WEB API - UsuariosApi

**DOWNLOADS NECESSÁRIOS**

VisualStudio Community 2022

.NET 6.0.402

**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

Services->TokenService

**ANOTAÇÕES**

//throw new NotImplementedException();