ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2024/1

Luiz Fernando Souza / Quitéria Danno

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 7.0.16 se for versão .net 7)
   - ➔ dotnet add package System.IdentityModel.Tokens.Jwt (- v 7.3.1 se for versão .net 7)
   - ➔ dotnet add package Microsoft.IdentityModel.Tokens (- v 7.3.1 se for versão .net 7)

   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="7.3.1"/>
<PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="7.3.1"/>
<PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="7.0.16"/>
```

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*. Faça o mesmo na classe **PersonagensController.cs**

```
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.Secutiry.Claims, Microsoft.IdentityModel.Tokens, System.Text, System, 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 como a seguir

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

app.MapControllers();
```

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.

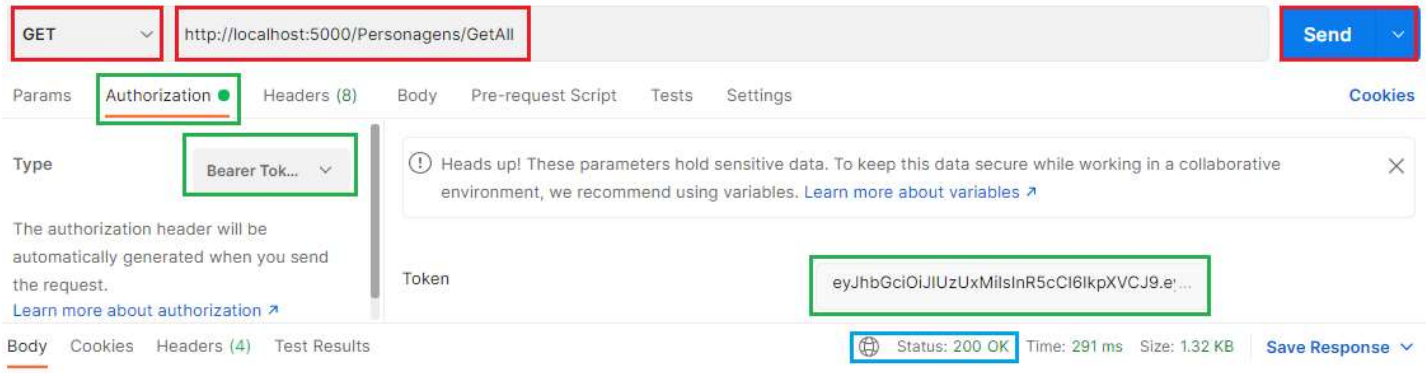
- Realize novamente o teste e copie o token gerado e guarde em um bloco de notas para usarmos, quando necessário.

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), com um espaço e depois o token gerado anteriormente. Dessa maneira, a API reconhecerá que existe permissão para acessar os métodos da controller.

Obtendo os Personagens de acordo com a leitura das Reivindicações de Usuário (Claims)

Para esta 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.

14. Programe o método GetByUser na controller de Personagem.

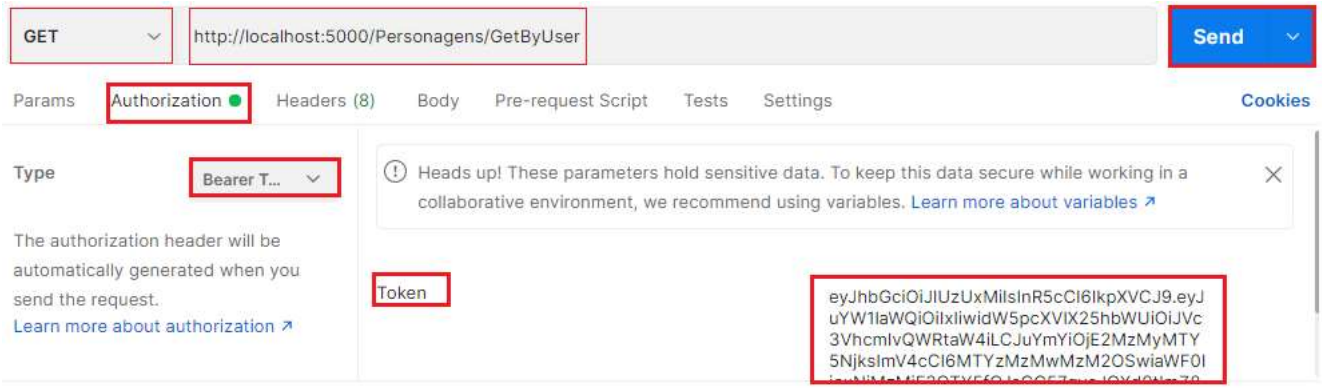
```
[HttpGet("GetByUser")]
0 references
public async Task<IActionResult> GetByUserAsync()
{
    try
    {
      1 int id = int.Parse(User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value);

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

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

- Necessário using de System.Linq e System.Security.Claims

(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.

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



Identificação centralizada do Usuário

Ao invés de criar a codificação para identificar o usuário em cada método, criaremos a seguir um método 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.

16. Abra a classe Program.cs e adicione a codificação abaixo. Será necessário o using Microsoft.Extensions.DependencyInjection.Extensions;

```
builder.Services.TryAddSingleton<IHttpContextAccessor, HttpContextAccessor>();

var app = builder.Build();
```

17. Modifique o construtor da controller PersonagemController declarando a variável do tipo HttpContextAcessor e inicializando-a.

```
private readonly IConfiguration _configuration;
1 reference
private readonly IHttpContextAccessor _httpContextAccessor;

0 references
public PersonagensController(DataContext context, IConfiguration configuration, IHttpContextAccessor httpContextAccessor)
{
    //Inicialização do atributo a partir de um parâmetro
    _context = context;
    _configuration = configuration;
    _httpContextAccessor = httpContextAccessor;
}
```

- O código abaixo criará a condição para quando esta classe for instanciada em memória, possamos acessar a configuração Singleton.

18. Abra a controlller de Personagem e crie um método que centralizará a ação de obter o Id do usuário. Observe o método está se utilizando da variável criada na etapa anterior para coletar o dado identificador da Claim, o Id.

```
private int ObterUsuarioId()
{
    return int.Parse(_httpContextAccessor.HttpContext.User.FindFirstValue(ClaimTypes.NameIdentifier));
}
```

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.

Para que o Id o usuário fique salvo na tabela de personagens, faremos busca do id através das Claims obtidas no Token, depois buscaremos no Banco de Dados, guardando na propriedade Usuário existente no objeto do tipo Personagem.

Salvando o Personagem com o Id do Usuário

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

        novoPersonagem.Usuario = _context.Usuarios.FirstOrDefault(uBusca => uBusca.Id == ObterUsuarioId());

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

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

Atualizando o Personagem com o Id do usuário preenchido. Observe que desta vez o uso await/async

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

        novoPersonagem.Usuario = _context.Usuarios.FirstOrDefault(uBusca => uBusca.Id == ObterUsuarioId());

        _context.Personagens.Update(novoPersonagem);
        int linhasAfetadas = await _context.SaveChangesAsync();

        return Ok(linhasAfetadas);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

- 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.