



## 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.

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

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

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

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

2. Modifique o construtor da controller PersonagemController declarando a variável do tipo IHttpContextAccessor 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.
3. Abra a controller 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 Usuariold 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.

```
[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.



## 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 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.

```
GET http://localhost:5000/Personagens/GetAll
```



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.