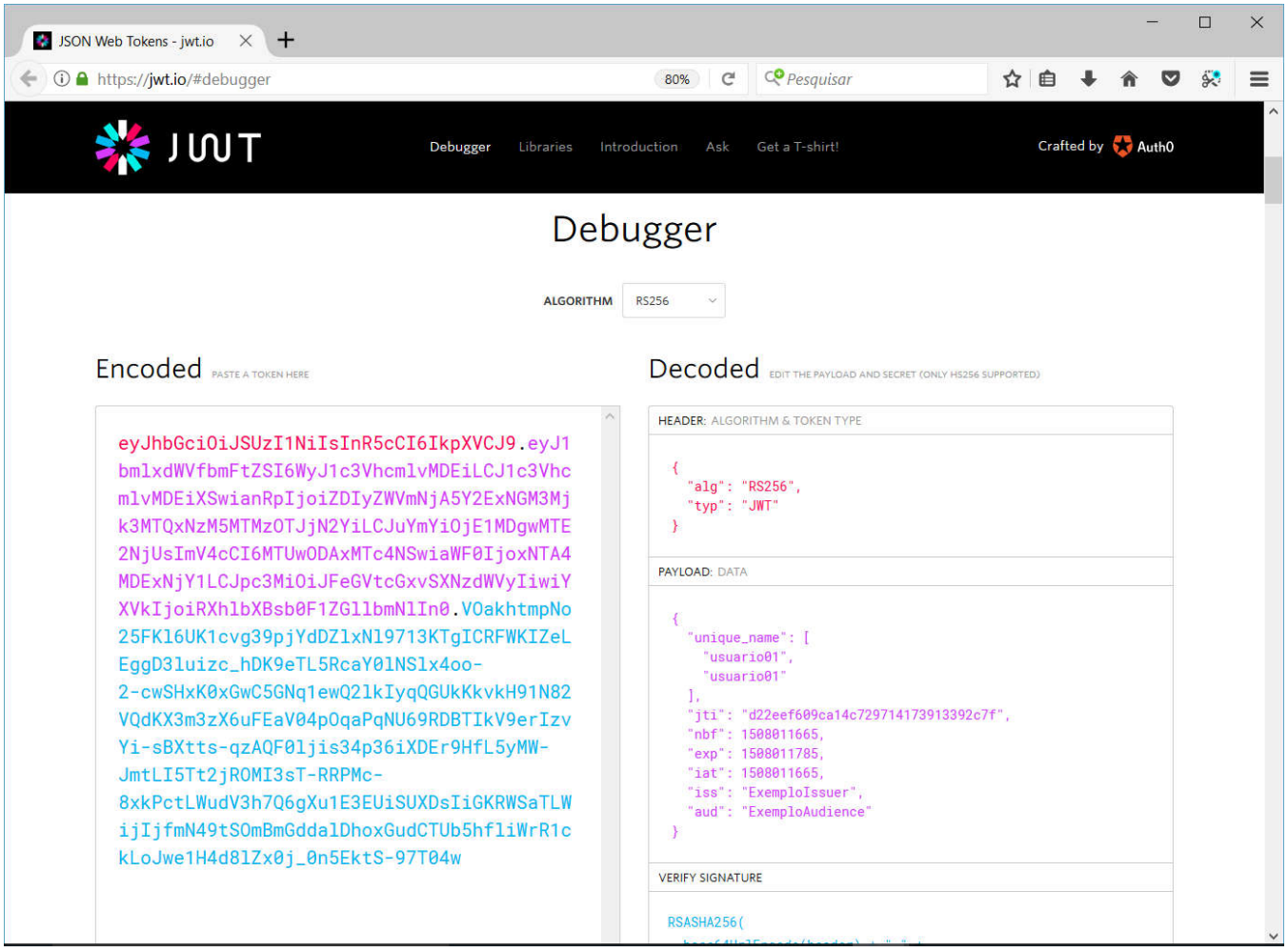
O valor a seguir representa um padrão de token gerado com base no padrão **RSA**:

> ***eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmlxdWVfbmFtZSI6WyJ1c3VhcmlvMDEiLCJ1c3VhcmlvMDEiXSwianRpIjoiZDIyZWVmNjA5Y2ExNGM3Mjk3MTQxNzM5MTMzOTJjN2YiLCJuYmYiOjE1MDgwMTE2NjUsImV4cCI6MTUwODAxMTc4NSwiaWF0IjoxNTA4MDExNjY1LCJpc3Mi***

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

> *O* *t*
> *k* *c*
> *_hDK9eTL5RcaY0lNSlx4oo-*
> *2-cwSHxK0xGwC5GNq1ewQ2lkIyqQGUkKkvkH91N82VQdKX3m3zX6u*
> *FEaV04pOqaPqNU69RDBTIkV9erIzvYi-sBXtts-*
> *qzAQF0ljis34p36iXDEr9HfL5yMW-JmtLI5Tt2jROMI3sT-RRPMc-*
> *8xkPctLWudV3h7Q6gXu1E3EUiSUXDsIiGKRWSaTLWijIjfmN49tSOmB*
> *mGddalDhoxGudCTUb5hfliWrR1ckLoJwe1H4d8lZx0j_0n5EktS-*
> *97T04w*

No próprio site da especificação **JWT** existe um **debbuger** que permite analisar este conteúdo:



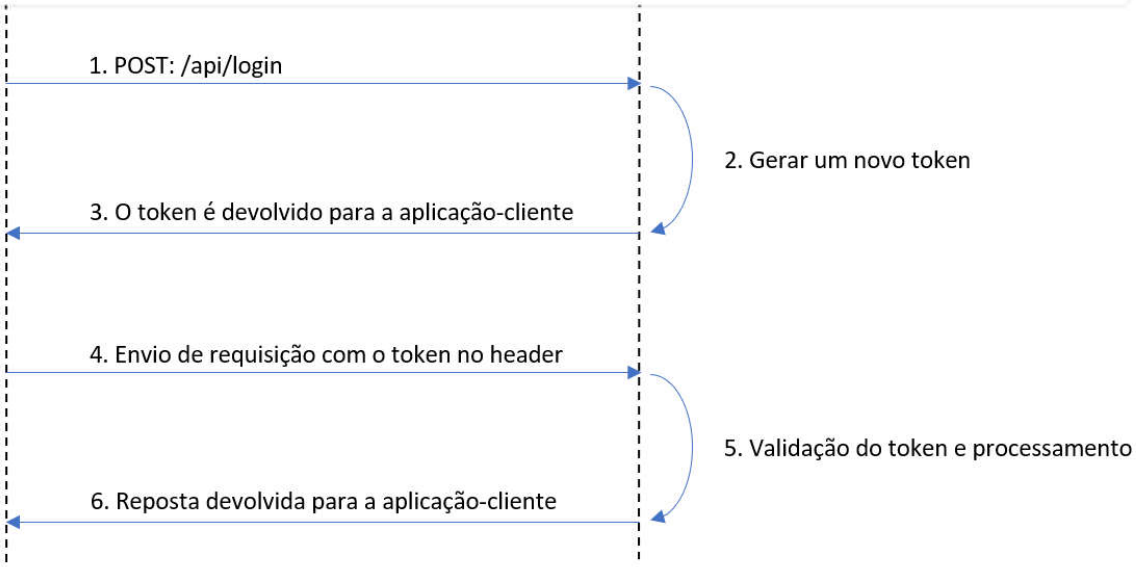
E
er

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy. ×

1. Uma requisição HTTP do tipo **POST** é enviada a um sistema Web, podendo conter em seu corpo informações como usuário e/ou chaves/senhas de acesso;
2. Caso as credenciais sejam válidas um token baseado na especificação JWT é gerado;
3. O token é então devolvido à aplicação que encaminhou a requisição. Importante destacar que essa estrutura (token) possui uma validade, expirando após um período de tempo estabelecido previamente. Todo este processo é **stateless**, ou seja, ocorre sem o armazenamento de estados na memória do servidor. A opção por um token dispensa também múltiplas pesquisas a um repositório com credenciais de acesso (um banco de dados, por exemplo), contribuindo assim para uma maior performance no uso de APIs REST;
4. Requisições a recursos de APIs REST da aplicação Web conterão no header o token obtido anteriormente;
5. O token informado na seção **Authorization** do header é então analisado.
6. Caso seja válido, o acesso ao recurso é então liberado. Uma resposta é devolvida então como resultado de um processamento na API REST.

Na imagem a seguir está uma representação esquemática deste processo:

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.



A nova plataforma Web da Microsoft também suporta **JWT** como mecanismo de autenticação em APIs REST. Nas próximas seções será abordada esta prática, com isto acontecendo através de um exemplo implementado com base no **ASP.NET Core 2.0**.

. . .

Alguns detalhes sobre a aplicação de exemplo

A API apresentada neste artigo fará a conversão de alturas em pés (unidade comumente utilizada na aviação) para o equivalente em metros. Já descrevi inclusive a implementação desta funcionalidade em outro artigo:

ASP.NET Core: APIs REST na nuvem com Docker e Azure Web App

Além do **ASP.NET Core 2.0** serão utilizados também o **SQL Server** e o

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

D ...s (I ...s autenticação detalhado mais adiante.

Para este post tomei ainda como base um exemplo em **ASP.NET Core 1.0** disponibilizado pela própria Microsoft:

How to achieve a bearer token authentication and authorization in ASP.NET Core

> *E por falar em **tecnologias Microsoft**, não deixem também de acompanhar o **Azure Tech Nights**, que acontecerá entre os dias **20 e 28 de Agosto de 2018**. Será um evento **NOTURNO, ONLINE e GRATUITO** promovido pelo **Canal .NET**, com apresentações focadas no **Microsoft Azure** e cobrindo temas como **microsserviços, Inteligência Artificial, desenvolvimento Web, bancos de dados, NoSQL, Docker, Kubernetes** e muito mais.*
>
> *Entre os palestrantes teremos **MVPs Microsoft, MTACs** e **Especialistas de Mercado**.*
>
> *Para efetuar a inscrição acessem este **link**.*
>
> *A grade com as palestras e outras informações podem ser encontradas no site oficial do **Azure Tech Nights**.*

. . .

Criando a base de dados

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

dbo.Users, na qual constarão IDs de usuários e suas respectivas chaves de acesso:

```
USE ExemploJWT
GO

CREATE TABLE dbo.Users(
        UserID varchar(20) NOT NULL,
        AccessKey varchar(32) NOT NULL,
        CONSTRAINT PK_Clientes PRIMARY KEY (UserID)
)
GO


INSERT INTO dbo.Users
           (UserID
           ,AccessKey)
     VALUES
           ('usuario01'
           ,'94be650011cf412ca906fc335f615cdc')

INSERT INTO dbo.Users
           (UserID
           ,AccessKey)
     VALUES
           ('usuario02'
           ,'531fd5b19d58438da0fd9afface43b3c')
```

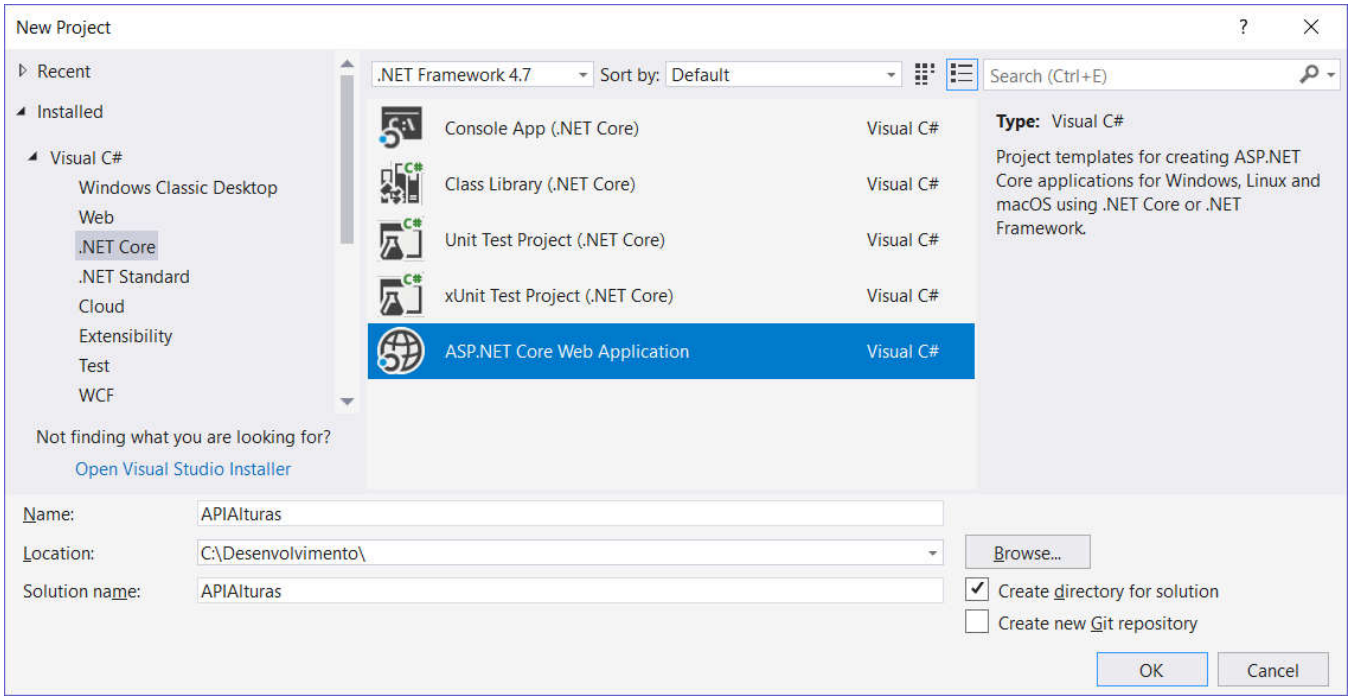
Scripts-Users.sql hosted with ❤ by **GitHub** **view raw**

. . .

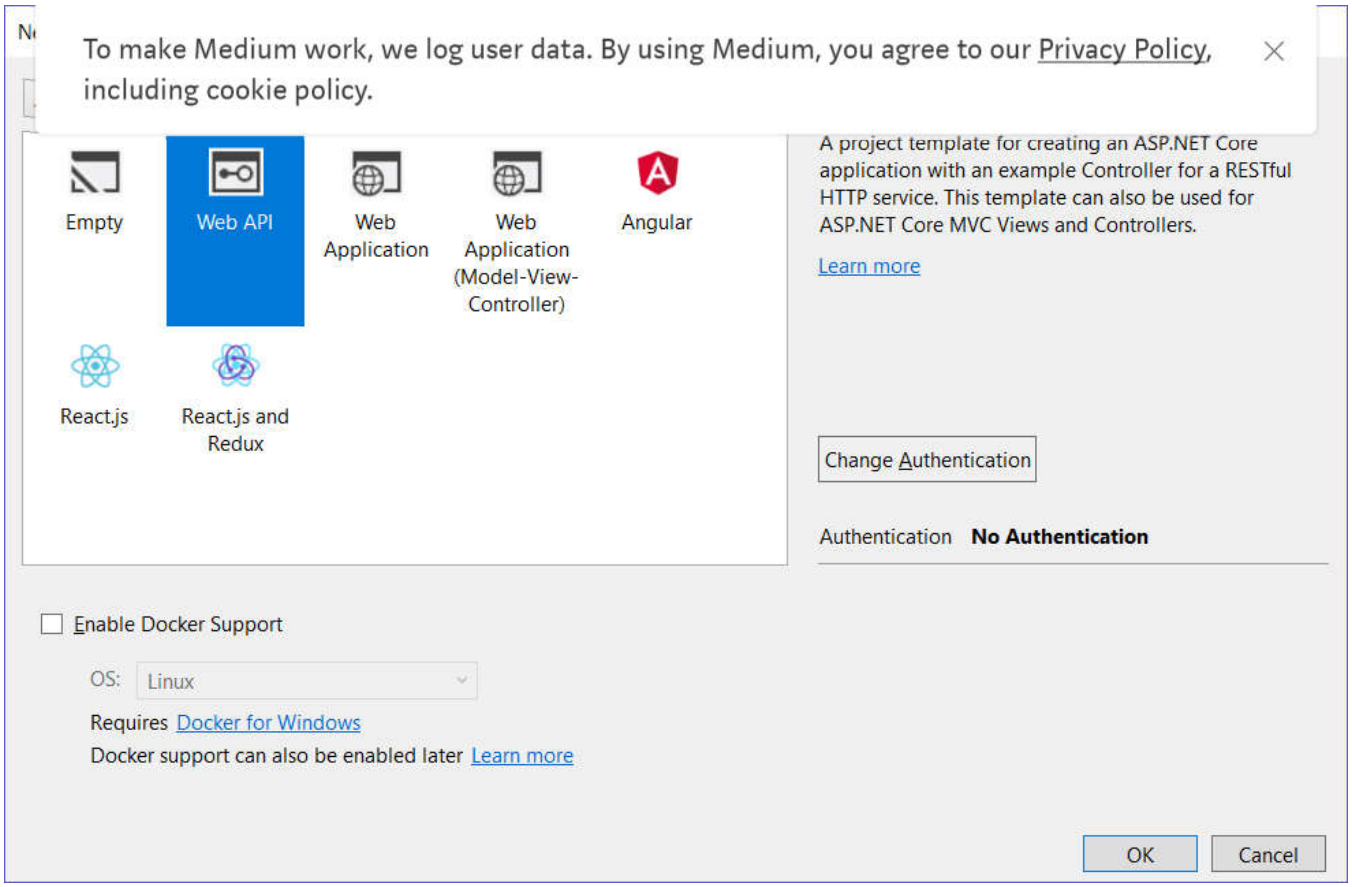
Implementando a aplicação de exemplo

Será criado um projeto do tipo **ASP.NET Core Web Application** chamado

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.



Selecionar para isto o template **Web API**, além das opções **.NET Core** e **ASP.NET Core 2.0**:



No arquivo **appsettings.json** serão incluídas a string de conexão para acesso à base **ExemploJWT**, além de configurações para a geração do token (Audience, Issuer e tempo de duração em segundos):

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

```
  },
  "TokenConfigurations": {
    "Audience": "ExemploAudience",
    "Issuer": "ExemploIssuer",
    "Seconds": 120
  },
  "Logging": {
    "IncludeScopes": false,
    "Debug": {
      "LogLevel": {
        "Default": "Warning"
      }
    },
    "Console": {
      "LogLevel": {
        "Default": "Warning"
      }
    }
  }
}
```

appsettings.json hosted with ❤ by GitHub view raw

A próxima listagem traz a implementação das seguintes estruturas:

- **User**: tipo empregado na manipulação de credenciais de usuários;
- **TokenConfigurations**: classe que conterá configurações (**Audience**, **Issuer** - emissor, **Seconds** - tempo de validade em segundos) empregadas na geração de tokens. Estas definições serão obtidas a partir do arquivo **appsettings.json.**

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

```
    {
        public string UserID { get; set; }
        public string AccessKey { get; set; }
    }

    public class TokenConfigurations
    {
        public string Audience { get; set; }
        public string Issuer { get; set; }
        public int Seconds { get; set; }
    }
}
```

User-TokenConfigurations.cs hosted with ❤ by GitHub view raw

Já a classe **UsersDAO** acessará a base **ExemploJWT** e fará uso do **Dapper**, retornando através do método **Find** instâncias do tipo **User** que conterão o ID de um usuário e sua respectiva chave de acesso:

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

```
4
5   namespace APIAlturas
6   {
7       public class UsersDAO
8       {
9           private IConfiguration _configuration;
10
11          public UsersDAO(IConfiguration configuration)
12          {
13              _configuration = configuration;
14          }
15
16          public User Find(string userID)
17          {
18              using (SqlConnection conexao = new SqlConnection(
19                  _configuration.GetConnectionString("ExemploJWT")))
20              {
21                  return conexao.QueryFirstOrDefault<User>(
22                      "SELECT UserID, AccessKey " +
23                      "FROM dbo.Users " +
24                      "WHERE UserID = @UserID", new { UserID = userID });
25              }
26          }
27      }
28  }
```

UsersDAO.cs hosted with ❤ by GitHub view raw

No tipo **SigningConfigurations** foram definidos:

- A propriedade **Key**, à qual será vinculada uma instância da classe **SecurityKey** (namespace **Microsoft.IdentityModel.Tokens**) armazenando a chave de criptografia utilizada na criação de tokens;
- A propriedade **SigningCredentials**, que receberá um objeto baseado em uma classe também chamada **SigningCredentials** (namespace

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

assinaturas digitais para tokens;

- Um construtor responsável pela inicialização das propriedades **Key** e **SigningCredentials**. Este elemento fará uso para isto dos tipos **RSACryptoServiceProvider** (namespace **System.Security.Cryptography**), **RsaSecurityKey** (namespace **Microsoft.IdentityModel.Tokens**) e **SecurityAlgorithms** (namespace **Microsoft.IdentityModel.Tokens**), determinando assim o uso do padrão **RSA** como algoritmo de criptografia usado na produção de tokens.

```
using System.Security.Cryptography;
using Microsoft.IdentityModel.Tokens;

namespace APIAlturas
{
    public class SigningConfigurations
    {
        public SecurityKey Key { get; }
        public SigningCredentials SigningCredentials { get; }

        public SigningConfigurations()
        {
            using (var provider = new RSACryptoServiceProvider(2048))
            {
                Key = new RsaSecurityKey(provider.ExportParameters(true));
            }

            SigningCredentials = new SigningCredentials(
                Key, SecurityAlgorithms.RsaSha256Signature);
        }
    }
}
```

SigningConfigurations.cs hosted with ❤ by GitHub view raw

O
aj

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

- Uma referência de **TokenConfigurations** será criada a partir do objeto vinculado à propriedade **Configuration** e do conteúdo definido na seção de mesmo nome no arquivo **appsettings.json**;
- Instâncias dos tipos **SigningConfigurations** e **TokenConfigurations** serão configuradas via método **AddSingleton**, de forma que uma única referência das mesmas seja empregada durante todo o tempo em que a aplicação permanecer em execução. Quanto a **UsersDAO**, o método **AddTransient** determina que referências desta classe sejam geradas toda vez que uma dependência for encontrada;
- Em seguida serão invocados os métodos **AddAuthentication** e **AddJwtBearer**. A chamada a **AddAuthentication** especificará os schemas utilizados para a autenticação do tipo **Bearer**. Já em **AddJwtBearer** serão definidas configurações como a chave e o algoritmo de criptografia utilizados, a necessidade de analisar se um token ainda é válido e o tempo de tolerância para expiração de um token (zero, no caso desta aplicação de testes);
- A chamada ao método **AddAuthorization** ativará o uso de tokens com o intuito de autorizar ou não o acesso a recursos da aplicação de testes.

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

```
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Options;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication.JwtBearer;

namespace APIAlturas
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddTransient<UsersDAO>();

            var signingConfigurations = new SigningConfigurations();
            services.AddSingleton(signingConfigurations);

            var tokenConfigurations = new TokenConfigurations();
            new ConfigureFromConfigurationOptions<TokenConfigurations>(
                Configuration.GetSection("TokenConfigurations"))
                    .Configure(tokenConfigurations);
            services.AddSingleton(tokenConfigurations);


            services.AddAuthentication(authOptions =>
            {
                authOptions.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
                authOptions.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            }).AddJwtBearer(bearerOptions =>
            {
                var paramsValidation = bearerOptions.TokenValidationParameters;
                paramsValidation.IssuerSigningKey = signingConfigurations.Key;
                paramsValidation.ValidAudience = tokenConfigurations.Audience;
                paramsValidation.ValidIssuer = tokenConfigurations.Issuer;
```

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

válidas, produzirá como resultado um token com tempo de duração de **2 minutos** (**120 segundos**, valor configurado anteriormente no arquivo **appsettings.json**):

- O método **Post** receberá requisições HTTP do tipo **POST**, tendo sido marcado com o atributo **AllowAnonymous** para assim possibilitar o acesso de usuários não-autenticados;
- As instâncias dos tipos **UsersDAO**, **SigningConfigurations** e **TokenConfigurations** foram marcadas com o atributo **FromServices** no método **Post**, o que indica que as mesmas serão resolvidas via mecanismo de injeção de dependências do ASP.NET Core;
- O parâmetro **usuario** foi marcado com o atributo **FromBody**, correspondendo às credenciais (ID do usuário + chave de acesso) que serão enviadas no corpo de uma requisição. As informações desta referência (**usuario**) serão então comparadas com o retorno produzido pela instância do tipo **UsersDAO**, determinando assim a validade do usuário e da chave de acesso em questão;
- Em se tratando de credenciais de um usuário existente claims serão geradas, o período de expiração calculado e um token criado por meio de uma instância do tipo **JwtSecurityTokenHandler** (namespace **System.IdentityModel.Tokens.Jwt**). Este último elemento é então transformado em uma string por meio do método **WriteToken** e, finalmente, devolvido como retorno da Action **Post** (juntamente com outras informações como horário de geração e expiração do token);
- Se o usuário for inválido um objeto então será devolvido, indicando que

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Principal;
using Microsoft.IdentityModel.Tokens;

namespace APIAlturas.Controllers
{
    [Route("api/[controller]")]
    public class LoginController : Controller
    {
        [AllowAnonymous]
        [HttpPost]
        public object Post(
            [FromBody]User usuario,
            [FromServices]UsersDAO usersDAO,
            [FromServices]SigningConfigurations signingConfigurations,
            [FromServices]TokenConfigurations tokenConfigurations)
        {
            bool credenciaisValidas = false;
            if (usuario != null && !String.IsNullOrWhiteSpace(usuario.UserID))
            {
                var usuarioBase = usersDAO.Find(usuario.UserID);
                credenciaisValidas = (usuarioBase != null &&
                    usuario.UserID == usuarioBase.UserID &&
                    usuario.AccessKey == usuarioBase.AccessKey);
            }

            if (credenciaisValidas)
            {
                ClaimsIdentity identity = new ClaimsIdentity(
                    new GenericIdentity(usuario.UserID, "Login"),
                    new[] {
                        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString("N")),
                        new Claim(JwtRegisteredClaimNames.UniqueName, usuario.UserID)
                    }
                );

                DateTime dataCriacao = DateTime.Now;
                DateTime dataExpiracao = dataCriacao +
                    TimeSpan.FromSeconds(tokenConfigurations.Seconds);

```

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

- A Action **Get** foi marcada com o atributo **Authorize** e executará o cálculo esperado;
- O atributo **Authorize** está recebendo como parâmetro o valor **"Bearer"**, o que indica o uso de **Bearer Authentication**. Requisições recebidas pelo método **Get** apenas serão processadas se contiverem em seu header um token válido.

```
using System;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;

namespace APIAlturas.Controllers
{
    [Route("api/[controller]")]
    public class ConversorAlturasController : Controller
    {
        [Authorize("Bearer")]
        [HttpGet("PesMetros/{alturaPes}")]
        public object Get(double alturaPes)
        {
            return new
            {
                AlturaPes = alturaPes,
                AlturaMetros = Math.Round(alturaPes * 0.3048, 4)
            };
        }
    }
}
```

ConversorAlturasController.cs hosted with ❤ by **GitHub** **view raw**

O projeto descrito nesta seção já foi disponibilizado no **GitHub**, caso deseje

ef

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

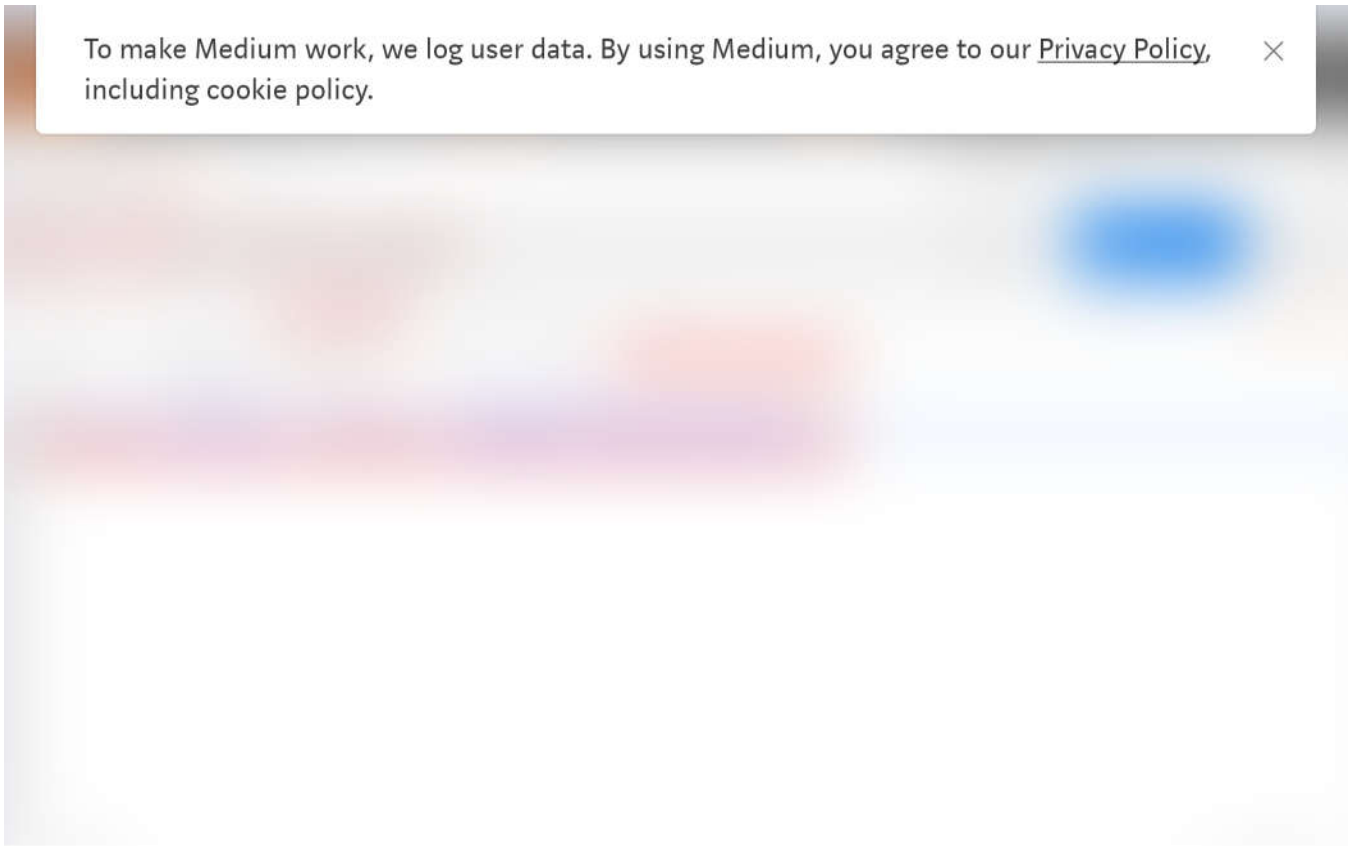
https://github.com/renatogroffe/ASPNETCore2_JWT

. . .

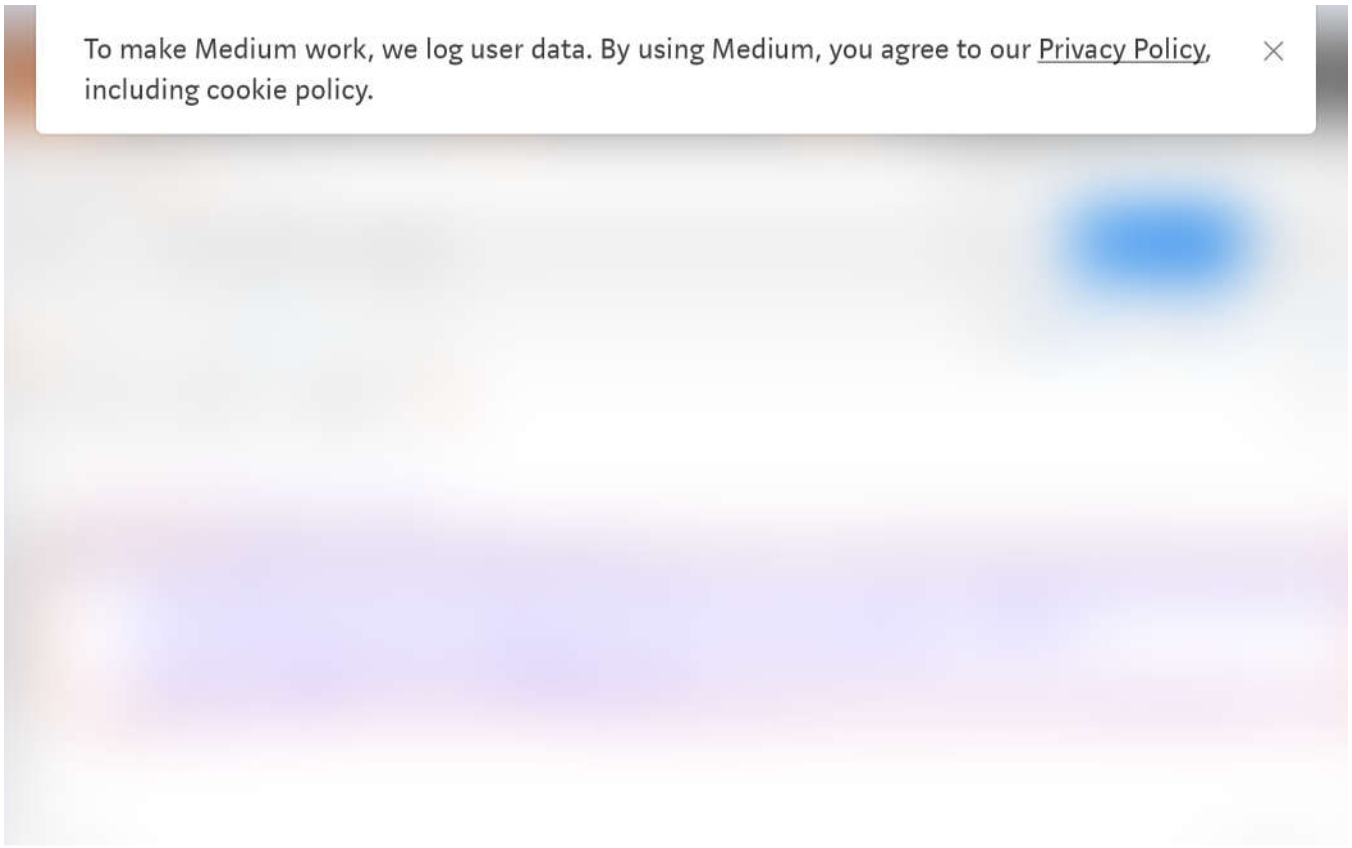
Testes via Postman

Com o projeto **APIAlturas** em execução serão realizados testes via **Postman**, um utilitário gratuito e multiplataforma bastante popular entre desenvolvedores de APIs REST.

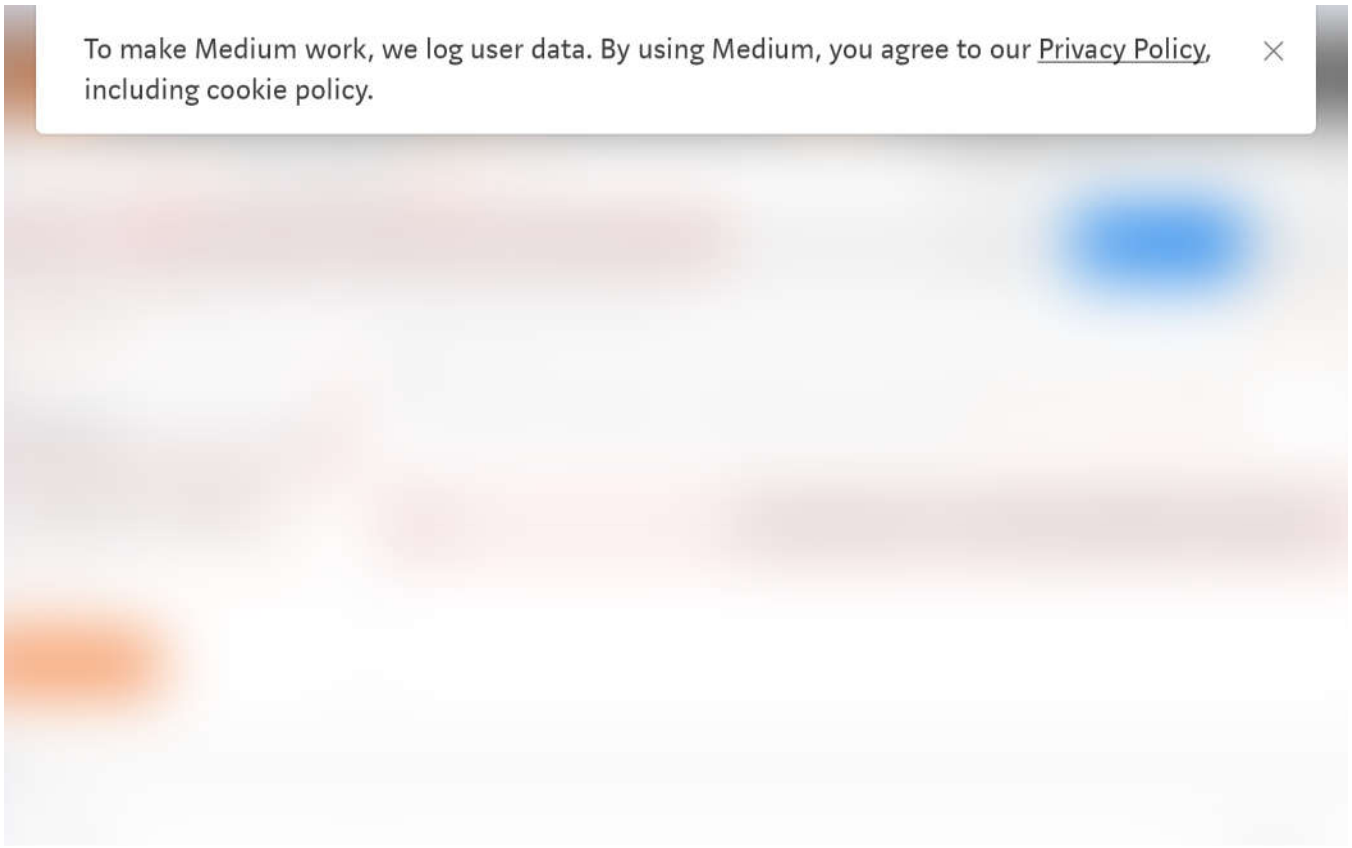
O primeiro passo consiste no envio de uma requisição HTTP do tipo **POST** para a obtenção de um token. Isto será feito através da URL http://localhost:56435/api/login, informando no corpo da solicitação uma string JSON contendo o usuário (**userID**) e a chave de acesso (**accessKey**):



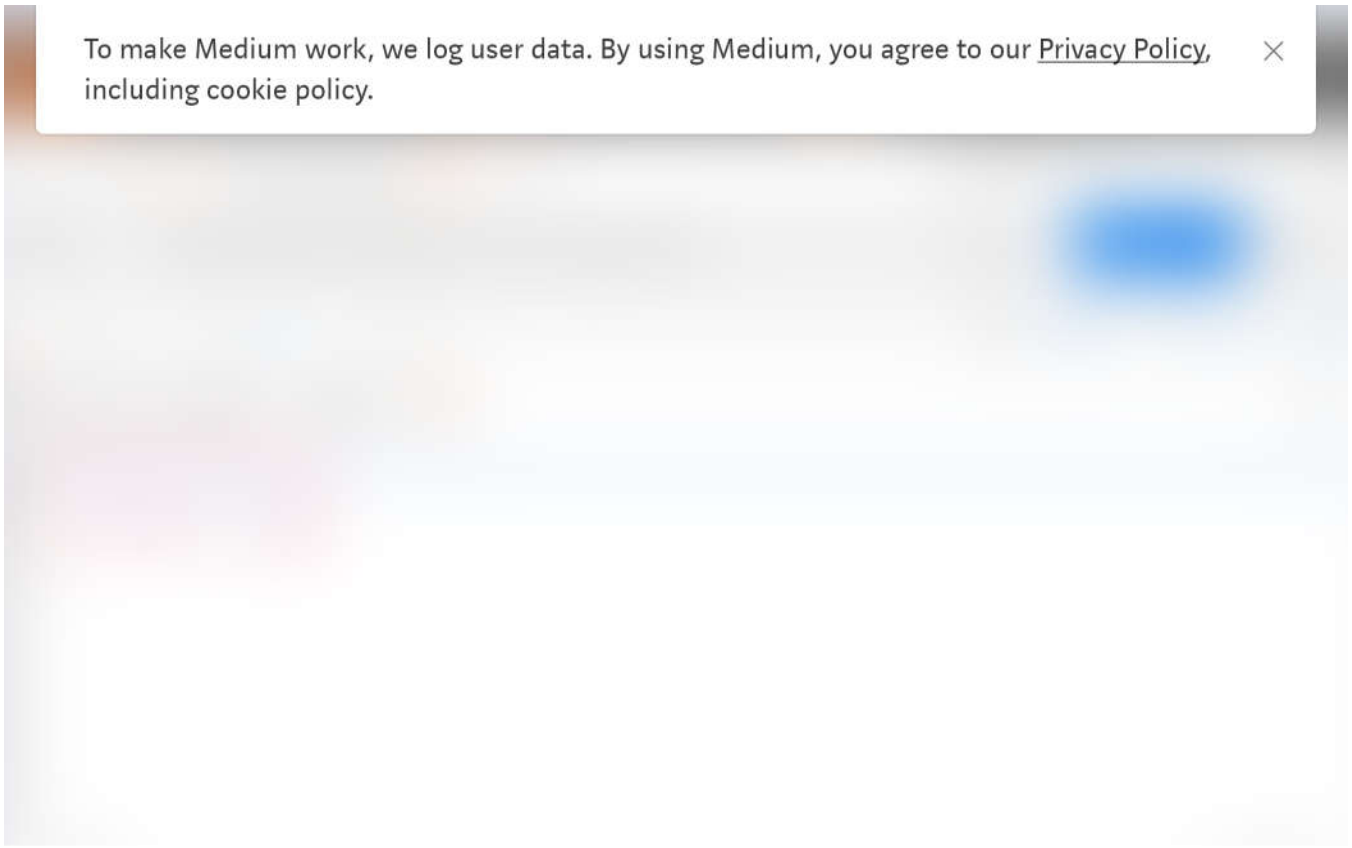
Ao acionar o botão **Send** será retornada uma string JSON contendo informações como a duração e o token de autenticação (este último destacado em vermelho):



Uma requisição para a conversão de uma altura de **100 pés** será enviada agora. Esta solicitação do tipo **GET** terá como URL o valor http://localhost:56435/api/conversoralturas/pesmetros/100, fazendo uso do token de autenticação gerado no passo anterior:



Esta ação produzirá como resultado um JSON contendo a altura equivalente em metros (**30,48 m**):



. . .

Conclusão

Embora este artigo demonstre o uso de uma solução customizada para controle de usuários, outras soluções também podem ser empregadas em conjunto com **JWT (JSON Web Tokens)**. Exemplos disto são tecnologias como **ASP.NET Core Identity** e **AD (Active Directory)**.

Em um próximo artigo abordarei o consumo de **APIs REST** que utilizam **JWT**, com isto acontecendo a partir de aplicações baseadas no **.NET Core.**

Não deixe também de acompanhar o post a seguir, em que venho agrupando todos os conteúdos que tenho produzido sobre **.NET Core 2.0** e **ASP.NET Core 2.0:**

To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.

Referências

ASP.NET Core - Documentation

Conteúdos gratuitos sobre ASP.NET Core, .NET Core e C# 7.0

JSON Web Tokens - jwt.io

JWT no ASP.NET Core - Standalone

Aspnet Dotnet Core Jwt Web Development Aspnetcore

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just $5/month. Upgrade

About Help Legal