ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

**DESENVOLVIMENTO DE SISTEMAS – 2024/1**

*Luiz Fernando Souza / Quitéria Danno*

---

**Aula 12 – Mapeamento de tabelas e colunas - Preparando API para a disputa entre os personagens – Classe Random para Sorteios**

1. Inclua o arquivo Script06_TabelaDisputas.sql em seu projeto e execute as instruções no banco de dados, referente a tabela disputas.

2. Crie a classe **Disputa.cs** na pasta Models, com as propriedades sinalizadas abaixo. NotMapped exigirá o using *System.ComponentModel.DataAnnotations.Schema*

```
public class Disputa
{
    0 references
    public int Id { get; set; }
    0 references
    public DateTime? DataDisputa { get; set; }
    0 references
    public int AtacanteId { get; set; }
    0 references
    public int OponenteId { get; set; }
    0 references
    public string Narracao { get; set; } = string.Empty;

    [NotMapped]
    0 references
    public int? HabilidadeId { get; set; }

    [NotMapped]
    0 references
    public List<int>? ListaIdPersonagens { get; set; }

    [NotMapped]
    0 references
    public List<string>? Resultados { get; set; }
}
```

- Perceba que temos uma lista de inteiros que armazenará o Id dos personagens envolvidos na disputa, que poderá ser mais que dois e uma lista de strings para armazenar os resultados. Compile o código e confirme a inexistência de erros.

3. Abra a classe **DataContext** e faça o mapeamento da classe para o banco

```
public DbSet<PersonagemHabilidade> TB_PERSONAGENS_HABILIDADES { get; set; }
0 references
public DbSet<Disputa> TB_DISPUTAS { get; set; }

0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Personagem>().ToTable("TB_PERSONAGENS");
    modelBuilder.Entity<Arma>().ToTable("TB_ARMAS");
    modelBuilder.Entity<Usuario>().ToTable("TB_USUARIOS");
    modelBuilder.Entity<Habilidade>().ToTable("TB_HABILIDADES");
    modelBuilder.Entity<PersonagemHabilidade>().ToTable("TB_PERSONAGENS_HABILIDADES");
    modelBuilder.Entity<Disputa>().ToTable("TB_DISPUTAS");
```

4. Ainda na classe DataContext prograremos antes do final método OnModelCreating o mapeamento de todas as configurações da tabela de disputas já que ela foi criada sem migração. Esse é um cenário em que um banco já existe é importante para sabermos como lidar para adequar ele ao EntityFramework.

```
    modelBuilder.Entity<Usuario>().HasData(user);
    //Fim da criação do usuário padrão.

    //Define que se o Perfil não for informado, o valor padrão será jogador
    modelBuilder.Entity<Usuario>().Property(u => u.Perfil).HasDefaultValue("Jogador");

    modelBuilder.Entity<Disputa>().HasKey(d => d.Id);//Indicação da chave primária da entidade.
    //Abaixo fica o mapeamento do nome das colunas da tabela para as propriedades da classe.
    modelBuilder.Entity<Disputa>().Property(d => d.DataDisputa).HasColumnName("Dt_Disputa");
    modelBuilder.Entity<Disputa>().Property(d => d.AtacanteId).HasColumnName("AtacanteId");
    modelBuilder.Entity<Disputa>().Property(d => d.OponenteId).HasColumnName("OponenteId");
    modelBuilder.Entity<Disputa>().Property(d => d.Narracao).HasColumnName("Tx_Narracao");
}
```

5. Crie uma controller chamada DisputasController.cs.

```
using Microsoft.AspNetCore.Mvc;
namespace RpgApi.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class DisputasController : ControllerBase
    {
        //Construtores e métodos aqui.
    }
}
```

6. Declare a variável do contexto do banco de dados e inicialize ela no construtor. O contexto necessitará do using RpgApi.Data

```
using Microsoft.AspNetCore.Mvc;
using RpgApi.Data;

namespace RpgApi.Controllers
{
    [ApiController]
    [Route("[controller]")]
    0 references
    public class DisputasController : ControllerBase
    {
        1 reference
        private readonly DataContext _context;
        0 references
        public DisputasController(DataContext context)
        {
            _context = context;
        }
    }
}
```

7. Dentro da controller, programe o método que fará um ataque a outro personagem usando uma Arma. Necessário using RpgApi.Models. Observe que esse método tem uma rota chamada *Arma*.

```
[HttpPost("Arma")]
0 references
public async Task<IActionResult> AtaqueComArmaAsync(Disputa d)
{
    try
    {
        //Programação dos próximos passos aqui
        return Ok(d);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

8. Programe dentro do bloco try do método a busca no banco de dados do personagem que está atacando e do seu oponente, através dos dados que foram passados por parâmetro no objeto “d” do tipo Disputa. Necessário o using *Microsoft.EntityFrameworkCore*

```
Personagem? atacante = await _context.TB_PERSONAGENS
    .Include(p => p.Arma)
    .FirstOrDefaultAsync(p => p.Id == d.AtacanteId);

Personagem? oponente = await _context.TB_PERSONAGENS
    .FirstOrDefaultAsync(p => p.Id == d.OponenteId);
```

9. Tendo identificado os personagens envolvidos na disputa, criaremos a lógica para definir o valor do ataque, o dano da arma e quanto isso custará em pontos de vida para o oponente. Usaremos uma classe randômica para gerar valores aleatórios envolvendo alguma das propriedades

```
A int dano = atacante.Arma.Dano + (new Random().Next(atacante.Forca));

B dano = dano - new Random().Next(oponente.Defesa);

  if (dano > 0)
   C  oponente.PontosVida = oponente.PontosVida - (int)dano;
  if (oponente.PontosVida <= 0)
   D  d.Narracao = $"{oponente.Nome} foi derrotado!";
```

   (A) Somamos o dano da arma do atacante a um valor aleatório da força do atacante no intervalo entre 0 e a força que ele tem no cadastro.
   (B) Subtraímos pelo valor da defesa do oponente em um valor aleatório entre 0 e o valor da defesa do oponente.
   (C) Se o valor for maior que 0, subtraímos o valor do dano nos pontos de vida do oponente
   (D) Se o valor for menor ou igual a 0, guardamos a mensagem que o oponente foi derrotado.

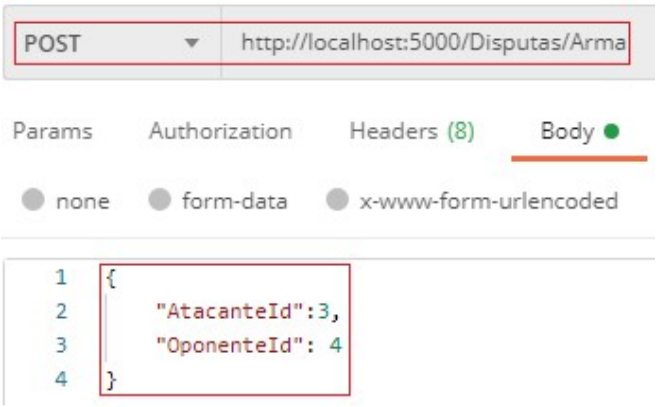
10. Na sequência atualizaremos os dados do oponente, já que ele pode ter pontos de vida subtraídos.

```
_context.TB_PERSONAGENS.Update(oponente);
await _context.SaveChangesAsync();
```

11. Insira um resumo do que aconteceu e salve na tabela de disputas antes do retorno do método. Necessário o using System.Text

```
StringBuilder dados = new StringBuilder();
dados.AppendFormat(" Atacante: {0}. ", atacante.Nome);
dados.AppendFormat(" Oponente: {0}. ", oponente.Nome);
dados.AppendFormat(" Pontos de vida do atacante: {0}. ", atacante.PontosVida);
dados.AppendFormat(" Pontos de vida do oponente: {0}. ", oponente.PontosVida);
dados.AppendFormat(" Arma Utilizada: {0}. ", atacante.Arma.Nome);
dados.AppendFormat(" Dano: {0}. ", dano);

d.Narracao += dados.ToString();
d.DataDisputa = DateTime.Now;
_context.TB_DISPUTAS.Add(d);
_context.SaveChanges();

return Ok(d);
```

12. Teste os dados no postman de acordo com os Ids de personagem que você tem na base de dados. O Persnagem deve ter uma arma vinculada a ele. Caso seja necessário atualizar os dados dos personagens, abra a tabela no banco de dados e redefina os valores. Futuramente definiremos métodos para realizar estas operações.




**Ataque através de Habilidades**

13. Crie o método para realizar o ataque através das habilidades com a estrutura a seguir

```
[HttpPost("Habilidade")]
0 references
public async Task<IActionResult> AtaqueComHabilidadeAsync(Disputa d)
{
    try
    {
        //Programção dos próximos passos aqui
        return Ok(d);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

14. Programe dentro do bloco try antes do return Ok.

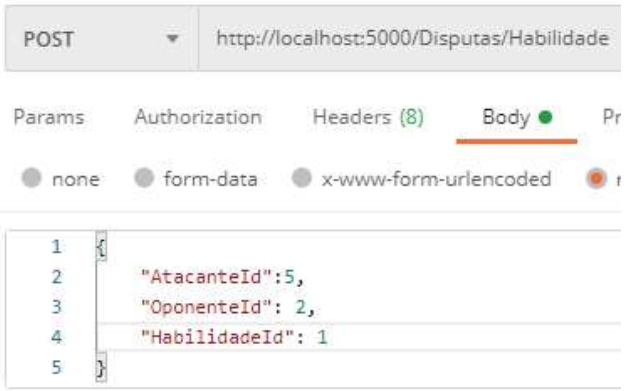
```
Personagem atacante = await _context.TB_PERSONAGENS
    .Include(p => p.PersonagemHabilidades).ThenInclude(ph => ph.Habilidade)
    .FirstOrDefaultAsync(p => p.Id == d.AtacanteId);

Personagem oponente = await _context.TB_PERSONAGENS
    .FirstOrDefaultAsync(p => p.Id == d.OponenteId);

PersonagemHabilidade ph = await _context.TB_PERSONAGENS_HABILIDADES                    A
    .Include(p => p.Habilidade).FirstOrDefaultAsync(phBusca => phBusca.HabilidadeId == d.HabilidadeId
    && phBusca.PersonagemId == d.AtacanteId);
```

(A) Busca da habilidade através do id da habilidade contida no objeto recebido por parâmetro.

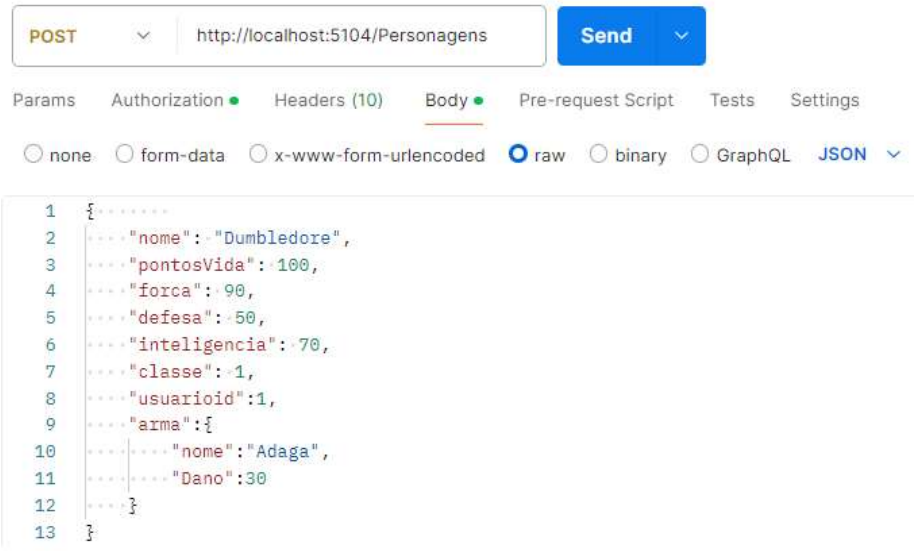
```
    if (ph == null)
     B  d.Narracao = $"{atacante.Nome} não possui esta habilidade";
    else
    {
     C  int dano = ph.Habilidade.Dano + (new Random().Next(atacante.Inteligencia));
        dano = dano - new Random().Next(oponente.Defesa);

        if (dano > 0)
            oponente.PontosVida = oponente.PontosVida - dano;
        if (oponente.PontosVida <= 0)
            d.Narracao += $"{oponente.Nome} foi derrotado!";

        _context.TB_PERSONAGENS.Update(oponente);
        await _context.SaveChangesAsync();

        StringBuilder dados = new StringBuilder();
        dados.AppendFormat(" Atacante: {0}. ", atacante.Nome);
        dados.AppendFormat(" Oponente: {0}. ", oponente.Nome);
        dados.AppendFormat(" Pontos de vida do atacante: {0}. ", atacante.PontosVida);
        dados.AppendFormat(" Pontos de vida do oponente: {0}. ", oponente.PontosVida);
        dados.AppendFormat(" Habilidade Utilizada: {0}. ", ph.Habilidade.Nome);
        dados.AppendFormat(" Dano: {0}. ", dano);

     D  d.Narracao += dados.ToString();
        d.DataDisputa = DateTime.Now;
        _context.TB_DISPUTAS.Add(d);
        _context.SaveChanges();
    }
    return Ok(d);
}
```

(B) Se a habilidade não for encontrada para o atacante, guardamos na narração a mensagem.
(C) Caso contrário realizamos o ataque e atualizamos os dados.
(D) Salvamento dos dados da disputa na tabela correspondente.

15. O teste no postman deve ser realizado conforme a seguir. O personagem que está atacando deve possuir a habilidade informada no Postman




16. Autoinclude: Abra o postman e faça a configuração de autoinserção de objetos relacionados conforme a imagem abaixo, realize o envio e confira as tabelas de Personagens e Armas no banco de dados.




- O recurso de auto inclusão é muito interessante e só é possível pois existe os relacionamentos criados, facilitando a vinculação de duas entidades na inserção das informações no banco de dados.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

**DESENVOLVIMENTO DE SISTEMAS – 2024/1**

*Luiz Fernando Souza / Quitéria Danno*

---

**Adicional**

Exemplo de Sorteio com Random:
https://youtu.be/p4DS4SROf0E

Exemplo de métodos da classe Random:
https://www.tutorialsteacher.com/articles/generate-random-numbers-in-csharp

```
[HttpGet("PersonagemRandom")]
0 references
public async Task<IActionResult> Sorteio()
{
    List<Personagem> personagens =
        await _context.Personagens.ToListAsync();

    //Sorteio com numero da quantidade de personagens
    int sorteio = new Random().Next(personagens.Count);

    //busca na lista pelo indice sorteado (Não é o ID)
    Personagem p = personagens[sorteio];

    string msg =
        string.Format("Nº Sorteado {0}. Personagem: {1}", sorteio, p.Nome);

    return Ok(msg);
}
```