

## > Ficha Prática Nº 6

### Objetivos/Temas:

- Segurança em aplicações web
- Microsoft Identity Core

## > Exercícios

- CRIAR REGRAS DE ACESSO
- CONFIGURAR ROLES
- CRIAR E MANIPULAR ROLES
- CRIAR E MANIPULAR UTILIZADORES
- ATRIBUIR ROLES

### >> Microsoft ASP.Net Identity Core – Autorização

A autorização refere-se ao processo de determinar se um utilizador tem permissão para realizar determinada operação / aceder a um recurso (listar registos, editar registos, etc.).

Existem 3 tipos de autorização:

- Autorização Simples
  - Atributo **[Authorize]** - Só utilizadores autenticados é que podem aceder ao recurso (controller, método).
- Autorização com Roles
  - Atributo **[Authorize(Roles="Role1, Role2, ..., Role10")]** - Só utilizadores autenticados e que pertençam a uma das roles enumeradas é que podem aceder ao recurso (controller, método).
- Autorização com Políticas
  - Atributo **[Authorize(Policy = "Maiores18")]** – Só utilizadores que cumpram os requisitos definidos na politica de acesso "Maiores18" é que podem aceder ao recurso(controller, método).

1. Adicione o atributo **[Authorize]** na classe **AgendamentoController** - *controller Agendamento*.

```
[Authorize]  
public class AgendamentosController : Controller
```

2. Termine a sessão na aplicação web (caso tenha sessão iniciada).
3. Clique na opção “**Agendamentos**” no menu superior.
  - Analise o comportamento da aplicação.
4. Experimente aceder a qualquer vista do **controller Agendamentos** e analise o resultado obtido.
5. Remova o atributo **[Authorize]** adicionado no ponto 1.

```
[Authorize]
public class AgendamentosController : Controller
```

6. Adicione o atributo **[Authorize]** no método **Index** do **controller Agendamento**.

```
// GET: Agendamentos
[Authorize]
public async Task<IActionResult> Index(){
    var applicationDbContext = _context.Agendamentos.Include(a => a.tipoDeAula);
    return View(await applicationDbContext.ToListAsync());
}
```

7. Ainda com a sessão terminada, experimente aceder:

- À listagem de agendamentos – método *Index* – controller *Agendamentos*.
- Ao pedido de agendamento – método *Pedido* – controller *Agendamentos*.

Verifique o comportamento de aplicação. Veja as diferenças entre aplicar o atributo **[Authorize]** na classe que define o controller ou aplicar apenas num método específico.

## >> Microsoft ASP.Net Identity Core – Roles (IdentityRole)

**Roles** servem para identificar as funções que um utilizador desempenha/pertence numa aplicação. Um utilizador pode pertencer a uma ou mais Roles. Por exemplo, o “*João*” pode ser **administrador** e **funcionário** enquanto o “*Paulo*” pode ser apenas **Cliente**.

Desta forma é possível restringir o acesso a determinados(as) recursos/funcionalidades da aplicação, através da criação de regras de acesso (autorização).

Para ativar/registar a funcionalidade **Roles** é necessário modificar o **Program.cs**:

```
builder.Services.AddDefaultIdentity<ApplicationUser>(
    options => options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Tipos de autorização com Roles:

- Simple – o utilizador tem de pertencer à role indicada

```
[Authorize(Roles = "Administrador")]
public class AgendamentosController : Controller
```

- Múltipla ( ou ) – o utilizador tem de pertencer **a uma** das roles indicadas

```
[Authorize(Roles = "Administrator,Funcionario,Gestor ")]
public class AgendamentosController : Controller
```

- Múltipla ( e ) – o utilizador tem de pertencer **a todas** as rolas indicadas

```
[Authorize(Roles = "Administrator")]
[Authorize(Roles = "Funcionario")]
public class AgendamentosController : Controller
```

Para criar Roles/Utilizadores Iniciais na aplicação faça o seguinte:

8. Crie uma classe, dentro da diretoria **Data**, com o nome **Inicializacao**, com o seguinte código (com a devida adaptação ao seu projecto):

```
using Microsoft.AspNetCore.Identity;
using PWEB_AulasP_2223.Models;
using System;

namespace PWEB_AulasP_2223.Data{
    public enum Roles{
        Admin,
        Formador,
        Cliente
    }

    public static class Inicializacao{
        public static async Task CriaDadosIniciais(UserManager<ApplicationUser>
userManager, RoleManager<IdentityRole> roleManager){
            //Adicionar default Roles
            await roleManager.CreateAsync(new IdentityRole(Roles.Admin.ToString()));
            await roleManager.CreateAsync(new IdentityRole(Roles.Formador.ToString()));
            await roleManager.CreateAsync(new IdentityRole(Roles.Cliente.ToString()));

            //Adicionar Default User - Admin
            var defaultUser = new ApplicationUser{
                UserName = "admin@localhost.com",
                Email = "admin@localhost.com",
                PrimeiroNome = "Administrador",
                UltimoNome = "Local",
                EmailConfirmed = true,
                PhoneNumberConfirmed = true
            };
            var user = await userManager.FindByEmailAsync(defaultUser.Email);
            if (user == null){
                await userManager.CreateAsync(defaultUser, "Is3C..00");
                await userManager.AddToRoleAsync(defaultUser,
Roles.Admin.ToString());
            }
        }
    }
}
```

9. Adicione as seguintes linhas ao ficheiro **Program.cs**, depois da linha `var app = builder.Build();`:

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        var userManager = services.GetRequiredService<userManager<ApplicationUser>>();
        var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
        await Inicializacao.CriaDadosIniciais(userManager, roleManager);
    }
    catch (Exception)
    {
        throw;
    }
}
```

10. Execute a aplicação e verifique na base de dados se foram criados os registos relativos às 3 Roles (**Admin, Formador, Cliente**) e ao utilizador **admin@localhost.com**.

dbo.AspNetRoles [Data] ApplicationDbContext.cs ScaffoldingRea

Max Rows: 1000

| Id               | Name     | NormalizedNa... | ConcurrencySt... |
|------------------|----------|-----------------|------------------|
| 06f0355a-66d6... | Admin    | ADMIN           | 7eb10116-28d6... |
| 81660b3a-4ce2... | Formador | FORMADOR        | 73054e29-7cf7... |
| ea858796-5f61... | Cliente  | CLIENTE         | 064f02be-911b... |
| NULL             | NULL     | NULL            | NULL             |

Figura 1 - tabelaAspNetRoles

|  | Id                | UserName        | NormalizedUserName  | Email           | NormalizedEmail     |
|--|-------------------|-----------------|---------------------|-----------------|---------------------|
|  | 140d57d3-813c-... | tiago2@isec.pt  | TIAGO2@ISEC.PT      | tiago2@isec.pt  | TIAGO2@ISEC.PT      |
|  | 2a5e252f-14f3-... | tiago3@isec.pt  | TIAGO3@ISEC.PT      | tiago3@isec.pt  | TIAGO3@ISEC.PT      |
|  | 326421b8-8deb-... | tiago4@isec.pt  | TIAGO4@ISEC.PT      | tiago4@isec.pt  | TIAGO4@ISEC.PT      |
|  | 417f625-a93c-...  | admin@localhost | ADMIN@LOCALHOST.COM | admin@localhost | ADMIN@LOCALHOST.COM |
|  | 73fac34c-cb91-... | tiago@isec.pt   | TIAGO@ISEC.PT       | tiago@isec.pt   | TIAGO@ISEC.PT       |
|  | NULL              | NULL            | NULL                | NULL            | NULL                |

Figura 2 tabela - AspNetUsers

11. Adicione as seguintes regras de autorização baseadas em Roles:

- *Controller Cursos*
  - Apenas os utilizadores com a Role *Admin* podem efetuar as operações de *Editar*, *Apagar* e *Criar*.
- *Controller Categorias*
  - Apenas os utilizadores com a Role *Admin* podem efetuar as operações de *Editar*, *Apagar* e *Criar*.

- *Controller Agendamentos*
  - Apenas os utilizadores com a Role *Admin* podem efetuar as operações de *Editar*, *Apagar*.
  - Apenas os utilizadores com a Role *Cliente* podem efetuar um agendamento (*Criar*).

12. Faça as alterações necessárias à página de Registo de utilizadores por forma a que os novos utilizadores sejam adicionados à Role Cliente.

## >> Gestão Roles

13. Crie um *controller* “**RoleManager**”.

14. Crie uma *view* vazia com o nome “**Index**” (~/*Views/RoleManager/Index.cshtml*).

15. Substitua o código da Classe `RoleManagerController` por:

```
public class RoleManagerController : Controller
{
    private readonly RoleManager<IdentityRole> _roleManager;
    public RoleManagerController(RoleManager<IdentityRole> roleManager)
    {
        /* código a criar */
    }
    public async Task<IActionResult> Index()
    {
        /* código a criar */
        return View(roles);
    }
    [HttpPost]
    public async Task<IActionResult> AddRole(string roleName)
    {
        /* código a criar */
        return RedirectToAction("Index");
    }
}
```

16. Substitua o código da *view Index* que criou no ponto 15 por:

```

@model IEnumerable<Microsoft.AspNetCore.Identity.IdentityRole>
@{
    ViewData["Title"] = "Role Manager";
    Layout = "~/Views/Shared/_Layout2.cshtml";
}
<h1>Gestão de Roles</h1>
<div class="row">
    <div class="col-md-4">
        <form method="post" asp-action="AddRole" asp-controller="RoleManager">
            <div class="input-group">
                <input name="roleName" class="form-control">
                <span class="input-group-btn">
                    <button class="btn btn-warning">Add New Role</button>
                </span>
            </div>
        </form>
    </div>
</div>
<table class="table table-striped table-hover table-bordered mt-4">
    <thead>
        <tr>
            <th>Role</th>
            <th>Id</th>
        </tr>
    </thead>
    <tbody>
        /* código a criar */
    </tbody>
</table>

```

17. Faça as alterações necessárias ao código fornecido for forma a ser possível listar todas as Roles existentes bem como adicionar novas Roles.

## >> Gestão de utilizadores / atribuição de Roles

18. Crie os seguintes ViewModel:

```

public class UserRolesViewModel
{
    public string UserId { get; set; }
    public string PrimeiroNome { get; set; }
    public string UltimoNome { get; set; }
    public string UserName { get; set; }
    public IEnumerable<string> Roles { get; set; }
}

public class ManageUserRolesViewModel
{
    public string RoleId { get; set; }
    public string RoleName { get; set; }
    public bool Selected { get; set; }
}

```

19. Crie um controller “**UserRolesManager**” e substitua o código da classe por:

```

public class UserRolesManagerController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;

    public UserRolesManagerController(UserManager<ApplicationUser> userManager,
    RoleManager<IdentityRole> roleManager)
    {
        /* código a criar */
    }
    public async Task<IActionResult> Index()
    {
        var users = await _userManager.Users.ToListAsync();
        /* código a criar */
        return View(userRolesViewModel);
    }
    private async Task<List<string>> GetUserRoles(ApplicationUser user)
    {
        return new List<string>(await _userManager.GetRolesAsync(user));
    }
    public async Task<IActionResult> Details(string userId)
    {
        /* código a criar */
        return View(model);
    }
    [HttpPost]
    public async Task<IActionResult> Details(List<ManageUserRolesViewModel> model,
    string userId)
    {
        /* código a criar */
        return RedirectToAction("Index");
    }
}

```

20. Crie duas views *Index* e *Details*.

21. Faça as alterações necessárias ao código fornecido e às views criadas de forma a:

- Listar todos os utilizadores e as suas roles.



- Ver em detalhe as Roles de um utilizador e modificar as mesmas

