**DOWNLOADS NECESSÁRIOS**

Visual Studio 2022 (com ASP.NET)

VSCode

Postman

DB Browser for SQLite

Projeto inicial - <https://github.com/alura-cursos/4320-seguranca-aspnetcore>

**CONFIGURAÇÕES INICIAIS**

Instalar gerenciador de BD do .NET - Na pasta MedVoll.Web, pelo CMD

*dotnet tool install --global dotnet-ef*

Aplicar as migrations iniciais do projeto -Pelo CMD, na pasta MedVoll.Web

*dotnet ef database update*

Com o projeto aberto no Visual Strudio - Instalar Identity - Pelo Nuget

Na interface do Visual Studio -> Ferramentas -> Gerenciador de Pacotes Nuget-> Gerenciar Pacotes

Nome: Microsoft.AspNetCore.Identity.EntityFrameworkCore (v. 9.0.0)

Alterar classe ApplicationDbContext para herdar de IdentityDbContext

public **class** **ApplicationDbContext** : IdentityDbContext

Scaffolding ao ASP.NETCoreIdentity - Criar a estrutura necessária para usar o identity na aplicação (nas views Account/Login e Account/Register)

-Clique com o botão direito no projeto MedVoll.Web no Solution Explorer.

-Escolha o menu: Add > Add New Scaffolded Item.

-Na lista, selecione Identity.

-Marque os arquivos necessários:

Account\Login

Account\Register

-No campo DbContext class, selecione:ApplicationDbContext (MedVoll.Web.Data).

Gerar migração para o esquema do ASP.NET Core Identity - Pelo CMD, na pasta MedVoll.Web

dotnet ef migrations add CreateIdentitySchema

Aplicar migração ao BD

dotnet ef database update

\*Para consultar o banco de dados da aplicação

Botão direito em vollmed.db -> abrir com -> DB Browser fo SQLite

**CRIAÇÃO DE USUÁRIOS DE TESTE**

Criar classe Data-> IdentitySeeder

using Microsoft.AspNetCore.Identity;

namespace MedVoll.Web.Data

{

public class IdentitySeeder

{

//MÉTODO QUE CRIA DOIS USUÁRIOS DE PAPEL User

public static async Task SeedUsersAsync(IServiceProvider serviceProvider)

{

//OBJETOS UserManager E RoleManager - GESTÃO DE USUÁRIOS E PAPÉIS

var userManager = serviceProvider.GetRequiredService<UserManager<IdentityUser>>();

var roleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();

// Verifica e cria a função "User", se necessário

// CRIA O PAPÉL DE User, CASO AINDA NÃO EXISTA

const string userRole = "User";

if (!await roleManager.RoleExistsAsync(userRole))

{

await roleManager.CreateAsync(new IdentityRole(userRole));

}

// COM A FUNÇÃO CreateUserAsync - CRIA DOIS USUÁRIOS ESPECÍFICOS

await CreateUserAsync(userManager, "alice@smith.com", "Password@123", userRole);

await CreateUserAsync(userManager, "bob@smith.com", "Password@123", userRole);

}

//MÉTODO QUE CRIA USUÁRIOS-DADO O OBJETO UserManager, EMAIL, SENHA E PAPÉL

private static async Task CreateUserAsync(UserManager<IdentityUser> userManager, string email, string password, string role)

{

// SE O USUÁRIO JÁ EXISTE (VERIFICADO PELO E-MAIL) NÃO CRIA

if (await userManager.FindByEmailAsync(email) != null)

{

return;

}

//SE NÃO EXISTE..

var user = new IdentityUser

{

UserName = email,

Email = email,

EmailConfirmed = true // Para evitar a necessidade de confirmação de email

};

// Cria o usuário

var result = await userManager.CreateAsync(user, password);

if (result.Succeeded)

{

// Atribui a função ao usuário

await userManager.AddToRoleAsync(user, role);

}

else

{

throw new Exception($"Erro ao criar usuário {email}: {string.Join(", ", result.Errors)}");

}

}

}

}

Program->Linha22 - builder.Services.AddDefaultIdentity.... Fazer com que a aplicação seja capaz de usar o IdentitySeeder.cs para adicionar papéis aos usuários

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>();

Program-> Linha 53 e abaixo - Criar um escopo ServiceProvider que vai ser capaz de injetar dependências entre as classes do projeto

using (var scope = app.Services.CreateScope())

{

var services = scope.ServiceProvider;

try

{

await IdentitySeeder.SeedUsersAsync(services);

}

catch (Exception ex)

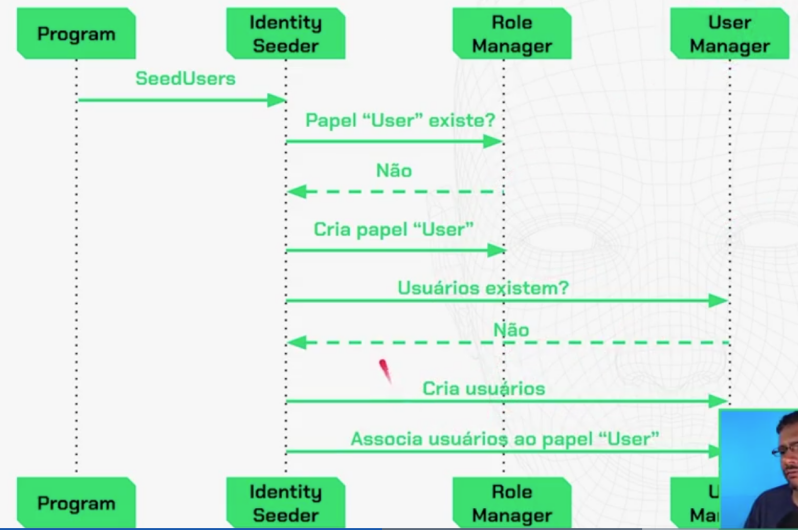
{

Console.WriteLine($"Erro ao executar o Seeder: {ex.Message}");

}

}

Diagrama de sequência - Criação de usuários teste



Como consultar as informações inseridas na BD?

Botão direito em vollmed.db -> abrir com DB Browser for SQLite-> Browse Data->

Assim é possível consultar nas tabelas AspNetUsers e AspNetRoles os usuários e papéis criados pela última execução.

Lembrando que: considerando como o código foi escrito, caso seja executado novamente, não vai duplicar os usuários/papéis ou gerar erro, dado que, caso já existam os registros, ele simplesmente não faz nada.

**CSRF - CROS-SITE REQUEST FORGERY**

Tipo de ataque onde o hacker se aproveita da sessão em aberto do usuário para acessar métodos que precisem de autenticação. No momento, a aplicação não tem proteção contra este tipo de ataque, tanto que, se o método de adicionar médico for acionado, pelo Postman, ele funciona (o que simula um uso indevido de sessão)

Como evitar?

Em Program->Linha 32 - var app = builder.Build(); - escrever acima

//MIDDLEWARE CONTRA ATAQUES CSRF

//

builder.Services.AddAntiforgery(options =>

{

options.Cookie.Name = "VollMed.AntiForgery"; // Nome personalizado do cookie

options.Cookie.HttpOnly = true; // Evitar acesso via JavaScript

options.HeaderName = "X-CSRF-TOKEN"; // Cabeçalho personalizado para APIs

});

Em Controllers->MedicoController - Mudar o método SalvarAsync para usar o token de autenticação do cookie da sessão, ou seja, evitar acessos externos não autenticados

//MÉTODO QUE ADICIONA UM MÉDICO NA APLICAÇÃO

//VALIDAÇÃO DO COOKIE COM TOKEN DE AUTENTICAÇÃO

//COOKIE CONFIGURADO EM PROGRAM->builder.Services.AddAntiforgery(options....

[ValidateAntiForgeryToken]

[HttpPost]

[Route("")]

public async Task<IActionResult> SalvarAsync([FromForm] MedicoDto dados)

{...

**CONTROLE DE AUTORIZAÇÃO**

Como controlar o acesso aos métodos de MedicoController?

Por exemplo, na homepage da aplicação, clicando em médicos, qualquer pessoas, sem autenticação, consegue visualizar e editar os dados dos médicos.

Isso pode ser remediado com a definição do atributo [Authorize] para todos os métodos da classe pertinente

No exemplo do projeto, será definido o atributo [Authorize] para todos métodos das classes MedicoController e ConsultaController

Controllers->MedicoController

[Authorize]

[Route("medicos")]

public class MedicoController : BaseController ...

Controllers->ConsultaController

[Authorize]

[Route("consultas")]

public class ConsultaController : BaseController...

Program->Linha 38 - Acima de var app = builder.Build()

//CONTROLE DE AUTORIZAÇÃO - [Authorize] NO INÍCIO DA CLASSE

builder.Services.AddAuthorization();

Program->Linha 56 - Acima de app.UseAuthorization

//USO DE UM MIDDLEWARE DE AUTENTICAÇÃO NO PIPELINE DE REQUISIÇÕES

app.UseAuthentication();

Program-> Linha 63- Abaixo de app.MapControllerRoute....

//HABILITA AS PÁGINAS DE LOGIN E CADASTRO - FEITO NO SCAFOLDING

//E HABILITA O USO DESSES RECURSOS NOS ARQUIVOS ESTÁTICOS DO ASP.NET CORE IDENTITY - CSS, JS

app.MapRazorPages().WithStaticAssets();

Feito isso, agora quando, a partir da homepage, o usuário acessar a aba médicos, ao invés de permitir o acesso, ele será redirecionado para a página de login.

\*Caso o login seja feito, como ainda não foi implementado o logout, para fazer o logout, acessar [https://localhost:7058/Identity/Account/Logout](https://localhost:/7058/Identity/Account/Logout)

Ainda pensando em controlar acesso, para que a homepage do site seja aberta a todos, sem uso de autenticação, caso seja necessário explicitar isso, um atributo [AllowAnonymous] pode ser utilizado no início da classe

[AllowAnonymous]//PERMITE ACESSO AOS MÉTODOS DESSA CLASSE SEM AUTENTICAÇÃO

public class HomeController : Controller

{...

\*Sem definir, por padrão, já permite acesso sem autenticação

**CONFIRMAÇÃO DE E-MAIL PARA CADASTRO**

Na aplicação, pode ser configurado a exigência do usuário confirmar o cadastro na sua caixa de entrada de e-mail. Como a aplicação não tem emissor de e-mail, ela simula

Program->Linha 25 - abaixo de builder.Services.AddDefaultIdentity<IdentityUser...

//CONFIRMAÇÃO DE E-MAIL PARA CADASTRAR USUÁRIO

//EM TEORIA, O USUÁRIO PRECISARIA ACESSAR SUA CAIXA DE ENTRADA PARA CONFIRMAR O E-MAIL E CADASTRAR

//MAS COMO A APLIAÇÃO NÃO TEM UM EMISSOR DE E-MAIL, ELA MESMA OFERECE A OPÇÃO DE CONFIRMAR

builder.Services.Configure<IdentityOptions>(options =>

{

options.SignIn.RequireConfirmedEmail = true; // Exigir e-mails confirmados para login

options.SignIn.RequireConfirmedPhoneNumber = false; // Não exigir confirmação de número de telefone

});

**BLOQUEIO DE ACESSO POR 2MIN APÓS ERRAR A SENHA 3X**

Program-> Linha 25 - abaixo de ....options.SignIn.RequireConfirmedEmail...

//BLOQUEIO DE TENTATIVA DE ACESSO POR 2 MINUTOS APÓS ERRAR A SENHA 3X

builder.Services.Configure<IdentityOptions>(options =>

{

options.Lockout.AllowedForNewUsers = true;

options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(2);

options.Lockout.MaxFailedAccessAttempts = 3;

});

Areas->Identity->Pages->Account->Login.html->Login.cshtml.cs

Linha 114 - dentro do método OnPostAsync, que é responsável por efetuar o login, mudar um atributo da função PasswordSignInAsync, que é lockoutOnFailure, passando valor true

var result = await \_signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: true);

Sendo assim, se o usuário errar a senha 3x, é bloqueado por 2min.

É possível confirmar até quando o usuário fica bloqueado na tabela AspNetUsers, no campo LockOutEnd.

**MELHORIAS NA SEGURANÇA PARA COOKIES E SENHAS**

Program->Linha 44 - Abaixo de ...options.Lockout.MaxFailedAccessAttempts = 3;...

//CONFIGURAÇÕES ADICIONAIS PARA A SENHA DE LOGIN

builder.Services.Configure<IdentityOptions>(options =>

{

options.Password.RequireDigit = true; // Exigir pelo menos um número

options.Password.RequireLowercase = true; // Exigir pelo menos uma letra minúscula

options.Password.RequireUppercase = true; // Exigir pelo menos uma letra maiúscula

options.Password.RequireNonAlphanumeric = true; // Exigir caracteres especiais

options.Password.RequiredLength = 8; // Tamanho mínimo da senha

});

//CONFIGURAÇÕES ADICIONAIS PARA O COMPORTAMENTO DA SESSÃO / COOKIE

builder.Services.ConfigureApplicationCookie(options =>

{

options.LoginPath = "/Identity/Account/Login"; // Redireciona para login se não autenticado

options.LogoutPath = "/Identity/Account/Logout"; // Caminho para logout

options.AccessDeniedPath = "/Identity/Account/AccessDenied"; // Caminho para acesso negado

options.ExpireTimeSpan = TimeSpan.FromMinutes(2); // Tempo de expiração da sessão em caso de inatividade

options.SlidingExpiration = true; // Renova o cookie sempre que houver atividade

options.Cookie.HttpOnly = true; // Impede acesso via JavaScript

options.Cookie.SecurePolicy = CookieSecurePolicy.Always; // Exige HTTPS

options.Cookie.SameSite = SameSiteMode.Strict; // Restringe envio desse cookie para outro site fora da aplicação

});

//MAIS CONFIGURAÇÕES ADICIONAIS PARA COOKIES

//AddSession - MIDDLEWARE PARA A PIPELINE DA EXECUÇÃO DO .NET CORE PARA A SESSÃO

builder.Services.AddSession(options =>

{

options.Cookie.HttpOnly = true;//NÃO PODE SER ACESSADO VIA JS

//APLICAÇÃO FORÇA O USO DE COOKIE, MESMO SEM O USUÁRIO CONCORDAR

options.Cookie.IsEssential = true;

options.Cookie.SecurePolicy = CookieSecurePolicy.Always; //SESSÃO EXIGE HTTPS

//1 MIN DE TEMPO PARA QUALQUER INFORMAÇÃO ADICIONADA NA SESSÃO, CASO NÃO HAJA NENHUM MECANISMO DE RENOVAÇÃO

options.IdleTimeout = TimeSpan.FromMinutes(1);

});

Program-> Linha 103 - abaixo de var app = builder.Build();

//USA O QUE ESTIVER DEFINIDO EM builder.Services.AddSession....

app.UseSession();

**CONFIGURAR A APLIAÇÃO PARA MOSTRAR NO RODAPÉ DO SITE O IDENTIFICADOR DA SESSÃO**

No caso da aplicação, sempre que uma sessão for iniciada por um usuário, será exibido no rodapé da página um identificador, que nesse contexto é chamado de **VollMedCard**, um número identificador de cada usuário (só exemplo, na verdade o identificado gerado para esse teste é o mesmo para todos usuários)

Areas->Identity->Pages->Account->Login.cshtml->Login.cshtml.cs - Linha 117 - abaixo de if (result.Succeeded}...

HttpContext.Session.SetString("VollMedCard", "1234.4567.7890.1234");

Controllers->BaseController - Linha 12 - abaixo de ViewData["Especialidades"]...

//MOSTRAR NA VIEW O DADO DA VARIÁVEL DE SESSÃO VollMedCard

ViewData["VollMedCard"] = HttpContext.Session.GetString("VollMedCard");

Views->Shared->\_footer.cshtml -Linha 2 - adicionar tag div para que o footer mostre também os dados do VollMedCard

<div class="container">

<span class="text-center">VollMed Card: <b>@ViewData["VollMedCard"]</b></span>

</div>

**ATAQUE DE SESSÃO**

Caso um atacante consiga sequestrar os cookies da sessão para que possa acessar indevidamente as funcionalidades do sistema que exigem esses autenticação de cookies, como evitar?

Nesse caso podem ser implementadas duas medidas de segurança: sempre limpar os dados da sessão residual no momento do login, além de refazer o login em segundo plano com as informações atuais (reautenticação transparente) e exigir que o IP atual da requisição seja o mesmo da sessão, ou seja, o mesmo usuário/computador no momento do login é o que usa as funções do sistema.

**LIMPANDO DADOS DA SESSÃO RESIDUAL**

Inserir no construtor da classe LoginModel o parâmetro que gerencia usuários, nesse caso, userManager

Areas->Identity->Pages->Account->Login.cshtml->Login.cshtml.cs

private readonly UserManager<IdentityUser> \_userManager;

public LoginModel(SignInManager<IdentityUser> signInManager, ILogger<LoginModel> logger, UserManager<IdentityUser> userManager)

{

\_signInManager = signInManager;

\_logger = logger;

\_userManager = userManager;

}

Limpar dados da sessão e reemitir o cookie de autenticação

Areas->Identity->Pages->Account->Login.cshtml->Login.cshtml.cs - linha 119 abaixo de if (result.Succeeded)

//LIMPEZA DE DADOS DA SESSÃO RESIDUAL

HttpContext.Session.Clear();

//REEMISSÃO DO COOKIE DE AUTENTICAÇÃO APÓS LOGIN

var user = await \_userManager.FindByEmailAsync(Input.Email);

await \_signInManager.SignOutAsync();

await \_signInManager.SignInAsync(user, Input.RememberMe);

**COMPARAÇÃO DE IPs DA SESSÃO E DA REQUISIÇÃO**

Criar um construtor para a classe BaseController, com o atributo de tipo objeto SignInManager (gestão de usuários)

Controllers->BaseController

private readonly SignInManager<IdentityUser> \_signInManager;

//CONSTRUTOR DA CLASSE

public BaseController(SignInManager<IdentityUser> signInManager)

{

\_signInManager = signInManager;

}

Adicionar método que compara o IP atual da requisição e o IP da sessão

Controllers->BaseController - Final da classe - linha 40

//MÉTODO QUE COMPARA O IP ATUAL COM O IP DA SESSÃO - PARA EVITAR ATAQUE DE IPs

//EXTERNOS À SESSÃO COM COOKIES SEQUESTRADOS

//MÉTODO USADO EM OnActionExecuting

private bool CheckSessionSecurity()

{

var currentIp = HttpContext.Connection.RemoteIpAddress.ToString();

var sessionIp = HttpContext.Session.GetString("IpAddress");

if (sessionIp != null && sessionIp != currentIp)

{

HttpContext.Session.Clear();

\_signInManager.SignOutAsync();

return false;

}

HttpContext.Session.SetString("IpAddress", currentIp);

return true;

}

Adicionar ao método OnActionExecuting (método executado sempre que um método dos controllers é acessado) a chamada ao método CheckSessionSecurity, fazendo com que, caso o acesso venha de um IP diferente do IP da sessão, o site é redirecionado para o login, já \_deslogado\_ .

//MÉTODO EXECUTADO SEMPRE QUE UMA REQUISIÇÃO É FEITA AOS CONTROLLERS

public override void OnActionExecuting(ActionExecutingContext context)

{

//SE FOR UMA REQUISIÇÃO DE UM IP DIFERENTE DO IP DA SESSÃO, REDIRECIONA

//PARA LOGIN - JÁ DESLOGADO

if (!CheckSessionSecurity())

{

context.Result = new RedirectResult("./Login");

}...

Agora com um novo construtor que usa um novo atributo na classe BaseController, é preciso herdar esse atributo nas classes que herdam de BaseCotroller, que no caso são MedicoController e ConsultaController

Controllers->MedicoController - mudar construtor para herdar atributo signInManager de BaseController

public MedicoController(IMedicoService service, SignInManager<IdentityUser> signInManager)

: base(signInManager)

{

\_service = service;

}

Controllers->ConsultaController - mudar construtor para herdar atributo signInManager de BaseController

public ConsultaController(IConsultaService consultaService, IMedicoService medicoService, SignInManager<IdentityUser> signInManager)

: base(signInManager)

{

\_consultaservice = consultaService;

\_medicoService = medicoService;

}

Com isso, sempre que forem feitas requisições ao sistema, o IP salvo da sessão e o IP atual da requisição serão comparados, evitando ataques de IPs externos.

**VALIDAÇÃO DE DADOS**

Validar os dados é importante para a proteção e o devido funcionamento da aplicação. Isso pode ser feito por meio de Data Annotations, aplicadas aos atributos das classes DTOs

Dtos->MedicoDto-Nos atributos

public long? Id { get; set; }

public string \_method { get; set; }

[Required(ErrorMessage = "Campo obrigatório")]

[MinLength(5, ErrorMessage = "Campo deve ter no mínimo 5 caracteres")]

public string Nome { get; set; }

[Required(ErrorMessage = "Campo obrigatório")]

[EmailAddress]

public string Email { get; set; }

[Required(ErrorMessage = "Campo obrigatório")]

[StringLength(6, MinimumLength =4,

ErrorMessage ="Campo deve ter de 4 a 6 dígitos numéricos")]

public string Crm { get; set; }

[Required(ErrorMessage = "Campo obrigatório")]

[RegularExpression(@"^(?:\d{8}|\d{9}|\d{4}-\d{4}|\d{5}-\d{4}|**\(**\d{2}**\)**\s\*\d{4}-\d{4}|**\(**\d{2}**\)**\s\*\d{5}-\d{4}|**\(**\d{2}**\)**\s\*\d{9})$", ErrorMessage = "Telefone inválido")]

public string Telefone { get; set; }

[Required(ErrorMessage = "Campo obrigatório")]

public Especialidade Especialidade { get; set; }

Utilização dessa validação em Controllers->MedicoController - linha 61 - abaixo de if (dados.\_method == "delete"...

//AO ENVIAR OS DADOS DO MÉDICO PARA CADASTRO, SE NÃO FOR UM OBJETO DTO VÁLIDO

//MOSTRA NOVAMENTE O FORMULÁRIO

if (!ModelState.IsValid)

{

return View(PaginaCadastro, dados);

}

Mudar a view do formulário de cadastro de médico para destacar quais campos estão inválidos. Isso é feito adicionando a tag <**span** **asp-validation-for**="X" class="text-danger"></**span**>

para cada tag que contém um campo que contenha validação

Views->Medico->Formulario.cshtml

<**form** method="post" **asp-action**="" **asp-controller**="Medicos">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<input type="hidden" name="\_method" value="post">

<**input** **type**="hidden" **asp-for**="Id" />

<div class="form-group">

<**label** **asp-for**="Nome">Nome:</**label**>

<**input** **asp-for**="Nome" class="form-control" />

<**span** **asp-validation-for**="Nome" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Email">Email:</**label**>

<**input** **asp-for**="Email" class="form-control" />

<**span** **asp-validation-for**="Email" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Telefone">Telefone:</**label**>

<**input** **asp-for**="Telefone" class="form-control" placeholder="( ) -" />

<**span** **asp-validation-for**="Telefone" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Crm" class="control-label">CRM:</**label**>

<**input** **asp-for**="Crm" class="form-control" />

<**span** **asp-validation-for**="Crm" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Especialidade">Especialidade</**label**>

<**select** **asp-for**="Especialidade" class="form-control" >

<**option** **value**="">[SELECIONE UMA ESPECIALIDADE]</**option**>

@foreach (var especialidade in especialidades)

{

<**option** **value**="@especialidade">

@especialidade.GetDisplayName()

</**option**>

}

</**select**>

<**span** **asp-validation-for**="Especialidade" class="text-danger"></**span**>

</div>

<div class="buttons">

<button type="submit" class="btn btn-primary">

<img src="/assets/plus.png" alt="Ícone de adicionar" class="btn-icon">Cadastrar

</button>

<a href="@Url.Action("Listagem", "Medicos" )" class="btn btn-secondary">

<**img** src="~/assets/back.png" alt="Voltar" class="btn-icon">Voltar

</a>

</div>

</**form**>

**PROTEÇÃO CONTRA XSS (CROSS-SITE SCRIPTING) E SNIFFING DE MIME TYPE**

Cross-site Scripting - Execução de trechos de código de fontes externas à aplicação

Scripting in line - Execução de trechos de códigos maliciosos inseridos em entradas da aplicação.

Sniffing de MIME Type - Execução de códigos maliciosos disfarcçados como um arquivo de formato comum (png, pdf, jpg) e que na verdade é um script aguardando para ser executado.

Como evitar?

Program-> linha 152 - acima da última linha app.Run()

// Middleware para adicionar cabeçalhos de segurança contra:

// 1. XSS (Cross-Site Scripting):

// 2. Script in line

// 3. Sniffing de MIME Type:

app.Use(async (context, next) =>

{

//SÓ ACEITA SCRIPTS DO PRÓPRIO SITE - para evitar XSS.

context.Response.Headers.Add("Content-Security-Policy", "default-src 'self'; script-src 'self';");

// Previne a interpretação incorreta de MIME types.

context.Response.Headers.Add("X-Content-Type-Options", "nosniff");

await next();

});

**RESTRINGINDO ACESSO PELO PAPEL DO USUÁRIO**

Tipos de usuários diferentes precisam acessar funcionalidades distintas da aplicação. Para definir isso, primeiro vamos estabelecer que no cabeçalho da página sempre haja as opções de login / logout e “bem vindo usuário”. Isso é feito:

Adicionando o arquivo Views->Shared->\_LoginPartial.cshtml

@using Microsoft.AspNetCore.Identity

@inject SignInManager<IdentityUser> SignInManager

@inject UserManager<IdentityUser> UserManager

<ul class="navbar-nav">

@if (SignInManager.IsSignedIn(User))

{

<li class="nav-item">

<**a** id="manage" class="nav-link text-dark" **asp-area**="Identity" **asp-page**="/Account/Manage/Index" title="Manage">Hello @UserManager.GetUserName(User)!</**a**>

</li>

<li class="nav-item">

<**form** id="logoutForm" class="form-inline" **asp-area**="Identity" **asp-page**="/Account/Logout" **asp-route-returnUrl**="@Url.Action("Index", "Home", new { area = "" })">

<button id="logout" type="submit" class="nav-link btn btn-link text-dark border-0">Logout</button>

</**form**>

</li>

}

else

{

<li class="nav-item">

<**a** class="nav-link text-dark" id="register" **asp-area**="Identity" **asp-page**="/Account/Register">Register</**a**>

</li>

<li class="nav-item">

<**a** class="nav-link text-dark" id="login" **asp-area**="Identity" **asp-page**="/Account/Login">Login</**a**>

</li>

}

</ul>

Aplicando essa mudança para todos as páginas da aplicação

Views->Shared->\_Layout.cshtml - linha 36 entre as tags div

<**partial** **name**="\_LoginPartial"/>

Tornar Alice a admin da aplicação

Data->IdentitySeeder - Linha 24 antes de fechar o método SeedUsersAsync

//CRIA O PAPEL Admin CASO AINDA NÃO EXISTA

const string adminRole = "Admin";

if(! await roleManager.RoleExistsAsync(adminRole))

{

await roleManager.CreateAsync(new IdentityRole(adminRole));

}

//TORNA ALICE ADMIN DA APLICAÇÃO

IdentityUser? alice = await userManager.FindByEmailAsync("alice@smith.com");

IList<IdentityUser> admins = await userManager.GetUsersInRoleAsync(adminRole);

if (!admins.Any(a => a.Email == alice.Email))

{

await userManager.AddToRoleAsync(alice, adminRole);

}

Dadas as mudanças, agora iremos definir que as opções de editar e excluir médicos apareça somente para os usuários admin

Views->Medico->Listagem.cshtml - Linhas 35 até 44 - onde estão as tags dos botões excluir e editar

@if (User.IsInRole("Admin"))

{

<a href="@Url.Action("Formulario", "Medicos" , new { id=medico.Id })" class="btn btn-edit" title="Editar">

<**img** src="~/assets/edit.svg" alt="Editar">

</a>

<a href="#deleteModal" class="btn btn-delete" data-toggle="modal" data-id="@medico.Id" data-url="/medicos" title="Excluir">

<**img** src="~/assets/delete.svg" alt="Excluir">

</a>

}

E também definir que somente admins possam cadastrar novos médicos

Controllers->MedicoController-> Nos atributos [] do método

//SOMENTE USUÁRIOS Admin TEM ACESSO A ESSE MÉTODO

[Authorize(Roles = "Admin")]

**UTILIZANDO CLAIMS PRA MELHORAR A EXPERIÊNCIA DO USUÁRIO**

Criar claims para os dois usuários cadastrados, bob e alice

Data->IdentitySeeder - linha 40 ao final do método SeedUserAsync

//CLAIMS PARA ALICE

IList<Claim> userClaims = await userManager.GetClaimsAsync(alice);

await userManager.RemoveClaimsAsync(alice, userClaims);

await userManager.AddClaimAsync(alice, new Claim("FullName", "Alice Smith"));

await userManager.AddClaimAsync(alice, new Claim("Role", "Admin"));

//CLAIMS PARA BOB

userClaims = await userManager.GetClaimsAsync(bob);

await userManager.RemoveClaimsAsync(bob, userClaims);

await userManager.AddClaimAsync(bob, new Claim("FullName", "Bob Smith"));

Com essas claims definidas, fazer com que o cabeçalho de todas as páginas apareça o nome completo do usuário logado

Views->Shared->\_LoginPartial - linha 3, abaixo de @inject UserManager<IdentityUser> UserManager

@{

var fullNameClaim = User.Claims.FirstOrDefault(c => c.Type == "FullName");

string? userName = fullNameClaim?.Value ?? @UserManager.GetUserName(User);

}

Views->Shared->\_LoginPartial - linha 15, dentro da tag a

<li class="nav-item">

<**a** id="manage" class="nav-link text-dark" **asp-area**="Identity" **asp-page**="/Account/Manage/Index" title="Manage">Hello, @userName!</**a**>

</li>

**RESTRINGINDO ACESSO PELO PAPEL DO USUÁRIO - USANDO CLAIMS**

Assim como já foi feito anteriormente, é necessário restringir certos acessos a depender do papel do usuário. Agora isso vai ser implementado usando claims.

Restringir que somente usuários admin possam cadastrar novos médicos

Views->Medico->Listagem.cshtml

Linha 3 - @using System.Security.Claims;

Linha 12 - div com conteúdo Novo médico - envelopar com a condicional

@if(roleClaim?.Value == "Admin")

{

<div class="table-controls">

<a href="@Url.Action("Formulario", "Medicos" )" class="btn btn-tertiary">

<**img** src="~/assets/plus.png" alt="Ícone de adicionar" class="btn-icon">

Novo Médico

</a>

</div>

}