ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2025

Luiz Fernando Souza / Eliane Marion

Autenticação via Token com JSON Web Tokens (JWT)

Nossas aplicações até o momento tem todos os métodos programados com acesso livre na controller, mas podemos fazer com que os métodos das controllers sejam acessados apenas se houver uma autenticação válida, porém, esse processo acarretaria a necessidade de entrar com as credenciais a cada método solicitado.

Para facilitar os procedimentos, podemos utilizar um token que pode ficar armazenado no navegador de uma aplicação ou de um dispositivo, sendo que este token conterá informações essenciais de identificação de um usuário, mas não informações sensíveis que podem ser roubadas, além disso, o token pode ser criado já com uma data de validade atribuída.

JSON Web Tokens é um recurso muito útil para tudo isso que falamos até então e faremos a aplicação prática na API, mas para que conheça um pouco da teoria do que estamos falando, segue um vídeo para introdução sobre Token e JWT:

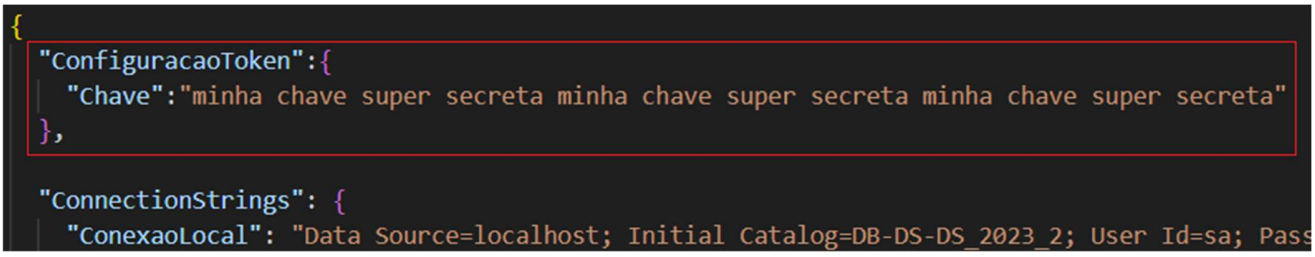
Token: https://youtu.be/LtVb9rhU41c

JWT: https://youtu.be/Gyq-yeot8qM

Artigo: https://www.brunobrito.net.br/jwt-cookies-oauth-bearer/

O JWT nada mais é do que uma série de caracteres que contém especificações sobre um usuário, chamamos essas especificações de Claims (Reivindicações), ou seja, através desses dados é possível saber quais os tipos de acesso que determinado usuário poderá ter numa API, por exemplo, ou simplesmente resgar informações armazenadas no token gerado.

1. Abra o projeto RpgApi, vá até o arquivo appsettings.json e insira a codificação que será a chave para gerar o token



```
{
  "ConfiguracaoToken":{
    "Chave":"minha chave super secreta minha chave super secreta minha chave super secreta"
  },

  "ConnectionStrings": {
    "ConexaoLocal": "Data Source=localhost; Initial Catalog=DB-DS-DS_2023_2; User Id=sa; Pass
```

2. Utilize o terminal para instalar os pacotes necessários para programação do método que vai gerar o Token:
   - ➔ dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer (-v 8.0.8 se for versão .net 8)
   - ➔ dotnet add package System.IdentityModel.Tokens.Jwt (- v 8.1.0 se for versão .net 8)
   - ➔ dotnet add package Microsoft.IdentityModel.Tokens (- v 8.1.0 se for versão .net 8)

   Perceba que ao abrir o arquivo RpgApi.csProj, as referências aos pacotes estarão dentro da tag **ItemGroup** conforme a seguir:

```
<PackageReference Include="Microsoft.IdentityModel.Tokens" Version="8.1.0"/>
<PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="8.1.0"/>
<PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="8.0.8"/>
```

3. Abra a classe Usuario e adicione uma propriedade chamada Token, conforme abaixo

```
[NotMapped]
0 references
public string Token { get; set; } = string.Empty;
```

4. Altere o construtor da classe **UsuariosController** para permitir acesso as configurações criadas anteriormente. Utilize o using *Microsoft.Extensions.Configuration*.

```
private readonly DataContext _context;
1 reference
private readonly IConfiguration _configuration;
0 references
public UsuariosController(DataContext context, IConfiguration configuration)
{
    _context = context;
    _configuration = configuration;
}
```

5. Na classe UsuariosController desenvolva o método que criará o token. Usings: System.Security.Claims, Microsoft.IdentityModel.Tokens, System.Text, System.IdentityModel.Tokens.Jwt

```
        private string CriarToken(Usuario usuario)

        {
            List<Claim> claims = new List<Claim>
            {
                new Claim(ClaimTypes.NameIdentifier, usuario.Id.ToString()),
                new Claim(ClaimTypes.Name, usuario.Username)
            };
            SymmetricSecurityKey key = new SymmetricSecurityKey(Encoding.UTF8
            .GetBytes(_configuration.GetSection("ConfiguracaoToken:Chave").Value));
            SigningCredentials creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha5
12Signature);
            SecurityTokenDescriptor tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
                Expires = DateTime.Now.AddDays(1),
                SigningCredentials = creds
            };
            JwtSecurityTokenHandler tokenHandler = new JwtSecurityTokenHandler();
            SecurityToken token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
```

6. Altere o retorno do método de autenticação para que no return Ok, usemos o trecho abaixo.

```
else
{
    usuario.DataAcesso = System.DateTime.Now;
    _context.TB_USUARIOS.Update(usuario);
    await _context.SaveChangesAsync(); //Confirma a alteração no banco

    usuario.PasswordHash = null;
    usuario.PasswordSalt = null;
    usuario.Token = CriarToken(usuario);
    return Ok(usuario);
}
```

7. Abra o postman e realize o teste de Autenticação de um usuário já salvo na base, confirmando que será exibido o token, caso a autenticação tenha sucesso. Você pode conferir os dados existentes no Token copiando-o e usando no site https://jwt.io/

8. Abra a classe Program.cs e faça a edição do método ConfigureServices como segue abaixo. Esta etapa será importante para que possamos resgatar as informações do Token. Necessário using para Microsoft.AspNetCore.Authentication.JwtBearer; Microsoft.IdentityModel.Tokens e System.Text.

```
builder.Services.AddControllers().AddNewtonsoftJson(options =>
        options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore
);

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII
                .GetBytes(builder.Configuration.GetSection("ConfiguracaoToken:Chave").Value)),
            ValidateIssuer = false,
            ValidateAudience = false
        };
    });
```

9. Ainda na classe Program.cs, adicione a linha que implanta a Autenticação e Autorização como a seguir

```
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
app.Run();
```

10. Vá até a controller de Usuários e adicione no topo da classe o atributo de Autorização. Com o atributo Authorize iremos limitar quem pode acessar a controller ou algum método dentro da Controller. Iniciaremos realizando os procedimentos de configuração.

```
[Authorize]
[ApiController]
[Route("[controller]")]
0 references
public class UsuariosController : ControllerBase
{
    5 references
    private readonly DataContext _context;
```

- Será necessário o *using Microsoft.AspNetCore.Authorization*.
- Tente realizar o teste do método de autenticação no postman para verificar qual status é retornado.

11. Provavelmente, o teste da etapa anterior deve ter retornado uma mensagem 401 (Unauthorized). Vamos inserir uma permissão para apenas para os métodos "***Autenticar***" e "***Registrar***" conforme a seguir

```
[AllowAnonymous]
[HttpPost("Autenticar")]
0 references
public async Task<IActionResult> AutenticarUsuario(Usuario credenciaisUsuario)
{
    Usuario usuario = await _context.Usuarios.FirstOrDefaultAsync(x =>
        x.Username.ToLower().Equals(credenciaisUsuario.Username.ToLower()));
```

- Esse atributo concede uma exceção para que um usuário anônimo consiga acessar o método contido numa controller restrita.

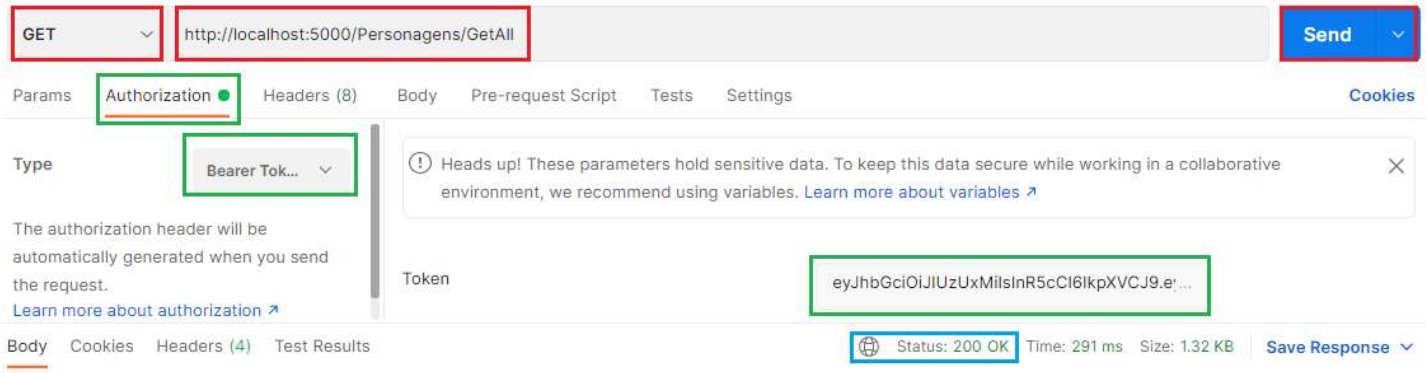
- Realize novamente o teste e copie o token gerado e guarde em um bloco de notas para usarmos, quando necessário.

12. Adicione o atributo Authorize no topo da controller de Personagem também

```
[Authorize]
[ApiController]
[Route("[controller]")]
0 references
public class PersonagemController : ControllerBase
{
```

- Será necessário o *using Microsoft.AspNetCore.Authorization*

13. Configuraremos o postman para testar o Método GetAll de Personagens passando os dados do Token no cabeçalho da requisição http, conforme as configurações feitas na sinalização em verde. A sinalização em Azul é o resultado obtido e a sinalização em vermelho são as configurações gerais que usamos para métodos get.



- Perceba que adicionamos a key (chave) “Authorization” e no Type a palavra “Bearer” (indica que é uma autorização de token Jwt). Dessa maneira, a API reconhecerá que existe permissão para acessar os métodos da controller.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2025

Luiz Fernando Souza / Eliane Marion

Identificação centralizada do Usuário

Ao invés de criar a codificação para identificar o usuário em cada método, criaremos a seguir uma classe e métodos para coletar os dados do usuário através do token que poderá ser usado nos demais métodos da controllers, e caso tenhamos que usar em outras controllers, será desenvolvido da mesma maneira.

1. Crie uma pasta chamada Extensions e dentro da pasta crie uma classe estática chamada ClaimTypesExtesion. Uma classe estática é aquela pode possuir métodos estáticos, sem que tenhamos que criar um objeto instanciado.

```
public static class ClaimTypesExtension
```

2. Crie um método de extensão conforme abaixo. Ele retornará o Id do usuário de acordo com a leitura feita na claim que está dentro do token. Necessário ***using System.Security.Claims***.

```
public static int UsuarioId(this ClaimsPrincipal user)
{
    try
    {
        var usuarioId = user.Claims.FirstOrDefault(x => x.Type == ClaimTypes.NameIdentifier)?.Value ?? string.Empty;
        return int.Parse(usuarioId);
    }
    catch
    {
        return 0;
    }
}
```

3. Vamos obter os personagens de acordo com a leitura das reivindicações do usuário (Claims). Programe o método GetByUser na controller de Personagem.

```
[HttpGet("GetByUser")]
0 references
public async Task<IActionResult> GetByUserAsync()
{
    try
    {
1       int id = User.UsuarioId();

        List<Personagem> lista = await _context.TB_PERSONAGENS
2           .Where(u => u.Usuario.Id == id)
            .ToListAsync();

        return Ok(lista);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message + " - " + ex.InnerException);
    }
}
```

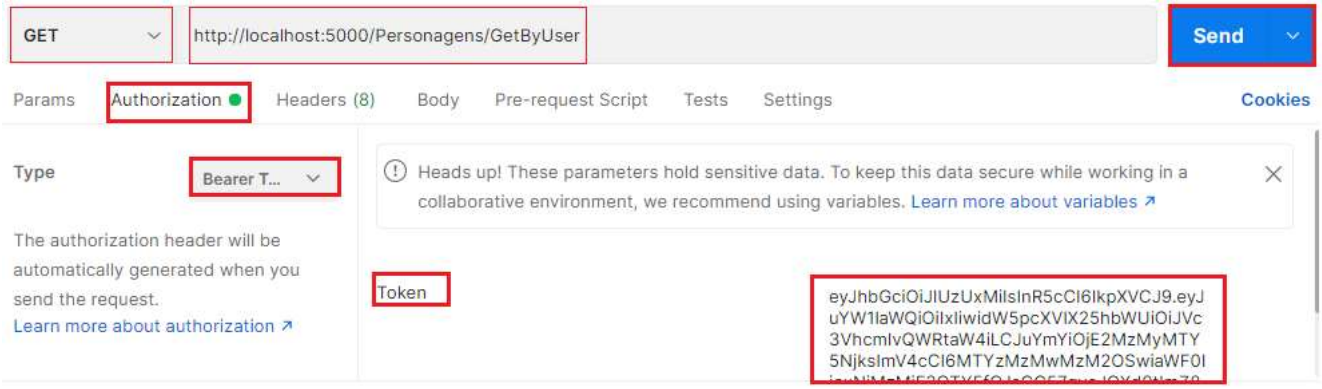
(1) Quando criamos o token criamos também Claims, sendo que a identificada como NameIdentifier foi atrelada ao Id do usuário, logo ao recuperar esta claim do Token, conseguimos o Id do usuário.

(2) O método que vai no Banco de dados para recuperar os personagens tem um filtro em que é passado o Id do usuário, desta forma recuperaremos os personagens atrelados a Id que a Claim continha.

Para a próxima etapa, sua tabela de usuários deverá ter mais de um usuário cadastrado, se o cenário não for esse, faça o cadastro de usuários via Postman e não esqueça de usar o token conforme o item anterior.

Sua tabela de personagens também deve ter mais que um personagem, caso o cenário não seja esse, insira via SQL e atribua na coluna UsuarioId de todos os personagens, algum Id de usuário existente criando variações. Isso será feito manualmente agora, pois vamos usar o relacionamento criado na aula anterior em que um personagem pertence a um usuário e um usuário pode ter vários personagens, mas nas próximas aulas automatizaremos o processo de inserção de personagem identificando o usuário para o postman.

4. Faça a configuração para testar via postman, utilizando o token. Não esqueça que o método criado tem uma rota diferente.



Com o relacionamento existente entre as tabelas usuários e personagens, a coluna UsuarioId da base de dados não está sendo preenchida automaticamente ao fazer um post de personagens. Com base nas linhas de programação realizadas nesta aula, podemos verificar uma forma de identificar o id do usuário que está autenticado, carregar um objeto do tipo Usuário através deste id e atribuir na propriedade usuário do objeto do tipo Personagem quando formos salvar e editar um personagem. Dica: A modificação deve ser feita na controller de Personagem.

Buscaremos no método de extensão, a Claim de Id obtida no Token, conforme sinalizado

```
[HttpPost]
0 references
public async Task<IActionResult> Add(Personagem novoPersonagem)
{
    try
    {
        if (novoPersonagem.PontosVida > 100)
            throw new Exception("Pontos de vida não pode ser maior que 100");

        novoPersonagem.Usuario = _context.TB_USUARIOS.FirstOrDefault(uBusca => uBusca.Id == User.UsuarioId());

        await _context.TB_PERSONAGENS.AddAsync(novoPersonagem);
        await _context.SaveChangesAsync();

        return Ok(novoPersonagem.Id);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message + " - " + ex.InnerException);
    }
}
```

- Faça o mesmo de método Update no trecho parecido com o que fizemos acima

- Para testar via postman é necessário fazer a autenticação do Usuário, copiar o Token e preencher o valor do Token no Header ao cadastrar ou atualizar o novo personagem, conforme fizemos anteriormente.

- Conseguimos realizar esse salvamento e atualização devido o relacionamento um para muitos, presente entre a classe usuário e personagem criado nas aulas anteriores, refletido no banco de dados quando fizemos a migração.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2025

Luiz Fernando Souza / Eliane Marion

Autenticação baseada em Papéis

Criaremos definições de perfis ou papéis para cada usuário, e assim, resgatar qual o tipo de usuário está requisitando operações para apresentar os dados e acessos que ele pode ter. Além disso, com a esta programação da API sendo realizada, será possível programar o projeto MVC para aplicar as chamadas da API.

Acrescentando mais uma claim para o Token:

Até este momento, guardávamos na coleção de claims os Id o usuário e o login dele, faremos a programação para acrescentar uma claim para guardar Perfil.

1. Abra a controller de usuário e modifique o método CriarToken

```
private string CriarToken(Usuario usuario)
{
    List<Claim> claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, usuario.Id.ToString()),
        new Claim(ClaimTypes.Name, usuario.Username),
        new Claim(ClaimTypes.Role, usuario.Perfil)
    };
```

2. Execute a API, faça o teste no postman. Copie o token gerado e use o site https://jwt.io/ para ver o conteúdo da Claim, devendo constar o Perfil.

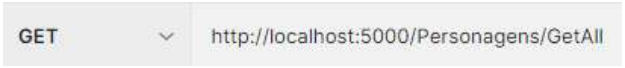
3. Abra a controller de Personagem e altere a anotação Authorize, que indicará o tipo de usuário que poderá acessar a controller.

```
[Authorize(Roles = "Jogador")]
```

4. Altere o usuário que você mais usa para o perfil Admin. Execute a API, autentique e tente buscar todos os personagens

Id	Username	PasswordHash	PasswordSalt	Perfil
1	UsuarioAdmin	0x46FF85DB9B46...	0xDF24CD36712...	Admin
2	UsuarioTeste	0xE9BCB9A8D1D...	0xDE020B594ED...	Jogador

5. Abra o postman, faça a autenticação e use o Token para obter todos os personagens.



6. Você perceberá que vai dar o erro 403, de proibição. Adicione a palavra Admin nas permissões da controller, execute a API e teste novamente.

```
[Authorize(Roles = "Jogador, Admin")]
```

7. Filtrando personagens de acordo com o perfil: Crie o método de extensão na classe ClaimTypesExtension para obter o perfil do usuário

```
public static string UsuarioPerfil(this ClaimsPrincipal user)
{
    try
    {
        var usuarioPerfil = user.Claims.FirstOrDefault(x => x.Type == ClaimTypes.Role)?.Value ?? string.Empty;
        return usuarioPerfil;
    }
    catch
    {
        return string.Empty;
    }
}
```

8. Crie o método GetByPerfilAsync para pegar os personagens de acordo com o perfil identificado. Observe que criamos a lista de maneira vazia em memória e consultamos qual o perfil do usuário, se ele for admin retornará todos os personagens, caso contrário vai filtrar os personagens que tem o id de usuário igual ao contido no token.

```
[HttpGet("GetByPerfil")]
0 references
public async Task<IActionResult> GetByPerfilAsync()
{
    try
    {
        List<Personagem> lista = new List<Personagem>();

        if (User.UsuarioPerfil() == "Admin")
            lista = await _context.TB_PERSONAGENS.ToListAsync();
        else
            lista = await _context.TB_PERSONAGENS
                    .Where(p => p.Usuario.Id == User.UsuarioId()).ToListAsync();
        return Ok(lista);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message + " - " + ex.InnerException);
    }
}
```

- Execute a API e tenha dois usuários criados em sua base, um admin e outro jogador, com Personagens que pertençam a estes usuários. Faça o teste logando com cada um destes usuários e use o token para requisitar ao método *GetByPerfil* para perceber que o resultado das consultas é diferente em cada caso.

- Boas práticas para retorno dos erros: Concatenar em todos os catch's InnerExceptions dos erros

```
return BadRequest(ex.Message + " - " + ex.InnerException);
```

- Com esses ajustes podemos realizar a publicação da API em servidores externos.