



A <PROGRAMAÇÃO> PRECISA DE TI

JSON WEB TOKENS

Laboratório Web



JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.

JWT.IO allows you to decode, verify and generate JWT.

[LEARN MORE ABOUT JWT](#)

JSON Web Token

- É um padrão aberto que define uma forma de transmitir informação de forma segura entre várias partes utilizando objetos JSON
- Como funciona?
 - O servidor gera um token JWT específico para um cliente e armazena o mesmo na sessão
 - Este token é passado para o cliente (cookies)
 - A partir deste momento todos os pedidos efetuados ao servidor incluem o token para aceder a rotas protegidas
- O objetivo é autenticarmos utilizadores e controlar o acesso aos recursos

JSON Web Token

- É composto por 3 partes:
 - **Header:** objeto JSON que indica o tipo de token (JWT)
 - **Payload:** objeto JSON com os dados (claims) da entidade
 - **Signature:** string criada usando o segredo e os components anteriores, é usado para verificar se o token não foi comprometido

eyJhbGciOiJodHRwOi8vd3d3LnczLm9yZy8yMDA1LzA0L3htbGRzaWctbW9yZSNoYWJjLXNoYTI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW11IjoiaZGphcmRpbSI0dHA6Ly9zY2h1bWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWR1bnRpdHkvY2xhaW1zL2VtYWlsYWRkcml6cyI6ImRqYXJkaW1AZ21haWwuY29tIiwiaHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmevd3MvMjAwNS8wNS9pZGVudG10eS9jbGFpbXMvbmFtZS9uYW11IjoiaW1iZXhwIjozNzAzMTkwLjE0IiwiaWF0IjoxNjU0MjU0MDAwfQ.CIr_BJ9t41VA5x3DAToDggBnTaHqNq4aK8LFJ_0z0Fs

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "http://www.w3.org/2001/04/xmldsig-more#hmac-
sha256",
  "typ": "JWT"
}
```





PAYLOAD: DATA

```
{
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims
/name": "djardim",
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims
/emailaddress": "djardim@gmail.com",
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims
/nameidentifier": "1",
  "exp": 1647703190,
  "iss": "djardim",
  "aud": "https://localhost:7071/"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
    base64UrlEncode(header) + "." +
```

Instalar o pacote JSON Web Token

	Microsoft.AspNetCore.Authentication by Microsoft ASP.NET Core common types used by the various authentication middleware components.	2.2.0
	Microsoft.AspNetCore.Authentication.JwtBearer by Microsoft ASP.NET Core middleware that enables an application to receive an OpenID Connect bearer token.	6.0.3
	Microsoft.EntityFrameworkCore by Microsoft Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.	6.0.3
	MySQL.EntityFrameworkCore by Oracle MySQL.EntityFrameworkCore for Entity Framework.	6.0.0

Gerar um token JWT?

- Para gerar um token é necessária a seguinte informação:
 - O segredo do token
 - Os dados para gerar os claims do token
 - O tempo de expiração do token


```
appsettings.json  X
Schema: https://json.schemastore.org/appsettings.json
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    }
8    "Jwt": {
9      "Key": "thisismysecretkey",
10     "Issuer": "djardim",
11     "Audience": "https://localhost:7071/"
12   },
13   "AllowedHosts": "*"
14 }
```

Parâmetros
de
configuração
para o JWT

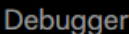

```
public string GenerateToken(string key, string issuer, string audience, UserDTO user)
{
    var claims = new[] {
        new Claim(ClaimTypes.Name, user.UserName),
        new Claim(ClaimTypes.Email, user.Email),
        new Claim(ClaimTypes.NameIdentifier, user.ID.ToString())
    };

    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));

    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256Signature);

    var tokenDescriptor = new JwtSecurityToken(
        issuer,
        audience,
        claims,
        expires: DateTime.Now.AddMinutes(EXPIRY_DURATION_MINUTES), signingCredentials: credentials);

    return new JwtSecurityTokenHandler().WriteToken(tokenDescriptor);
}
```



HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "http://www.w3.org/2001/04/xmldsig-more#hmac-
sha256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims
/name": "djardim",
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims
/emailaddress": "djardim@gmail.com",
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims
/nameidentifier": "1",
  "exp": 1647703190,
  "iss": "djardim",
  "aud": "https://localhost:7071/"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
    base64UrlEncode(header) + "." +
```

```
public bool IsTokenValid(string key, string issuer, string audience, string token)
{
    var mySecret = Encoding.UTF8.GetBytes(key);
    var mySecurityKey = new SymmetricSecurityKey(mySecret);
    var tokenHandler = new JwtSecurityTokenHandler();

    try
    {
        tokenHandler.ValidateToken(token,
            new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidIssuer = issuer,
                ValidAudience = audience,
                IssuerSigningKey = mySecurityKey,
            },
            // passed by reference
            out SecurityToken validatedToken);
    }
    catch
    {
        return false;
    }
    return true;
}
```

Como ativar os serviços para a Sessão e a Autenticação?

```
builder.Services.AddSession();
```

Adiciona os serviços necessários para a sessão

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options =>
{
    options.SaveToken = true;
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["Jwt:Issuer"],
        ValidAudience = builder.Configuration["Jwt:Audience"],
        IssuerSigningKey = new
            SymmetricSecurityKey
            (Encoding.UTF8.GetBytes
            (builder.Configuration["Jwt:Key"]))
    };
});
```

Como ativar os serviços para a Sessão e a Autenticação?

```
builder.Services.AddSession();
```

Adiciona os serviços necessários para a autenticação

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme) AddJwtBearer(options =>
{
    options.SaveToken = true;
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["Jwt:Issuer"],
        ValidAudience = builder.Configuration["Jwt:Audience"],
        IssuerSigningKey = new
            SymmetricSecurityKey
            (Encoding.UTF8.GetBytes
            (builder.Configuration["Jwt:Key"]))
    };
});
```


Como ativar os serviços para a Sessão e a Autenticação?

```
builder.Services.AddSession();
```

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options =>
{
    options.SaveToken = true;
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = builder.Configuration["Jwt:Issuer"],
        ValidAudience = builder.Configuration["Jwt:Audience"],
        IssuerSigningKey = new
            SymmetricSecurityKey
            (Encoding.UTF8.GetBytes
            (builder.Configuration["Jwt:Key"]))
    };
});
```

Estamos a definir as opções personalizadas de autenticação para utilizar JWT com as seguintes opções

Algumas das opções são lidas do ficheiro appsettings.json

É necessário ativar os seguintes middlewares

```
app.UseHttpsRedirection();  
app.UseStaticFiles();
```

```
app.UseRouting();  
app.UseSession();  
app.UseStatusCodePages();  
app.UseAuthentication();  
app.UseAuthorization();
```

- Middleware para ativar o estado da sessão (cookies) da aplicação
- Middleware para ativar autenticação na nossa aplicação
- Middleware para ativar autorização na nossa aplicação

Como proteger um endpoint?

```
[Authorize]
[Route("api/[controller]")]
[ApiController]
```

0 references

```
public class TestController : ControllerBase
{
```

```
    // GET: api/<TestController>
    [HttpGet]
```

0 references

```
    public TestModel Get()
    {
        return new TestModel { Id = 1, Name="Test" };
    }
}
```

- O atributo **Authorize** especifica que a classe ou o método requer a autorização especificada
- Pode ser colocado na classe ou no método

Como proteger um endpoint?

[Authorize]

0 references

```
public IActionResult ApiTest()
{
    TestModel test = null;
    using (var client = new HttpClient())
    {
        var token = HttpContext.Session.GetString("Token");
        client.BaseAddress = new Uri("https://localhost:7071/api/");

        // Add token to header
        client.DefaultRequestHeaders.Add("Authorization", "Bearer " + token);
        //HTTP GET
        var responseTask = client.GetAsync("test");
        responseTask.Wait();
    }
}
```

- O atributo **Authorize** especifica que a classe ou o método requer a autorização especificada
- Pode ser colocado na classe ou no método

Como expôr um endpoint?

```
[AllowAnonymous]  
[Route("login")]  
[HttpPost]
```

0 references

```
public IActionResult Login(UserModel userModel)  
{  
    if (string.IsNullOrEmpty(userModel.UserName) ||  
        {  
            return (RedirectToAction("Error"));  
        }  
}
```

- O atributo **AllowAnonymous** especifica que a classe ou o método não requer qualquer autorização

Como invocar um endpoint protegido no Postman?

- Da mesma forma que antes mas agora temos que enviar um cabeçalho de autorização
- O tipo do cabeçalho é **Bearer Token**
- **Bearer Token** significa “give access to the bearer of this token”
- Temos que enviar o token que foi gerado pelo servidor no momento da autenticação do utilizador

GET

https://localhost:7071/api/test

Send

Params

Authorization ●

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Type

Bearer Token

The authorization header will be automatically generated when you send the request.
[Learn more about authorization](#)

! Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Token

eyJhbGciOiJIodHRwOi8vd3d3LnczLm9yZy8...

Body

Cookies

Headers (4)

Test Results

Status: 200 OK Time: 58 ms Size: 170 B Save Response

Pretty

Raw

Preview

Visualize

JSON

1 {

2 "id": 1,

3 "name": "Test"

4 }

Como invocar um endpoint protegido através da aplicação?

- Tal como no Postman temos que arranjar forma de adicionar um cabeçalho de autorização no nossos pedidos
- Podemos criar um inline middleware responsável por isso
- Seleccionamos o token que está guardado na sessão
- E adicionámos um cabeçalho do tipo **Bearer** com o token

Inline Middleware for Header Bearer Token

```
app.Use(async (context, next) =>
{
    var token = context.Session.GetString("Token");

    if (!string.IsNullOrEmpty(token))
    {
        context.Request.Headers.Add("Authorization", "Bearer " + token);
    }

    await next();
});
```


Como implementar a autenticação?

- SIGNUP
 - Verificar se o utilizador já existe
 - Adicionar novo utilizador à BD
- LOGIN
 - Verificar se o utilizador existe ou credenciais corretas
 - Gerar um token para esse mesmo utilizador
 - Esse token será utilizado na autorização dos próximos pedidos
- LOGOUT
 - Remover o token da sessão do utilizador

[AllowAnonymous]

[HttpPost]

0 references

```
public async Task<IActionResult> SignUp(UserModel user)
{
    if (ModelState.IsValid)
    {
        var userExists = userService.FindByName(user.UserName);
        if (userExists != null)
            return StatusCode(StatusCodes.Status500InternalServerError, "User already exists!");

        var newUser = userService.Create(user);
        if (newUser is not null)
            return RedirectToAction(nameof(Index));
        else
            return RedirectToAction(nameof(Error));
    }
    else
    {
        return RedirectToAction(nameof(Error));
    }
}
```

```
[AllowAnonymous]
```

```
[Route("login")]
```

```
[HttpPost]
```

```
0 references
```

```
public IActionResult Login(UserModel userModel)
```

```
{
```

```
    if (string.IsNullOrEmpty(userModel.UserName) || string.IsNullOrEmpty(userModel.Password))
```

```
    {
```

```
        return (RedirectToAction("Error"));
```

```
    }
```

```
    var user = userService.Get(userModel.UserName, userModel.Password);
```

```
    var validUser = new UserViewModel { UserName = user.UserName, ID = user.ID, Role = user.Role, Email = user.Email };
```

```
    if (validUser != null)
```

```
    {
```

```
        string generatedToken = tokenService.GenerateToken(
```

```
            config["Jwt:Key"].ToString(),
```

```
            config["Jwt:Issuer"].ToString(),
```

```
            config["Jwt:Audience"].ToString(),
```

```
            validUser);
```

```
        if (generatedToken != null)
```

```
        {
```

```
            HttpContext.Session.SetString("Token", generatedToken);
```

```
            return RedirectToAction("UserDetails", validUser);
```

```
        }
```

```
        else
```

```
        {
```

```
            return (RedirectToAction("Error"));
```

```
        }
```

```
    }
```

```
    else
```

```
    {
```

```
        return (RedirectToAction("Error"));
```

```
    }
```

```
}
```

0 references

```
public IActionResult Logout()
{
    return View();
}
```

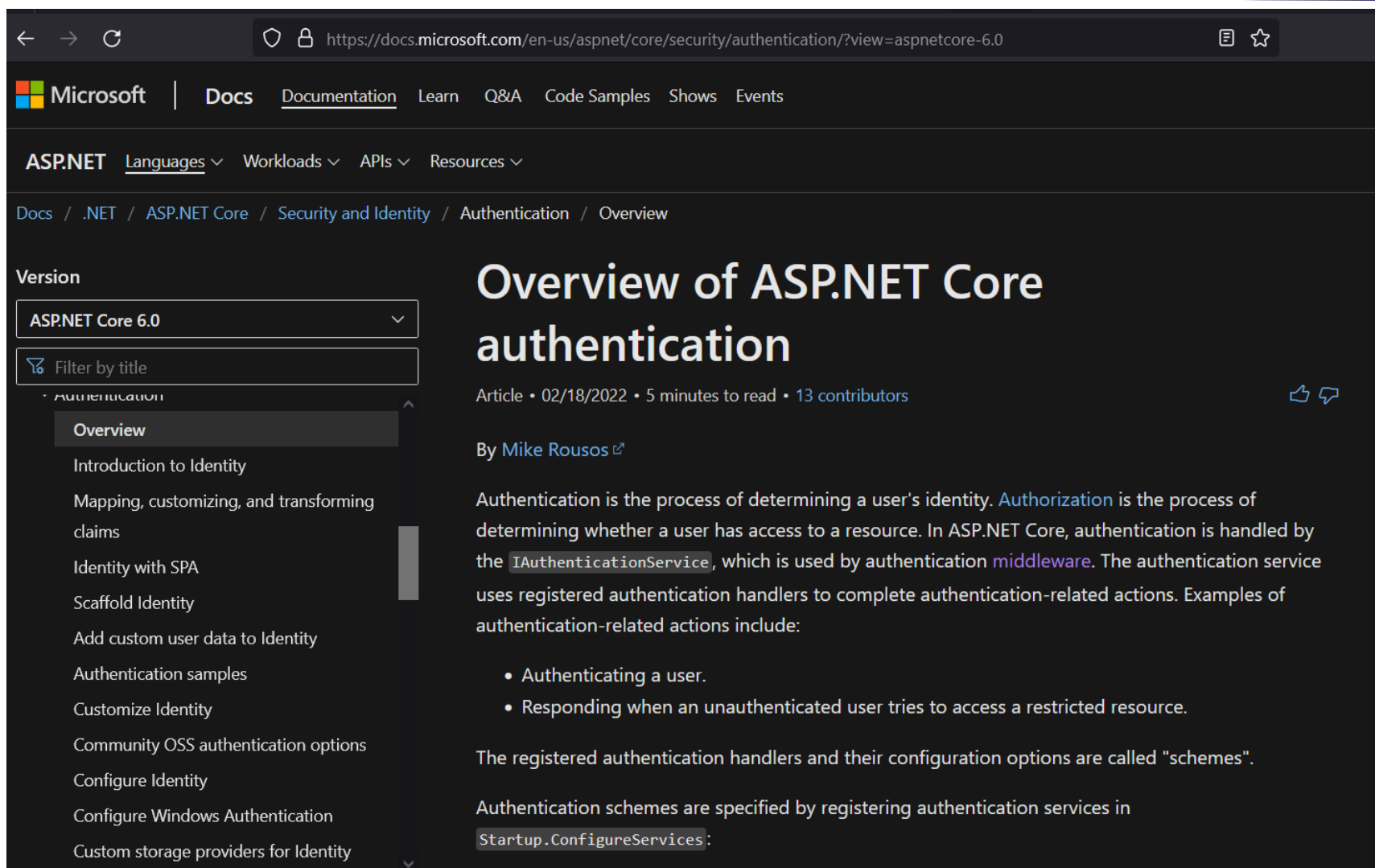
[AllowAnonymous]

[HttpPost]

0 references

```
public IActionResult LogoutUser()
{
    HttpContext.Session.Remove("Token");
    return (RedirectToAction("Index"));
}
```

Referências



The screenshot shows a web browser displaying the Microsoft Docs page for 'Overview of ASP.NET Core authentication'. The browser's address bar shows the URL: `https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-6.0`. The Microsoft Docs navigation bar is visible at the top, with links for Docs, Documentation, Learn, Q&A, Code Samples, Shows, and Events. Below this, the breadcrumb trail reads: Docs / .NET / ASP.NET Core / Security and Identity / Authentication / Overview. On the left side, there is a 'Version' dropdown menu set to 'ASP.NET Core 6.0' and a search filter box labeled 'Filter by title'. A sidebar menu lists various authentication topics, with 'Overview' currently selected. The main content area features the article title 'Overview of ASP.NET Core authentication' in a large font, followed by metadata: 'Article • 02/18/2022 • 5 minutes to read • 13 contributors'. The author is listed as 'By Mike Rousos'. The article text begins with 'Authentication is the process of determining a user's identity. Authorization is the process of determining whether a user has access to a resource. In ASP.NET Core, authentication is handled by the `IAuthenticationService`, which is used by authentication middleware. The authentication service uses registered authentication handlers to complete authentication-related actions. Examples of authentication-related actions include:

- Authenticating a user.
- Responding when an unauthenticated user tries to access a restricted resource.

The registered authentication handlers and their configuration options are called "schemes".

Authentication schemes are specified by registering authentication services in `Startup.ConfigureServices`:

qualificar

TAL

X

</>

A <PROGRAMAÇÃO