**CURSO .NET6: CRIANDO UMA WEB API**

**DOWNLOADS**

VisualStudio Community 2022

.NET 6.0.402

MySQL Community 8.0.31 (MySQL Server e MySQLWorkbench)

Postman - most recent version

**MACETES**

-Alt + Enter com mouse posicionado sobre a abertura dos colchetes- transforma a estrutura do código para estrutura de arquivo

namespace FilmesApi.Models

{

public class Filme

{

}

}

PARA

namespace FilmesApi.Models;

public class Filme

{

}

-Alt + Enter com mouse posicionado sobre o erro de importação de arquivo - constrói a importação automaticamente

**TEORIA**

**API** - Application Programming Interface

Uma interface de serviço - seguindo suas regras, a API disponibiliza serviços sem que o cliente precise se preocupar em como ela faz isso no servidor desse respectivo serviço.

**Rest** - um tipo de arquitetura de API - um padrão. Uma API que segue esse padrão é chamada de Restful.

**ESTRUTURA FRAMEWORK .NET 6.0**

Program - Onde a aplicação é iniciada, configurações, definições, dependências,...

WeatherForecast.cs - Modelo de API base

Appsettings.Development.json e appsettings.json - definições de bancos de dados, senhas, usuários,...

Controllers -> WeatherForecastController.cs - responsável por receber as requisições do usuário

launchSettings.json -> profiles - define a url da aplicação

**CRIANDO UMA APLICAÇÃO DO 0 - FILMESAPI**

APAGAR Controllers -> WeatherForecastController

APAGAR WeatherForecast

CRIAR Controllers -> FilmeController.cs (classe)

using FilmesApi.Models;

using Microsoft.AspNetCore.Mvc;//USO DO [Route("[controller]")] E [ApiController]

namespace FilmesApi.Controllers

{

[ApiController]

[Route("[controller]")]//DESIGNA A ROTA Filme (VEM DO NOME DA CLASS)

public class FilmeController : ControllerBase

{

//LISTA DE FILMES

private static List<Filme> filmes = new List<Filme>();

//MÉTODO QUE ADICIONA UM OBJETO FILME À LISTA

[HttpPost] // DESIGNA QUE O MÉTODO ABAIXO INSERE INFORMAÇÕES NA APLICAÇÃO

public void adicionaFilme([FromBody] Filme filme)

{ //[FromBody] DESGINA QUE O PARÂMETRO VIRÁ DO CORPO DA REQUISIÇÃO

filmes.Add(filme);

Console.WriteLine($"{filme.Titulo}\n{filme.Duracao}");//VERIFICAÇÃO DA INFORMAÇÃO

}

}

}

CRIAR Models->Filme.cs

namespace FilmesApi.Models;

public class Filme

{

public string Titulo { get; set; }

public string Genero { get; set; }

public int Duracao { get; set; }//DURAÇÃO EM MINUTOS

}

Em launchSettings.json - MUDAR

"launchBrowser": false, //BROWSER NÃO ABRE AUTOMATICAMENTE AO EXECUTAR A APLICAÇÃO

**INSERÇÃO DE DADOS NA APLICAÇÃO - POST**

Criação de um método que adiciona um objeto filme na memória, dado um estrutura json passada pelo Post. Isso é definido pelo método adicionaFilme no controlador

private static int id = 0;//VARIÁVEL QUE SERÁ ATRIBUÍDA AO CAMPO Id do objeto Filme

//MÉTODO QUE ADICIONA UM OBJETO FILME À LISTA

[HttpPost] // DESIGNA QUE O MÉTODO ABAIXO INSERE INFORMAÇÕES NA APLICAÇÃO

public void AdicionaFilme([FromBody] Filme filme)

{ //[FromBody] DESGINA QUE O PARÂMETRO VIRÁ DO CORPO DA REQUISIÇÃO

filme.Id = id++;// 0, 1, 2....

filmes.Add(filme);

}

Porém, ao seguir o padrão Rest, sempre que algo for adicionado, é necessário, além da inserção, retornar para o usuário o objeto adicionado e o caminho para ele. Isso pode ser feito pelo método CreatedAtAction().

private static int id = 0;//VARIÁVEL QUE SERÁ ATRIBUÍDA AO CAMPO Id do objeto Filme

//MÉTODO QUE ADICIONA UM OBJETO FILME À LISTA

[HttpPost] // DESIGNA QUE O MÉTODO ABAIXO INSERE INFORMAÇÕES NA APLICAÇÃO

public IActionResult AdicionaFilme([FromBody] Filme filme)

{ //[FromBody] DESGINA QUE O PARÂMETRO VIRÁ DO CORPO DA REQUISIÇÃO

filme.Id = id++;// 0, 1, 2....

filmes.Add(filme);

return CreatedAtAction(nameof(RecuperaFilmePorId),

new { id = filme.Id },

filme);

//CreatedAtAction - MÉTODO PADRÃO REST - RETORNA O OBJETO ADICIONADO E O SEU CAMINHO

//nameof(RecuperaFilmePorId) new { id = filme.Id } - CAMINHO DO OBJETO CRIADO

//filme - OBJETO CRIADO

}

NO POSTMAN

Execução da inserção



Retorno dela



Com caminho



**VALIDAR DADOS DO USUÁRIO COM Data Annotations**

Validar a entrada de dados do usuário pode ser feita na própria classe em Models

using System.ComponentModel.DataAnnotations;

namespace FilmesApi.Models;

public class Filme

{

[Required(ErrorMessage = "Título do filme obrigatório")]

[MaxLength(50, ErrorMessage = "Título do filme não pode exceder 50 caractéres")]

public string Titulo { get; set; }

[Required(ErrorMessage = "Gênero do filme obrigatório")]

[MaxLength(50, ErrorMessage = "Gênero do filme não pode exceder 50 caractéres")]

public string Genero { get; set; }

[Required(ErrorMessage = "Duração do filme obrigatória")]

[Range(50,600, ErrorMessage ="Duração do filme precisa ser de 50 minutos a 10H")]

public int Duracao { get; set; }//DURAÇÃO EM MINUTOS

}

Assim, toda vez que um objeto Filme for usado, seus atributos/propriedades seguirão estas restrições de

**[Required]** - atributo não pode ser vazio

**[MaxLength] ou [StringLength]** - tamanho máximo da string

**[Range]** - valor mínimo e máximo

**(ErrorMessage = “...”)** - personaliza a mensgem de erro mostrada na tela

**LEITURA DE DADOS NA APLICAÇÃO - GET**

**TODOS OS DADOS**

Utilizando a mesma classe (FilmeController), agora com um método que recebe HttpGet

[HttpGet] //DESIGNA QUE O MÉTODO ABAIXO OBTEM INFORMAÇÕES DA APLICAÇÃO

public IEnumerable<Filme> recuperaFilmes()

{

return filmes;//LISTA DE FILMES

}

E no Postman





**DADOS QUE CORRESPONDEM A UM CRITÉRIO**

Utilizando a mesma classe (FilmeController), agora com um método que recebe HttpGet e um parâmetro id

//MÉTODO QUE RETORNA O PRIMEIRO FILME ENCONTRADO, DADO SEU ID

[HttpGet("{id}")]//MÉTODO ABAIXO USA O VERBO GET, MAS COM ID, DIFERENTE DO ACIMA

public Filme? RecuperaFilmePorId(int id)

{ //Filme? - NULLABLE - RETORNO PODE ASSUMIR VALOR NULL

return filmes.FirstOrDefault(filme => filme.Id == id);

}

Mudada também a estrutura da classe filme

public int Id { get; set; }

E NO POSTMAN





**PAGINAÇÃO DE DADOS COM SKIP E TAKE**

Quando a quantidade de dados em memória for muito grande, pode ser útil dar ao usuário a opção de mostrar partes desses dados somente. Isso pode ser feito com os métodos Skip e Take aplicados na função que usar o verbo Get no Controlador.

//MÉTODO QUE LISTA VÁRIOS FILMES DA APLICAÇÃO - PULANDO skip FILMES INICIAIS

//E MOSTRANDO OS PRÓXIMOS take FILMES

[HttpGet] //DESIGNA QUE O MÉTODO ABAIXO OBTEM INFORMAÇÕES DA APLICAÇÃO

public IEnumerable<Filme> RecuperaFilmes([FromQuery] int skip = 0, //SEM DEFINIR, skip É 0

[FromQuery] int take = 50) //SEM DEFINIR, take É 50

{

return filmes.Skip(skip).Take(take);//LISTA DE FILMES

}

E NO POSTMAN



**CONEXÃO COM BASE DE DADOS**

Downloads necessários

Ferramentas -> Gerenciador de Pacotes NuGet- Gerenciar pacotes solução->Procurar-> Baixar

Microsoft.EntityFrameworkCore - versão. 6.0.10

MicrosoftEntityFrameworkCore.Tools versão. 6.0.10

Pomelo.EntityFrameworkCore.MySql - versão 6.0.2

CRIAR PASTA Data -> Vai intermediar a ligação entre a base de dados e as entidades da aplicação

Data-> class FilmeContext

public class FilmeContext : DbContext//CLASSE QUE REMEDIA LIGAÇÃO FILME (ENTIDADE) <-> FILME (BD)

{

//PROPRIEDADES

//PROPRIEDADE DE ACESSO AOS FILMES DA BD

public DbSet<Filme> Filmes{ get; set; }

//CONSTRUTOR QUE USA O CONSTRUTOR BASE (CONSTRUTOR DE DbContext)

public FilmeContext(DbContextOptions<FilmeContext> opts)

: base(opts)

{

}

}

Configurar arquivo appsettings.json, considerando

Servidor local, nome da base de dados filme, usuário root e senha root

"AllowedHosts": "\*",

"ConnectionStrings": {

"FilmeConnection": "server=localhost;database=filme;user=root;password=root"

}

Configurar program.cs

using FilmesApi.Data;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

//VARIÁVEL PARA DEFINIÇÕES ABAIXO

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

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

opts.UseMySql(connectionString,

ServerVersion.AutoDetect(connectionString)));

DUVIDAS

Atributos estáticos?