ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2024/1

Luiz Fernando Souza / Quitéria Danno

Autenticação baseada em Papéis

Nesta aula 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 ter 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	0x31AE472BE700C0FB9E6D712F86A...	0x5FAA27F18506D9A16BC4A24EFD61DF74...	Admin
2	UsuarioTeste	0x97242B009D4D98EC3BE3E149339...	0xB01E4D66773B1FCBE8A4117F447802EA...	Jogador
3	UsuarioTeste2	0x3614DB7BE8750BE85EE3685715B...	0x5166F5E277FBC3583D000223C985C111F...	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")]
```

Filtrando Personagens de acordo com o perfil do usuário

7. Crie o método ObterPerfilUsuario na controller de personagem

```
private string ObterPerfilUsuario()
{
    return _httpContextAccessor.HttpContext.User.FindFirstValue(ClaimTypes.Role);
}
```

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(ObterperfilUsuario() == "Admin")
            lista = await _context.Personagens.ToListAsync();
        else
        {
            lista  = await _context.Personagens
                    .Where(p => p.Usuario.Id == ObterUsuarioId()).ToListAsync();
        }
        return Ok(lista);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

- 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.