ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

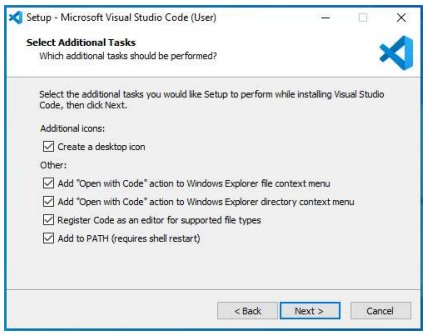
PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/2

Luiz Fernando Souza / Eliane Marion

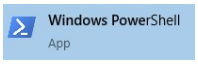
AULA 01 - INAUGURAL

- Apresentação da disciplina
- Competências, habilidades e bases tecnológicas da disciplina
- Formas de Avaliação
- Introdução e desenvolvimento do conteúdo

Download e Instalação do Visual Studio Code: Faça o Download do Visual Studio Code acessando https://code.visualstudio.com/download e instale deixando todas as opções selecionadas



Linhas de Comandos PowerShell: Para criação de projetos podemos utilizar a ferramenta de linha de comandos do Windows chamada de *PowerShell*



Verificando a versão do .NET Core instalado

```
C:\Users\luizs>dotnet --version
9.0.102
```

```
C:\Users\luizs>dotnet --list-sdks
8.0.405 [C:\Program Files\dotnet\sdk]
9.0.102 [C:\Program Files\dotnet\sdk]
```

Se seu computador não exibir nenhuma versão ou não reconhecer o comando, instale o .Net através do link a seguir: https://dotnet.microsoft.com/en-us/download/dotnet. A versão recomendada para as aulas é a 9.0

Version	Release type	Support phase	Latest release	Latest release date	End of support
.NET 9.0 (latest)	Standard Term Support	Active	9.0.1	January 14, 2025	May 12, 2026
.NET 8.0	Long Term Support	Active	8.0.12	January 14, 2025	November 10, 2026

Vá em computador, clique com o direito do mouse e em propriedades, para verificar se seu Windows é 32 ou 64 bits e faça o download compatível com seu computador

OS	Installers	Binaries
Linux	Package manager instructions	ARM32 | ARM64 | RHEL 6 x64 | x64 | x64 Alpine
macOS	x64	x64
Windows	x64 | x86	ARM32 | x64 | x86
All	dotnet-install scripts	

Windows 64 bits

Windows 32 bits

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/2

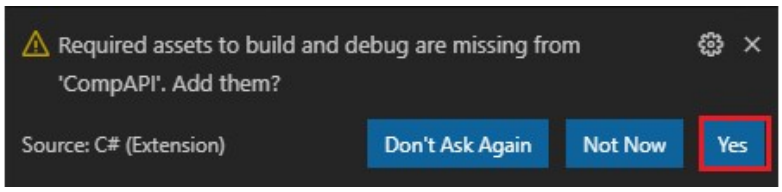
Luiz Fernando Souza / Eliane Marion

Criando Projetos WebApi

Crie uma pasta na sua organização de arquivos chamada **RpgApi** e abra ela no VS Code. Depois disso abra uma o terminal e digite o comando para criação de uma API, conforme abaixo

dotnet new webapi

Caso após a criação aparecerá uma a mensagem para ativar o modo de depuração para o C#. Escolha sim conforme abaixo



Identificação dos arquivos no Projeto

Classe Program: Será o ponto de partida ao rodar o projeto, como mencionado acima, nela está apontada a classe Startup.

Arquivo .csproject: Arquivo em que ficará registrado dos os pacotes baixados para utilização no projeto. Framework do banco de dados por exemplo.

Appsettings.json: Arquivo em que pode ser guardado informações de configurações, por exemplo o IP e dados de acesso de um banco de dados por exemplo.

Lauchsettings.json (pasta properties): Arquivo em que estarão informações sobre a execução do projeto, por exemplo qual o endereço que constará no navegador ao rodar a aplicação ou se utilizará o protocolo http ou https por exemplo. Neste arquivo remova o endereço *https* que aparece na propriedade applicationUrl para que o navegador não exiba mensagem de bloqueio ao executar o aplicativo

```
"CompDS": {
  "commandName": "Project",
  "launchBrowser": true,
  "launchUrl": "weatherforecast",
  "applicationUrl": "https://localhost:5001;http://localhost:5000",
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Development"
  }
}
```

- Isso é necessário para que ao rodar localmente o projeto, não ocorram problemas por não existir certificado de conexão segura.

Abra o terminal através do menu View → Terminal, digite a linha de comando `dotnet run` para rodar o projeto. Abra o navegador, digite o endereço e porta da API e o nome Controller que temos até então:

```
localhost:5000/WeatherForecast
```

O Navegador deverá exibir dados aleatórias em Cº e Fº que se trata da avaliação de temperaturas. Execute o comando CTRL + C no Visual Studio Code para interromper a aplicação assim que desejar.

Nas próximas etapas entenderemos melhor o que é uma Controller, mas como uma breve introdução, durante a criação do projeto foi criada uma Controller chamada WeatherForecast automaticamente na pasta correspondente

Testando API com Postman

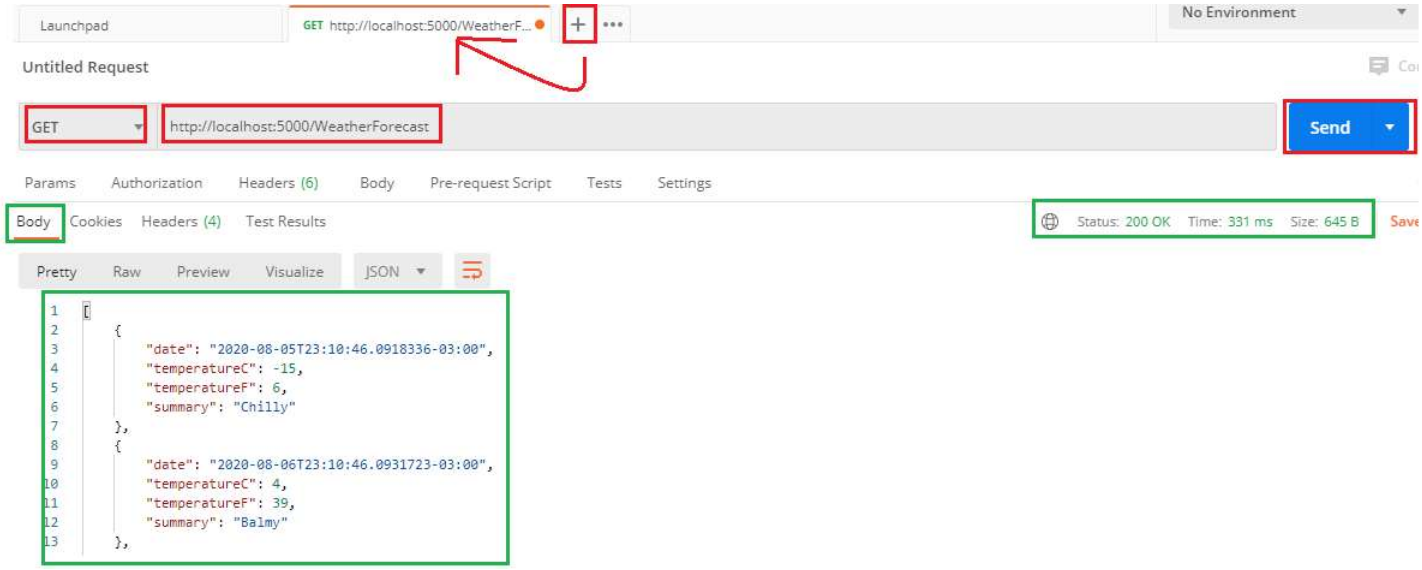
Como ainda não temos front-end (interface) ainda, precisaremos de uma ferramenta para executar testes no nosso back-end (Programação) e se mostra uma alternativa mais completa para testar todos os recursos que uma API oferece em comparação como o navegador. O Postman pode ser baixado através do endereço abaixo:

https://www.postman.com/downloads/

Você pode realizar o login através do gmail e manter o histórico de todos os seus testes dentro da ferramenta.

Em linhas gerais, o Postman é um API Client que podemos utilizar para realizar as requisições na API através dos principais métodos: Get, Post, Put e Delete

Execute a aplicação e realize as seguintes configurações no Postman para poder testar o método Get da API



- Em vermelho temos as configurações que devem ser feitas e em verde o resultado da requisição na API.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

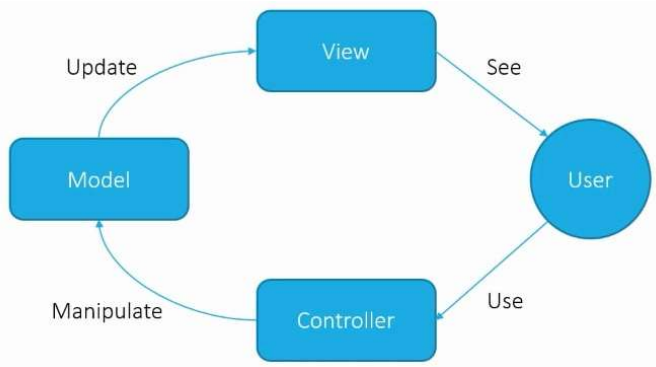
PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/2

Luiz Fernando Souza / Eliane Marion

Padrão MVC em Projetos WebAPI

- *Mapa Mental da Matéria:* https://mm.tt/1587285354?t=PMFTNcCUTT

Revisão do Padrão MVC



Divide a lógica de um programa em três elementos interconectados. Podemos exemplificar o padrão MVC conforme acima. O Usuário requisita dados a uma Controlador que carrega o Modelo e expõe os dados numa Visualização.

1. No projeto RpgApi, crie uma pasta chamada **Models** e dentro desta pasta crie uma classe chamada Personagem conforme abaixo. Utilize o atalho (digitando prop + TAB + TAB) para criar as propriedades.

```
namespace RpgApi.Models
{
    0 references
    public class Personagem
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Nome { get; set; }
        0 references
        public int PontosVida { get; set; }
        0 references
        public int Forca { get; set; }
        0 references
        public int Defesa { get; set; }
        0 references
        public int Inteligencia { get; set; }
    }
}
```

2. Dentro da pasta Models, crie uma outra pasta chamada **Enuns**. Clique com o botão direito na pasta *Enuns* recém-criada e adicione uma classe chamada **ClasseEnum**. Como queremos criar uma enumeração, alteraremos a palavra *class* por *enum* na assinatura da classe e acrescentaremos os itens deste enum.

```
namespace RpgApi.Models.Enuns
{
    2 references
    public enum ClasseEnum
    {
        1 reference
        Cavaleiro = 1,
        0 references
        Mago = 2,
        0 references
        Clerigo = 3
    }
}
```

3. Adicione na classe **Personagem** uma propriedade ligada a enumeração recém-criada, conforme abaixo. Será necessário fazer referência ao namespace da enumeração.

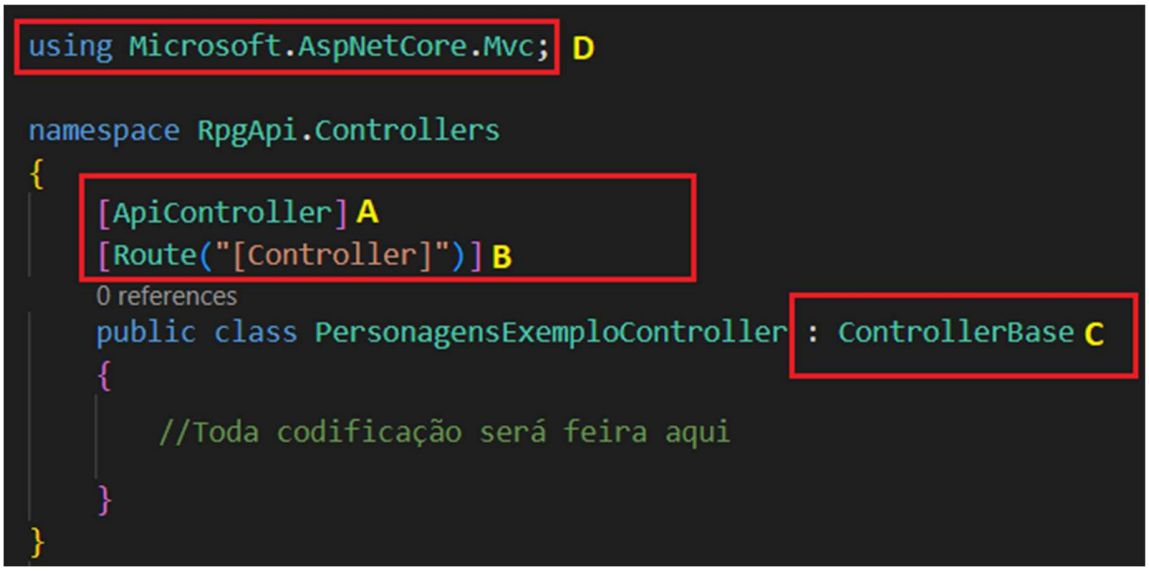
```
using RpgApi.Models.Enuns;

namespace RpgApi.Models
{
    0 references
    public class Personagem
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Nome { get; set; }
        0 references
        public int PontosVida { get; set; }
        0 references
        public int Forca { get; set; }
        0 references
        public int Defesa { get; set; }
        0 references
        public int Inteligencia { get; set; }
        0 references
        public ClasseEnum Classe{ get; set; }
    }
}
```

Luiz Fernando Souza / Eliane Marion

---

4. Crie uma classe chamada **PersonagensExemploController** na pasta *Controllers* e codifique conforme abaixo. A seguir informaremos a função de cada trecho de código.



   a) Atributo *ApiController* usado porque é uma API de requisição *http*
   b) Atributo *Route* usado para definir rota a ser digitada no navegador para chamar a *controller* e o Método
   c) Toda classe *Controller* herdará da *ControllerBase*.
   d) Os atributos *apicontroller*, *Route* e *ControllerBase* necessita do *using* de *Microsoft.AspNetCore.Mvc* que é o Framework utilizado.

5. Programe a criação de uma lista do tipo Personagem dentro da classe controller de maneira global. O usings necessários serão System.Collections.Generic, RpgApi.Models e RpgApi.Enuns.

```
[ApiController]
[Route("[Controller]")]
0 references
public class PersonagensExemploController : ControllerBase
{
    0 references
    private static List<Personagem> personagens = new List<Personagem>()
    {
        //Modo de criação e inclusão de objetos de uma só vez na lista
        new Personagem() { Id = 1, Nome = "Frodo", PontosVida=100, Forca=17, Defesa=23, Inteligencia=33, Classe=ClasseEnum.Cavaleiro},
        new Personagem() { Id = 2, Nome = "Sam", PontosVida=100, Forca=15, Defesa=25, Inteligencia=30, Classe=ClasseEnum.Cavaleiro},
        new Personagem() { Id = 3, Nome = "Galadriel", PontosVida=100, Forca=18, Defesa=21, Inteligencia=35, Classe=ClasseEnum.Clerigo },
        new Personagem() { Id = 4, Nome = "Gandalf", PontosVida=100, Forca=18, Defesa=18, Inteligencia=37, Classe=ClasseEnum.Mago },
        new Personagem() { Id = 5, Nome = "Hobbit", PontosVida=100, Forca=20, Defesa=17, Inteligencia=31, Classe=ClasseEnum.Cavaleiro },
        new Personagem() { Id = 6, Nome = "Celeborn", PontosVida=100, Forca=21, Defesa=13, Inteligencia=34, Classe=ClasseEnum.Clerigo },
        new Personagem() { Id = 7, Nome = "Radagast", PontosVida=100, Forca=25, Defesa=11, Inteligencia=35, Classe=ClasseEnum.Mago }
    };


}
```

6. Crie um método de nome GetFirst listar o primeiro personagem. Método Get que retornará Ok (Status http 200) e os dados contidos no objeto p.

```
public IActionResult GetFirst()
{
    Personagem p = personagens[0];
    return Ok(p);
}
```

7. Configure a classe program.cs para o uso das controllers através dos trechos sinalizados abaixo

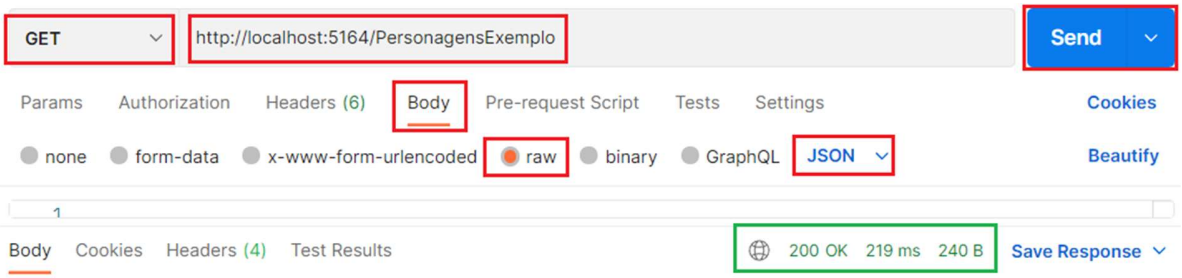
```
// Learn more about configuring OpenAPI
builder.Services.AddOpenApi();
builder.Services.AddControllers();

var app = builder.Build();
```

```
.WithName("GetWeatherForecast");

app.MapControllers();
app.Run();
```

8. Execute a aplicação (*dotnet run*) e faça a chamada no navegador ou postman para o método Get do endereço representado a seguir. **5164** é a porta que está rodando neste exemplo, observe para o seu comando qual é a porta executada.



Como víamos na aula sobre lista, uma lista é uma coleção de objetos que pode ser desde uma lista de números inteiros, uma lista de strings ou uma lista de objetos da classe Personagem, por exemplo. É possível realizar diversas operações com lista, como busca, soma, adição de itens, remoção. Aprenderemos a usar aos poucos estas funcionalidades.

9. Crie um método *Get* para que ele possa exibir a lista.

```
public IActionResult Get()
{
    return Ok(personagens);
}
```

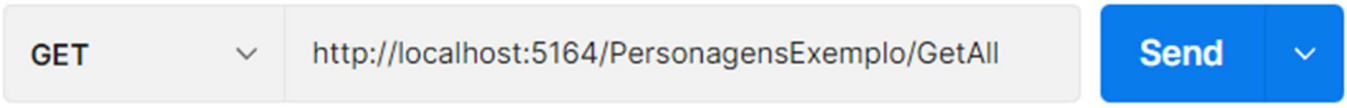
- Execute e tente realizar o *get* no *postman*. Você perceberá que retornará um erro pois existem dois métodos do tipo *HttpGet* e precisaremos diferenciar a rota deles.

10. Para resolver o problema descrito na etapa anterior, determinaremos nomenclaturas que os distinguem sendo Get. Isso terá o nome de **rota**.

```
[HttpGet("Get")]
0 references
public IActionResult GetFirst()
{
    return Ok(personagens[0]);
}

[HttpGet("GetAll")]
0 references
public IActionResult Get()
{
    return Ok(personagens);
}
```

- Execute novamente chamando um dos métodos como antes e o outro com a rota abaixo

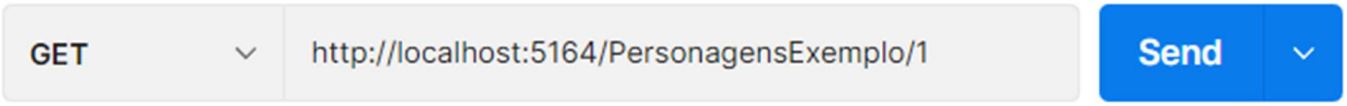


11. Crie um método *GetSingle* para que aceite um parâmetro pela rota. Esse parâmetro será usado como critério para fazer uma busca na lista.

```
[HttpGet("{id}")]
0 references
public IActionResult GetSingle(int id)
{
    return Ok(personagens.FirstOrDefault(pe => pe.Id == id));
}
```

- Métodos de operações em lista, como o *FirstOrDefault*, exigem o *using System.Linq*

Faça o teste no *postman*:



Métodos do tipo Post

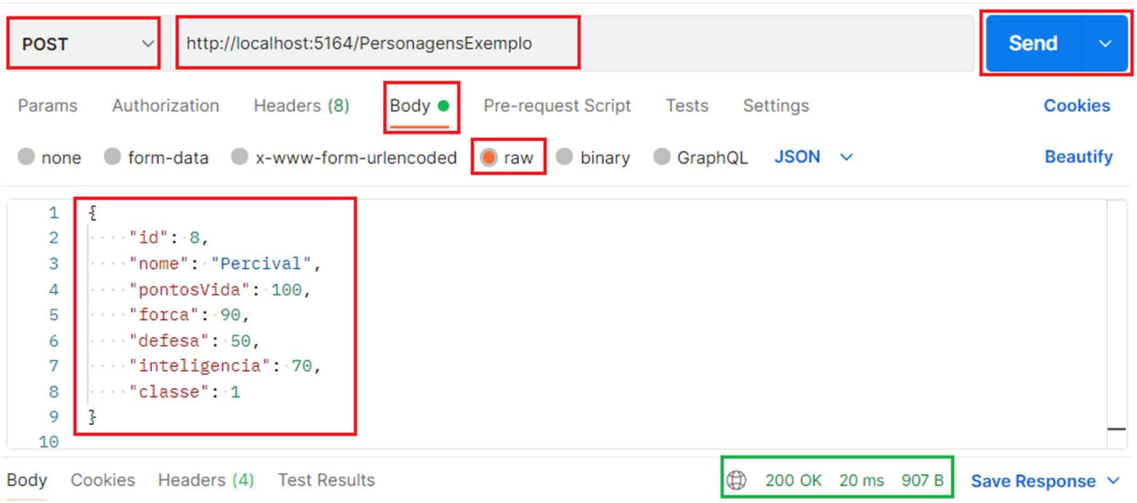
Os métodos *Post* são responsáveis por enviar dados para um servidor via corpo da requisição, logo é possível enviar objetos com suas propriedades totalmente preenchidas para que uma operação seja realizada, por exemplo, o salvamento numa base de dados ou adição em uma lista.

12. Crie um método do tipo *Post* conforme abaixo

```
[HttpPost]
0 references
public IActionResult AddPersonagem(Personagem novoPersonagem)
{
    personagens.Add(novoPersonagem);
    return Ok(personagens);
}
```

- Perceba que o objeto está preenchendo sendo adicionado a lista e ela está sendo retornada para o servidor.

13. Configure o *postman* para o teste do método *Post* e depois clique em *Send*:



- O resultado deverá ser a exibição da lista de personagens, contendo o recém adicionado por você.

- No exemplo acima poderíamos ter adicionado as demais propriedades do objeto personagem. Você pode fazer isso para fins de teste, não esquecendo de separar cada propriedade com vírgula.

Dica: Toda API no .net pode ter sua documentação visualizada no Swagger. Uma página que o projeto cria automaticamente. Acesse por http://localhost:XYZ/swagger. Substitua o XYZ pelo número da porta em que sua API está rodando no computador.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/2

Luiz Fernando Souza / Eliane Marion

Exemplos de Métodos usando listas – Aplicar na Controller PersonagensExemploController

- Ordenando uma lista por critério

```
[HttpGet("GetOrdenado")]
0 references
public IActionResult GetOrdem()
{
    List<Personagem> listaFinal = personagens.OrderBy(p => p.Forca).ToList();
    return Ok(listaFinal);
}
```

- Contar Itens de uma lista

```
[HttpGet("GetContagem")]
0 references
public IActionResult GetQuantidade()
{
    return Ok("Quantidade de personagens: " + personagens.Count);
}
```

- Somando valores da propriedade comum entre objetos de uma lista

```
[HttpGet("GetSomaForca")]
0 references
public IActionResult GetSomaForca()
{
    return Ok(personagens.Sum(p => p.Forca));
}
```

- Filtrando dados de uma lista de acordo com critérios

```
[HttpGet("GetSemCavaleiro")]
0 references
public IActionResult GetSemCavaleiro()
{
    List<Personagem> listaBusca = personagens.FindAll(p => p.Classe != ClasseEnum.Cavaleiro);
    return Ok(listaBusca);
}
```

- Busca por nome aproximado

```
[HttpGet("GetByNomeAproximado/{nome}")]
0 references
public IActionResult GetByNomeAproximado(string nome)
{
    List<Personagem> listaBusca = personagens.FindAll(p => p.Nome.Contains(nome));
    return Ok(listaBusca);
}
```

- Filtrando um personagem por algum critério e removendo o mesmo da lista

```
[HttpGet("GetRemovendoMago")]
0 references
public IActionResult GetRemovendoMagos()
{
    Personagem pRemove = personagens.Find(p => p.Classe == ClasseEnum.Mago);
    personagens.Remove(pRemove);
    return Ok("Personagem removido: " + pRemove.Nome);
}
```

- Filtro pela força

```
[HttpGet("GetByForca/{forca}")]
0 references
public IActionResult Get(int forca)
{
    List<Personagem> listaFinal = personagens.FindAll(p => p.Forca == forca);
    return Ok(listaFinal);
}
```

Exemplo de método Post com validação das propriedades

```
[HttpPost]
0 references
public IActionResult AddPersonagem(Personagem novoPersonagem)
{
    if (novoPersonagem.Inteligencia == 0)
        return BadRequest("Inteligência não pode ter o valor igual a 0 (zero).");

    personagens.Add(novoPersonagem);
    return Ok(personagens);
}
```

Informação adicional: Outra forma de executar o projeto, além do comando *dotnet run* é utilizar o comando play.

 (CTRL + Shift + D) → 

Referências para o estudo de listas

https://www.tutorialsteacher.com/csharp/csharp-list

https://www.dotnetperls.com/list