Métodos do tipo Post

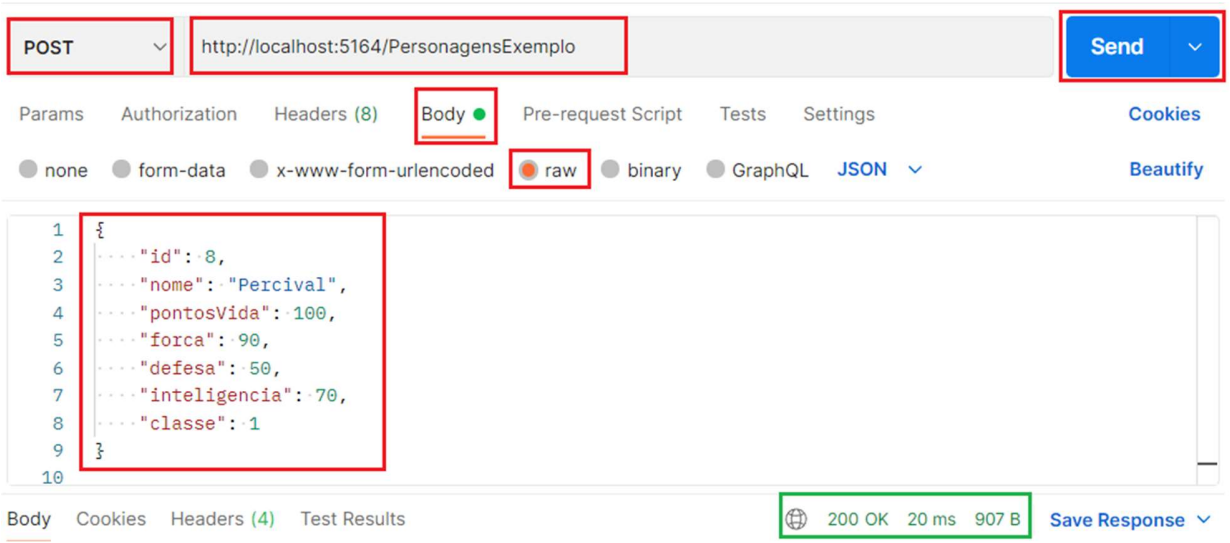
Os métodos *Post* são responsáveis por enviar dados para um servidor via corpo da requisição, logo é possível enviar objetos com suas propriedades totalmente preenchidas para que uma operação seja realizada, por exemplo, o salvamento numa base de dados ou adição em uma lista.

11. 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.

12. 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 – 2023/1

Luiz Fernando Souza / Elaine Marion

Métodos Put e Delete

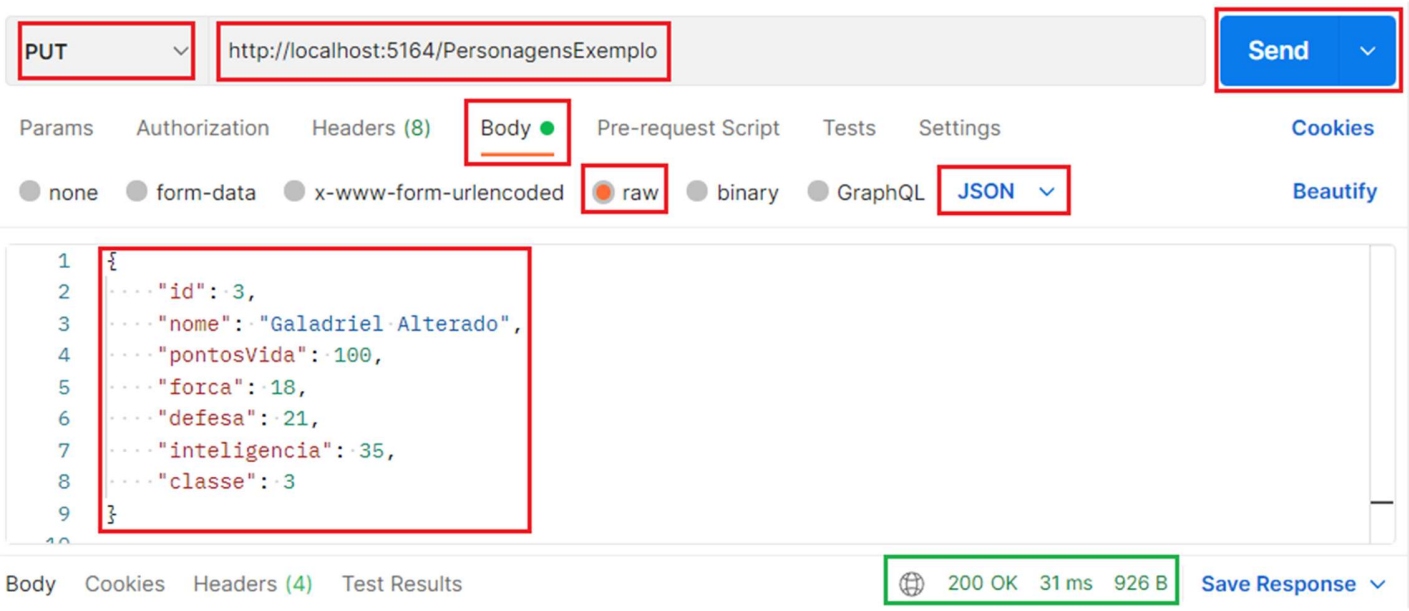
Método HttpPut

O Método *HttpPut* é utilizado quando queremos fazer uma alteração ou modificar um objeto existente. Para realizar esta alteração teremos que ter um identificador dentro do objeto preenchido para que possamos atualizá-lo. Geralmente utilizamos o Id para isso. Como estamos trabalhando com lista, a nossa lógica será encontrar o objeto através do Id, alterar as propriedades do objeto e depois exibir a lista novamente, conferindo se a atualização foi realizada.

1. Faça a criação do método *HttpPut* conforme a seguir

```
[HttpPut]
0 references
public IActionResult UpdatePersonagem(Personagem p)
{
    Personagem pernsagemAlterado = personagens.Find(pers => pers.Id == p.Id);
    pernsagemAlterado.Nome = p.Nome;
    pernsagemAlterado.PontosVida= p.PontosVida;
    pernsagemAlterado.Forca = p.Forca;
    pernsagemAlterado.Defesa = p.Defesa;
    pernsagemAlterado.Inteligencia = p.Inteligencia;
    pernsagemAlterado.Classe = p.Classe;

    return Ok(personagens);
}
```

2. Realize o teste no *postman* configurando a ferramenta conforme a seguir



Luiz Fernando Souza / Elaine Marion

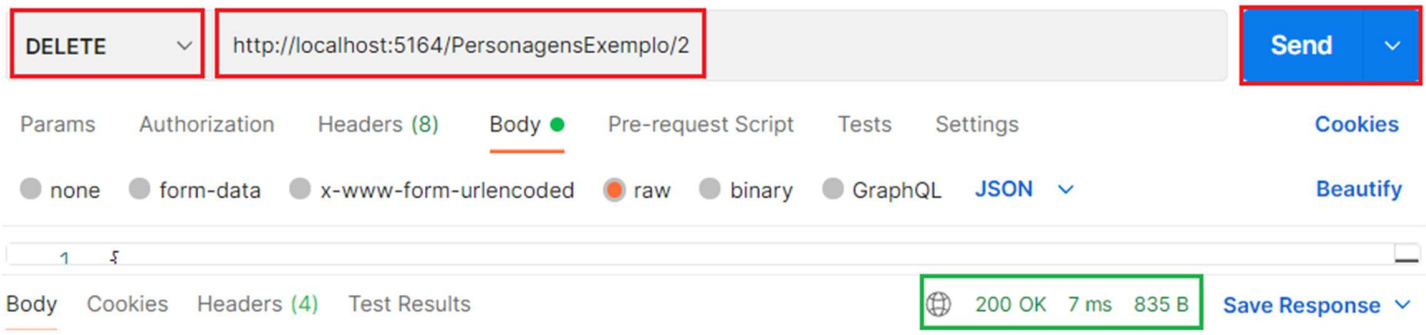
Método HttpDelete

O método *HttpDelete* é utilizado para fazer a remoção de um objeto. A configuração dele necessita que informemos apenas o id no parâmetro da rota.

3. Configure o método *HttpDelete* conforme abaixo

```
[HttpDelete("{id}")]
0 references
public IActionResult Delete(int id)
{
    personagens.RemoveAll(pers => pers.Id == id);

    return Ok(personagens);
}
```

4. Realize o teste com o *postman* passando um id na rota. Verifique se o objeto com o referido id foi removido da lista.



ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE II – 2023/1

Luiz Fernando Souza / Elaine Marion

Exemplos de Métodos usando listas – Aplicar na Controller PersonagemExemploController

- 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

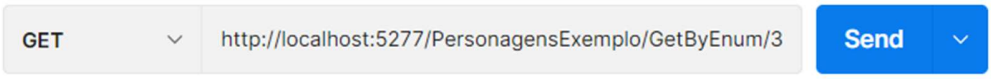
```
[HttpGet("GetByForca/{forca}")]
0 references
public IActionResult Get(int forca)
{
    List<Personagem> listaFinal = personagens.FindAll(p => p.Forca == forca);
    return Ok(listaFinal);
}
```

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);
}
```

- Exemplo de Filtragem em lista através de enum

```
[HttpGet("GetByEnum/{enumId}")]
0 references
public IActionResult GetByEnum(int enumId)
{
    //Conversao explicita de int para enum
    ClasseEnum enumDigitado = (ClasseEnum)enumId;

    List<Personagem> listaBusca = personagens.FindAll(p => p.Classe == enumDigitado);

    return Ok(listaBusca);
}
```



ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE II – 2023/1

Luiz Fernando Souza / Elaine Marion

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