WEB API - UsuariosApi

**DOWNLOADS NECESSÁRIOS**

VisualStudio Community 2022

.NET 6.0.402

**Baixar**->Ferramentas->Gerenciador de Pacotes do NuGet->Gerenciar Pacotes NuGet para Solução->

**INÍCIO DO PROJETO - ESTRUTURA BASE**

Models->Usuario

Data->Dtos->CreateUsuarioDto

Data->UsuarioDbContext

Profile->UsuarioProfile

**Baixar**->Ferramentas->Gerenciador de Pacotes do NuGet->Gerenciar Pacotes NuGet para Solução->

AutoMapper.Extensions.Microsoft.DependencyInjection (v.12.0.0)

Controllers->UsuarioController

**CONFIGURAÇÃO IDENTITY E BD**

**Baixar**->Ferramentas->Gerenciador de Pacotes do NuGet->Gerenciar Pacotes NuGet para Solução->

Microsoft.AspNetCore.Identity.EntityFrameworkCore (v.6.0.14)

Microsoft.EntityFrameworkCore.Tools (v.6.0.14)

Microsoft.Extensions.Identity.Stores (v.6.0.14) MUDAR NOTEBOOK

Pomelo.EntityFrameworkCore.MySql (v.6.0.2)

Program-> linha 7 abaixo

//VARIÁVEL PARA DEFINIÇÕES ABAIXO

var connectionString = builder.Configuration.GetConnectionString("UsuarioConnection");

builder.Services.AddDbContext<UsuarioDbContext>(opts =>

opts.UseMySql(connectionString,

ServerVersion.AutoDetect(connectionString)));

//ADICIONANDO Identity AO PROJETO - RELACIONADO COM USUARIO E SEU PAPEL

builder.Services

.AddIdentity<Usuario, IdentityRole>()

.AddEntityFrameworkStores<UsuarioDbContext>() //IDENTITY CUIDA DA CONEXÃO COM BD ATRAVÉS DE UsuarioDbContext

.AddDefaultTokenProviders();//AUTENTICAÇÃO

appsettings.json-> linha 8 abaixo

"ConnectionStrings": {

"UsuarioConnection" : "server=localhost;database=usuariodb;user=root;password=root"

}

Aplicar mudanças a BD

Ferramentas-> Gerenciador de pacotes do Nuget->Console do Gerenciador de Pacotes

Add-Migration “Criando Usuarios”

Update-Database

UsuarioController->AdicionaUsuario

using AutoMapper;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using UsuariosApi.Data.Dtos;

using UsuariosApi.Models;

namespace UsuariosApi.Controllers

{

[ApiController]

[Route("[Controller]")]

public class UsuarioController : ControllerBase

{

//PROPRIEDADES

private IMapper \_mapper; //AutoMap

private UserManager<Usuario> \_userManager;//GERENCIA MÉTODOS DO IDENTITY PARA USUÁRIO

//CONSTRUTOR

public UsuarioController(IMapper mapper, UserManager<Usuario> userManager)

{

\_mapper = mapper;

\_userManager = userManager;

}

//DEMAIS MÉTODOS

/// <summary> MÉTODO QUE ADICIONA UM USUÁRIO AO BD </summary>

/// <param name="dto">OBJETO DTO QUE RECEBE AS INFORMAÇÕES DE CRIAÇÃO</param>

/// <returns> Task<IActionResult> </returns>

/// <response code="201"> Em caso de inserção bem sucedida </response>

[HttpPost]

public async Task<IActionResult> CadastraUsuario(CreateUsuarioDto dto)

{ //async Task - RETORNO DE MÉTODO await É TIPO async Task<>

Usuario usuario = \_mapper.Map<Usuario>(dto);

//CRIAÇÃO USUÁRIO IDENTITY (USUÁRIO E SENHA)

//SENHA VEM DO OBJETO dto NESSE CASO, JÁ QUE OBJETO usuario NÃO TEM CAMPO SENHA

//await - INDICA QUE O PROGRAMA AGUARDA O USUÁRIO SER CRIADO PARA SALVAR EM resultado

IdentityResult resultado = await \_userManager.CreateAsync(usuario, dto.Password);

//EM CASO DE SUCESSO

if(resultado.Succeeded) return Ok("Usuário cadastrado!"); //201 CODE

//EM CASO DE FALHA

throw new ApplicationException("Falha ao cadastrar usuário");

}

}

}

Program->Configurar AutoMapper

//AUTOMAPPER

builder.Services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

Tabela de usuários acessível pelo WorkBench em BD usuarioDB, tabela aspnetusers

Método de adicionar usuário acessível via Postman url

Até o momento: url por https não funciona.

\*Lembrar que: O Identity traz mecanismos de segurança - senha fraca (sem maiúsculo, caractere especial, numero), campo de confirmação de senha inválido,....

Agora, será feita a mudança da lógia de adicionar um usuário como serviço, e não como uma lógica toda dentro de UsuarioController.

Criar pasta Services->CadastroService.cs->

using Microsoft.AspNetCore.Identity;

using UsuariosApi.Data.Dtos;

using UsuariosApi.Models;

using AutoMapper;

namespace UsuariosApi.Services

{

public class CadastroService

{

//ATRIBUTOS

private IMapper \_mapper; //AutoMap

private UserManager<Usuario> \_userManager;//GERENCIA MÉTODOS DO IDENTITY PARA USUÁRIO

//CONSTRUTOR

public CadastroService(IMapper mapper, UserManager<Usuario> userManager)

{

\_mapper = mapper;

\_userManager = userManager;

}

//DEMAIS MÉTODOS

public async Task Cadastra(CreateUsuarioDto dto)

{

Usuario usuario = \_mapper.Map<Usuario>(dto);

//CRIAÇÃO USUÁRIO IDENTITY (USUÁRIO E SENHA)

//SENHA VEM DO OBJETO dto NESSE CASO, JÁ QUE OBJETO usuario NÃO TEM CAMPO SENHA

//await - INDICA QUE O PROGRAMA AGUARDA O USUÁRIO SER CRIADO PARA SALVAR EM resultado

IdentityResult resultado = await \_userManager.CreateAsync(usuario, dto.Password);

//EM CASO DE FALHA

if (!resultado.Succeeded)

{

throw new ApplicationException("Falha ao cadastrar usuário");

}

}

}

}

E deixar Controllers->UsuarioController como:

[ApiController]

[Route("[Controller]")]

public class UsuarioController : ControllerBase

{

//PROPRIEDADES

private CadastroService \_cadastroService;//SERVIÇO QUE CONTÉM A LÓGIDA DO CADASTRO

//CONSTRUTOR

public UsuarioController(CadastroService cadastroService)

{

\_cadastroService = cadastroService;

}

//DEMAIS MÉTODOS

[HttpPost]

public async Task<IActionResult> CadastraUsuario(CreateUsuarioDto dto)

{ //async Task - RETORNO DE MÉTODO await É TIPO async Task<>

//LÓGICA DE CADASTRO COMO SERVIÇO EM Services->CadastroService->Cadastra()

await \_cadastroService.Cadastra(dto);

return Ok("Usuário cadastrado!"); //200 CODE

}

}

}

Para que o .Net entenda que está sendo definido o funcionamento de um controller com um serviço, precisa ser configurado em Program, após builder de automapper:

//CONFIGURADO SERVICE Services->CadastroService

builder.Services.AddScoped<CadastroService>();

Nesse momento, vamos usar o *AddScoped*assim o CadastroService sempre vai ser instanciado quando houver uma requisição nova que demande uma instância de CadastroService. Com isso, se fazemos uma requisição nova e chamamos o CadastroService, vamos instanciar um novo.

Se utilizássemos o *AddSingleton*seria um único CadastroService para todas as requisições que chegassem, ou seja, seria a mesma instância.

Já o *AddTransient*vai fazer sempre uma instância nova, mesmo que seja na mesma requisição. Assim, se chega uma requisição e precisamos de uma instância de CadastroService, vamos instanciar uma nova. Mas, se chega uma requisição e precisamos do CadastroService, também vai instanciar uma nova.

Após essas configurações, o cadastro de um novo usuário pode ser acessado pela mesma url <http://localhost:5076/usuario>

**LOGIN / AUTENTICAÇÃO**

Renomear CadastroService.cs e sua propriedade em UsuarioController para UsuarioService, dado que, agora, esse serviço incluirá também a lógica do login.

Criar Data->Dtos->LoginUsuarioDto

using System.ComponentModel.DataAnnotations;

namespace UsuariosApi.Data.Dtos

{

public class LoginUsuarioDto

{

[Required]

public string Username { get; set; }

[Required]

public string Password { get; set; }

}

}

Adicionar o método de login à Services->UsuarioServices

//MÉTODO QUE REALIZA O LOGIN DO USUÁRIO

public async Task Login(LoginUsuarioDto dto)

{

var resultado = await \_signInManager.PasswordSignInAsync(dto.Username,

dto.Password,

false, false);

if (!resultado.Succeeded)

{

throw new ApplicationException("Usuário não autenticado");

}

}

Adicionar em Controllers->UsuarioController o método de login (que usa o outro método login de serviços). E também, já diferenciar os dois métodos que usam o HttpPost, pelas rotas.

//DEMAIS MÉTODOS

//MÉTODO QUE UTILIZA SERVIÇO UsuarioService->Cadastra PARA CADASTRAR USUÁRIO

[HttpPost("cadastro")]

public async Task<IActionResult> CadastraUsuario(CreateUsuarioDto dto)

{ //async Task - RETORNO DE MÉTODO await É TIPO async Task<>

//LÓGICA DE CADASTRO COMO SERVIÇO EM Services->CadastroService->Cadastra()

await \_usuarioService.Cadastra(dto);

return Ok("Usuário cadastrado!"); //200 CODE

}

//MÉTODO QUE UTILIZA SERVIÇO UsuarioService->Login PARA LOGAR USUÁRIO

[HttpPost("login")]

public async Task<IActionResult> Login(LoginUsuarioDto dto)

{

await \_usuarioService.Login(dto);

return Ok("Usuário autenticado");

}

Agora temos duas rotas

<http://localhost:5076/usuario/cadastro> Cadastro de usuário

http://localhost:5076/usuario/login Login de usuário

**JSON WEB TOKEN**

Uma sequência de caractéres gerado por encoding hs256. São uma maneira de garantir ao sistema que o usuário está logado, exigindo esse token, gerado no momento do login, para acessar os métodos devidos.

**Baixar**->Ferramentas->Gerenciador de Pacotes do NuGet->Gerenciar Pacotes NuGet para Solução->

System.IdentityModel.Tokens.Jwt (v.6.27.0)

Services->UsuarioService

//MÉTODO QUE REALIZA O LOGIN DO USUÁRIO

public async Task<string> Login(LoginUsuarioDto dto)

{ //async Taks<> - POR CAUSA DO await

//string - POR QUE O TOKEN RETORNADO É STRING

var resultado = await \_signInManager.PasswordSignInAsync(dto.Username,

dto.Password,

false, false);

//LOGIN FALHOU

if (!resultado.Succeeded)

{

throw new ApplicationException("Usuário não autenticado");

}

//LOGIN OK

//CONSULTAR QUAL O USUÁRIO ATUAL - PELO Username (MAIÚSCULO)

//Username é único? (Identity GARANTE)

var usuario = \_signInManager

.UserManager

.Users

.FirstOrDefault(user => user.NormalizedUserName == dto.Username.ToUpper());

//OBTER O TOKEN DE ACESSO PARA ESSE USUÁRIO

var token = \_tokenService.GenerateToken(usuario);

//RETORNÁ-LO

return token;

}

Services->TokenService

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using UsuariosApi.Models;

namespace UsuariosApi.Services

{

public class TokenService

{

//PROPRIEDADES

//CONSTRUTOR

//DEMAIS MÉTODOS

//MÉTODO QUE GERA O JSON WEB TOKEN-JWT, DADO UM OBJETO USUARIO

public string GenerateToken(Usuario usuario)

{

//Claim[] - VETOR DE REIVINDICAÇÕES PARA A GERAÇÃO DO JWT

Claim[] claims = new Claim[]

{

new Claim("username", usuario.UserName),

new Claim("id", usuario.Id),//GERADO PELO BD

new Claim(ClaimTypes.DateOfBirth, usuario.DataNascimento.ToString())

//HÁ MUITOS ClaimTypes PARA CADA TIPO DE CAMPO

};

//CHAVE DE GERAÇÃO DAS CREDENCIAIS

var chave = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("luciano"));

//GERAÇÃO DA CHAVE USA UMA SEQUENCIA QUALQUER DE CARACTERES - luciano OU QUALQUER OUTRA COISA

//CREDENCIAIS

var signingCredentials = new SigningCredentials(chave, SecurityAlgorithms.HmacSha256);

//SecurityAlgorithms.HmacSha256 - Algorito de criptografia sha256

//GERAÇÃO JWT COM CREDENCIAIS

var token = new JwtSecurityToken

(

expires: DateTime.Now.AddMinutes(10),//EXPIRA EM 10MIN

claims: claims,//REINVINDICAÇÕES PASSADAS ACIMA

signingCredentials: signingCredentials

);

//RETORNO TOKEN EM FORMA DE STRING

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

Program-> linha 34, abaixo de UsuarioService

//CONFIGURADO SERVICE Services->TokenService

builder.Services.AddScoped<TokenService>();

Controllers->UsuarioController- Método Login

//MÉTODO QUE UTILIZA SERVIÇO UsuarioService->Login PARA LOGAR USUÁRIO

[HttpPost("login")]//ROTA login DIFERENCIA DOS DEMAIS MÉTODOS POST

public async Task<IActionResult> Login(LoginUsuarioDto dto)

{

var token = await \_usuarioService.Login(dto);

return Ok(token);

}

Com essas mudanças, agora, sempre que o login é efetuado com sucesso, é gerado um retorno com o token de acesso. Esse token pode ser *decoded* em <https://jwt.io/>

**UTILIZANDO O TOKEN**

Controllers->AcessoController - cria um método acessível somente se cumprida a política de idade mínima

Programs - configurar a idade mínima como política de acesso

//POLÍTICA DE ACESSO PARA AcessoController - [Authorize(Policy = "IdadeMinima")]

builder.Services.AddAuthorization(options =>

{

options.AddPolicy("IdadeMinima", policy =>

policy.AddRequirements(new IdadeMinima(18))

);

});

Authorization->IdadeMinima - Cria o “modelo” para a idade mínima

Authorization->IdadeAuthorization - Implementa a lógica de comparação entre a informação de idade no token *decoded* e a idade mínima definida em Program

Program - linha 32, abaixo do AutoMapper - Adicionar o serviço de IdadeAuthorization

No postman

GET - <http://localhost:5076/acesso> em Authorization-Beare Token com o token de acesso gerado em qualquer login bem sucedido

Mas, em IdadeAuthorization, dataNascimentoClaim está sendo null, porque ainda falta explicitarmos o uso do token.

**Baixar**->Ferramentas->Gerenciador de Pacotes do NuGet->Gerenciar Pacotes NuGet para Solução->

Microsoft.AspNetCore.Authentication.JwtBearer(v.6.0.14)

Program->Linha 49, acima de de builder.Services.AddAuthorization

//EXPLICITAÇÃO DO FUNCIOAMENTO DO TOKEN

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(options =>

{

options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey //CHAVE DE Services->TokenServices->GenerateToken

(Encoding.UTF8.GetBytes("luciano31646316465431654646465435454654")),

ValidateAudience = false, //MITIGA CASOS DE REDIRECIONAMENTO SE true

ValidateIssuer = false,

ClockSkew = TimeSpan.Zero//ALINHAMENTO RELÓGIO

};

});

Program-> linha 74, abaixo de app.UseHttpsRedirection();

app.UseAuthentication();

Agora em GET - <http://localhost:5076/acesso> em Authorization-Beare Token com o token de acesso gerado em qualquer login bem sucedido a autenticação pelo token está funcional (Com usuário acima de 18 anos - acesso autorizado. Com usuários menores de idade - 404 not found).

**QUESTÕES DE SEGURANÇA - USANDO SECRETS**

Secrets é um recurso do .NET capaz de guardar **localmente** informações sensíveis de um projeto durante o seu desenvolvimento. Por exemplo, para a nossa aplicação atual, a *assymmetric key* usada na geração do token, está exposta em Service->TokenService e Program

Para utilizar o Secrets no projeto, abrir um Git Bash em ...UsuariosApi/UsuariosApi->

***dotnet user-secrets init***

Isso habilita o uso do Secrets no projeto, o que pode ser observado na raiz do projeto (clicar, no VisualStudio, em UsuarioApi), na tag *<UserSecretsId>*

Para criar um Secret para guardar a *Symmetric Key,* no mesmo Git Bash aberto na pasta do projeto

***dotnet user-secrets set “apelido do secret” “valor dele na aplicação”***

No exemplo da nossa aplicação

***dotnet user-secrets set “SymmetricSecurityKey” “.....”***

Isso cria um arquivo json com o apelido e valor do Secret em C:\Users\Luciano\AppData\Roaming\Microsoft\UserSecrets\c576c902-a958-425a-aed4-97440b5fedce

Criar um Secret para as configurações de conexão com a base de dados (em appsettings.json)

***dotnet user-secrets set “ConnectionStrings:UsuarioConnection” “*server=....;database=...;user=..;password=...*”***

Para aplicar os Secrets criados no sistema, mudar nos trechos de código onde ainda estão expostos os valores

Program

var connectionString = builder.Configuration["ConnectionStrings:UsuarioConnection"];

.

.

.

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey //CHAVE DE Services->TokenServices->GenerateToken

(Encoding.UTF8.GetBytes(builder.Configuration["SymmetricSecurityKey"])),

ValidateAudience = false, //MITIGA CASOS DE REDIRECIONAMENTO SE true

ValidateIssuer = false,

ClockSkew = TimeSpan.Zero//ALINHAMENTO RELÓGIO

Service->TokenService

Como a classe TokenService não tem acesso direto ao builder.Configuration(assim como a Program), é preciso criar um construtor que injete esse acesso

//PROPRIEDADES

//PROPRIEDADE DE ACESSO AO builder.Configuration

private IConfiguration \_configuration;

//CONSTRUTOR

//CONSTRUTOR QUE INJETA O ACESSO AO builder.Configuration (para utilização do Secrets)

public TokenService(IConfiguration configuration)

{

\_configuration = configuration;

}

.

.

.

.

//CHAVE DE GERAÇÃO DAS CREDENCIAIS

var chave = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes(\_configuration["SymmetricSecurityKey"]));

//GERAÇÃO DA CHAVE USA UMA SEQUENCIA QUALQUER DE CARACTERES MINIMO 128bits

appsettings.json

?

**ANOTAÇÕES**

**ROTAS**

<http://localhost:5076/usuario/cadastro> Cadastro de usuário

http://localhost:5076/usuario/login Login de usuário

<http://localhost:5076/acesso> Autenticar JWT

//throw new NotImplementedException();