ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2025/2

Luiz Fernando Souza / Quitéria Danno

Atividade Aula 13: Disputa coletiva e atualização os dados de cada disputa - Métodos Complementares

Programar as etapas, testar localmente. Siga as instruções de entrega da atividade que serão dadas na aula.

1. Criaremos um método que fará a disputa entre uma lista de Personagens informada pelos ids deles. A sequência do método ficará conforme a programação abaixo. Faça o using **System.Linq**

```
[HttpPost("DisputaEmGrupo")]
0 references
public async Task<IActionResult> DisputaEmGrupoAsync(Disputa d)
{
    try
    {
        d.Resultados = new List<string>();//Instancia a lista de resultados

        //Busca na base dos personagens informados no parametro incluindo Armas e Habilidades
        List<Personagem> personagens = await _context.Personagens
            .Include(p => p.Arma)
            .Include(p => p.PersonagemHabilidades).ThenInclude(ph => ph.Habilidade)
            .Where(p => d.ListaIdPersonagens.Contains(p.Id)).ToListAsync();

        //Contagem de personagens vivos na lista obtida do banco de dados
        int qtdPersonagensVivos = personagens.FindAll(p => p.PontosVida > 0).Count;

        //Enquanto houver mais de um personagem vivo haverá disputa
        while (qtdPersonagensVivos > 1)
        {
            //ATENÇÃO: Todas as etapas a seguir devem ficar aqui dentro do While
        }
        //Código após o fechamento do While. Atualizará os pontos de vida,
        //disputas, vitórias e derrotas de todos personagens ao final das batalhas
        _context.Personagens.UpdateRange(personagens);
        await _context.SaveChangesAsync();

        return Ok(d); //retorna os dados de disputas
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

- ATENÇÃO: Toda codificação das etapas a seguir devem ser realizadas dentro do While

2. Realize a programação a seguir dentro do while e faça leitura dos comentários para ficar por dentro dos procedimentos.

```
//Enquanto houver mais de um personagem vivo haverá disputa
while (qtdPersonagensVivos > 1)
{
    //Seleciona personagens com pontos vida positivos e depois faz sorteio.
    List<Personagem> atacantes = personagens.Where(p => p.PontosVida > 0).ToList();
    Personagem atacante = atacantes[new Random().Next(atacantes.Count)];
    d.AtacanteId = atacante.Id;

    //Seleciona personagens com pontos vida positivos, exceto o atacante escolhido e depois faz sorteio.
    List<Personagem> oponentes = personagens.Where(p => p.Id != atacante.Id && p.PontosVida > 0).ToList();
    Personagem oponente = oponentes[new Random().Next(oponentes.Count)];
    d.OponenteId = oponente.Id;

    //declara e redefine a cada passagem do while o valor das variáveis que serão usadas.
    int dano = 0;
    string ataqueUsado = string.Empty;
    string resultado = string.Empty;

    //Sorteia entre 0 e 1: 0 é um ataque com arma e 1 é um ataque com habilidades
    bool ataqueUsaArma = (new Random().Next(1) == 0);

    if (ataqueUsaArma && atacante.Arma != null)
    {
        //Programação do ataque com arma
    }
    else if (atacante.PersonagemHabilidades.Count != 0)//Verifica se o personagem tem habilidades
    {
        //Programação do ataque com habilidade
    }
}
```

3. Procure o “**if**” do ataque com arma, pois programaremos a mesma lógica do método de ataque com arma, guardando o dano e o nome da arma nas variáveis criadas anteriormente

```
if (ataqueUsaArma && atacante.Arma != null)
{
    //Programação do ataque com arma caso o atacante possua arma (o != null) do if

    //Sorteio da Força
    dano = atacante.Arma.Dano + (new Random().Next(atacante.Forca));
    dano = dano - new Random().Next(oponente.Defesa); //Sorteio da defesa.
    ataqueUsado = atacante.Arma.Nome;

    if (dano > 0)
        oponente.PontosVida = oponente.PontosVida - (int)dano;

    //Formata a mensagem
    resultado =
        string.Format("{0} atacou {1} usando {2} com o dano {3}.", atacante.Nome, oponente.Nome, ataqueUsado, dano);
    d.Narracao += resultado; // Concatena o resultado com as narrações existentes.
    d.Resultados.Add(resultado);//Adiciona o resulta atual na lista de resultados.
}
```

4. Para o else, a configuração do ataque com habilidade ficará como a seguir

```
    else if (atacante.PersonagemHabilidades.Count != 0)//Verifica se o personagem tem habilidades na lista dele
    {
        //Programação do ataque com habilidade

        //Realiza o sorteio entre as habilidade existentes e na linha seguinte a seleciona.
        int sorteioHabilidadeId = new Random().Next(atacante.PersonagemHabilidades.Count);
        Habilidade habilidadeEscolhida = atacante.PersonagemHabilidades[sorteioHabilidadeId].Habilidade;
        ataqueUsado = habilidadeEscolhida.Nome;

        //Sorteio da inteligência somada ao dano
        dano = habilidadeEscolhida.Dano + (new Random().Next(atacante.Inteligencia));
        dano = dano - new Random().Next(oponente.Defesa);//Sorteio da defesa.

        if (dano > 0)
            oponente.PontosVida = oponente.PontosVida - (int)dano;

        resultado =
            string.Format("{0} atacou {1} usando {2} com o dano {3}.", atacante.Nome, oponente.Nome, ataqueUsado, dano);
        d.Narracao += resultado;
        d.Resultados.Add(resultado);
    }

    //Atenção: Aqui ficará a Programação da verificação do ataque usado e verificação se existe mais de um personagem vivo
}
//Código após o fechamento do While. Atualizará os pontos de vida,
//disputas, vitórias e derrotas de todos personagens ao final das batalhas
_context.Personagens.UpdateRange(personagens);
await _context.SaveChangesAsync();
```

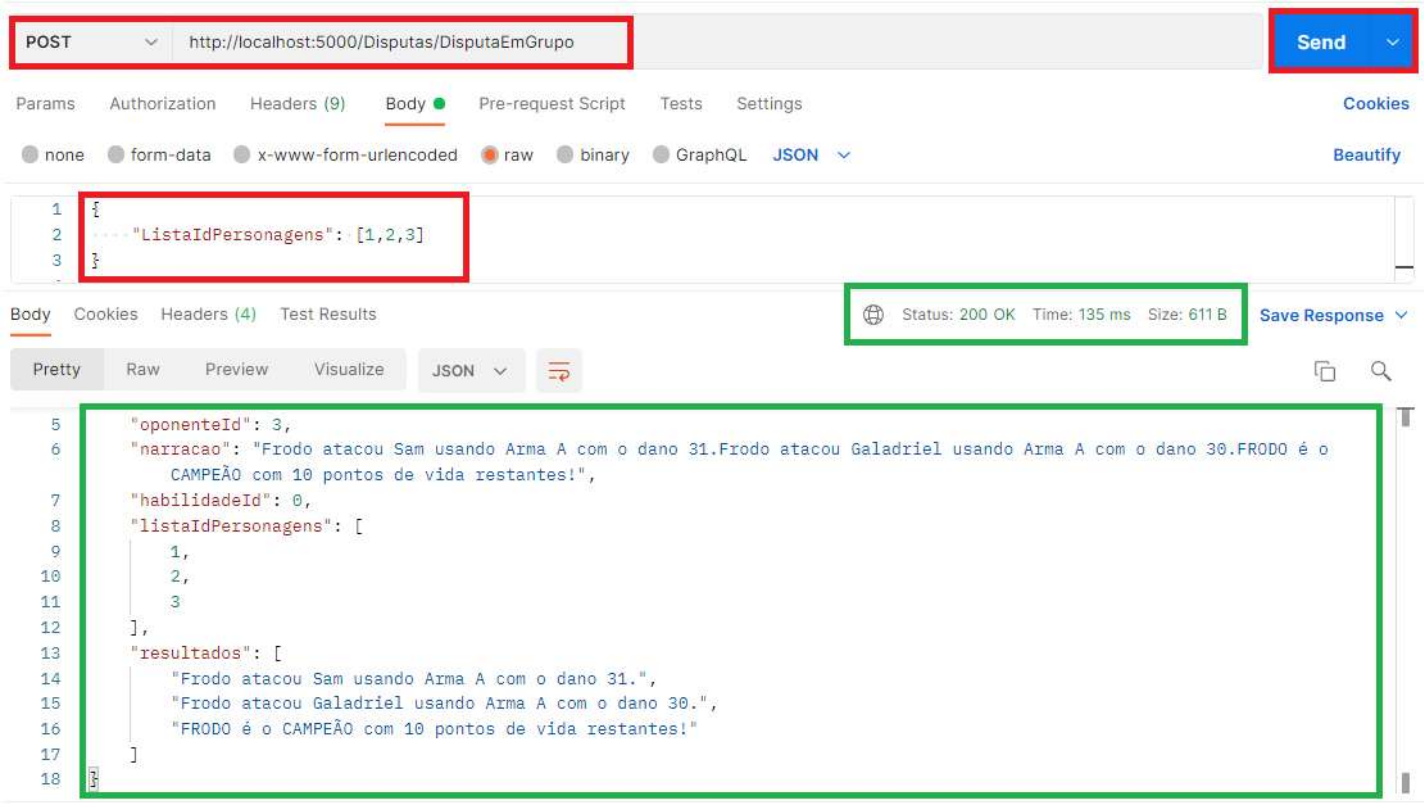
- A próxima codificação deverá ficar abaixo da sinalização do retângulo verde.

5. Após o fechamento do else, faremos a programação a seguir.

```
    //Atenção: Aqui ficará a Programação da verificação do ataque usado e verificação se existe mais de um personagem vivo
    if (!string.IsNullOrEmpty(ataqueUsado)) A
    {
        //Incrementa os dados dos combates
        atacante.Vitorias++;
        oponente.Derrotas++; B
        atacante.Disputas++;
        oponente.Disputas++;

        d.Id = 0;//Zera o Id para poder salvar os dados de disputa sem erro de chave.
        d.DataDisputa = DateTime.Now;
        _context.Disputas.Add(d);       C
        await _context.SaveChangesAsync();
    }

    qtdPersonagensVivos = personagens.FindAll(p => p.PontosVida > 0).Count; D

    if (qtdPersonagensVivos == 1)//Havendo só um personagem vivo, existe um CAMPEÃO!  E
    {
        string resultadoFinal =
            $"{atacante.Nome.ToUpper()} é CAMPEÃO com {atacante.PontosVida} pontos de vida restantes!";
    F
        d.Narracao += resultadoFinal; //Concatena o resultado final com as demais narrações.
        d.Resultados.Add(resultadoFinal); //Contatena o resultado final com os demais resultados.

        break; //break vai parar o While.
    }
}//Fim do While
//Código após o fechamento do While. Atualizará os pontos de vida,
//disputas, vitórias e derrotas de todos personagens ao final das batalhas
```

(A) Verificação se o ataque usado teve resultado, já que ele não existirá caso o personagem não tenha arma nem habilidades.
(B) Incremento dos dados de disputa do atacante e do oponente.
(C) Preparação para salvar os dados de disputa no banco de dados.
(D) Contagem dos personagens que ainda tem pontuação de vida positiva.
(E) Caso apenas um personagem se enquadre nesta situação, quer dizer que ele será o campeão.
(F) Preparação das mensagens finais e comando break para interromper o While.

6. Para realizar o teste no postman, basta realizar o envio dos Ids dos personagens separados por vírgula e dentro de colchetes, assim a lista chamada ListaIdPersonagens da classe Disputa será carregada quando os dados forem deserealizados.



- Para permitir disputas mais duradoras, você pode variar os valores do dano das armas e habilidades e a força, inteligência e defesa dos personagens, além de relacionar mais personagens para a disputa, conforme o body enviado no postman. Tenha Armas e habilidade diversificadas nas tabelas e associadas corretamente a cada Personagem.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2025/2

Luiz Fernando Souza / Quitéria Danno

Métodos para complementar as operações na API – Disponíveis para cópia – A cada método inserido realize a compilação e verifique se não faltará nenhum using.

Copie e adicione na Classe controller **DisputasController** – Apagar Disputas

```
        [HttpDelete("ApagarDisputas")]
        public async Task<IActionResult> DeleteAsync()
        {
            try
            {
                List<Disputa> disputas = await _context.TB_DISPUTAS.ToListAsync();

                _context.TB_DISPUTAS.RemoveRange(disputas);
                await _context.SaveChangesAsync();

                return Ok("Disputas apagadas");
            }
            catch (System.Exception ex)
            {                return BadRequest(ex.Message); }
        }
```

Copie e adicione na Classe controller **DisputasController** – Listar Disputas

```
        [HttpGet("Listar")]
        public async Task<IActionResult> ListarAsync()
        {
            try
            {
                List<Disputa> disputas =
                   await _context.TB_DISPUTAS.ToListAsync();

                return Ok(disputas);
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

Copie e adicione na Classe controller **PersonagensController** – Restaurar Pontos de Vida

```
[HttpPut("RestaurarPontosVida")]
public async Task<IActionResult> RestaurarPontosVidaAsync(Personagem p)
{
    try
    {
        int linhasAfetadas = 0;
        Personagem? pEncontrado =

        await _context.Personagens.FirstOrDefaultAsync(pBusca => pBusca.Id == p.Id);
        pEncontrado.PontosVida = 100;

        bool atualizou = await TryUpdateModelAsync<Personagem>(pEncontrado, "p",
            pAtualizar => pAtualizar.PontosVida);
        // EF vai detectar e atualizar apenas as colunas que foram alteradas.
        if (atualizou)
            linhasAfetadas = await _context.SaveChangesAsync();

        return Ok(linhasAfetadas);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

```
//Método para alteração da foto
[HttpPut("AtualizarFoto")]
public async Task<IActionResult> AtualizarFotoAsync(Personagem p)
{
    try
    {
        Personagem personagem = await _context.TB_PERSONAGENS
           .FirstOrDefaultAsync(x => x.Id == p.Id);
        personagem.FotoPersonagem = p.FotoPersonagem;
        var attach = _context.Attach(personagem);
        attach.Property(x => x.Id).IsModified = false;
        attach.Property(x => x.FotoPersonagem).IsModified = true;
        int linhasAfetadas = await _context.SaveChangesAsync();
        return Ok(linhasAfetadas);
    }
    catch (System.Exception ex)
    { return BadRequest(ex.Message);}}
```

Copie e adicione na Classe controller **PersonagensController** – Zerar Ranking

```
[HttpPut("ZerarRanking")]
        public async Task<IActionResult> ZerarRankingAsync(Personagem p)
        {
            try
            {
                Personagem pEncontrado =
                  await _context.TB_PERSONAGENS.FirstOrDefaultAsync(pBusca => pBusca.Id == p.Id);

                pEncontrado.Disputas = 0;
                pEncontrado.Vitorias = 0;
                pEncontrado.Derrotas = 0;
                int linhasAfetadas = 0;

                bool atualizou = await TryUpdateModelAsync<Personagem>(pEncontrado, "p",
                    pAtualizar => pAtualizar.Disputas,
                    pAtualizar => pAtualizar.Vitorias,
                    pAtualizar => pAtualizar.Derrotas);

                // EF vai detectar e atualizar apenas as colunas que foram alteradas.
                if (atualizou)
                    linhasAfetadas = await _context.SaveChangesAsync();

                return Ok(linhasAfetadas);
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

Copie e adicione na Classe controller **PersonagensController** – Zerar ranking geral e restaurar vidas geral

```
[HttpPut("ZerarRankingRestaurarVidas")]
public async Task<IActionResult> ZerarRankingRestaurarVidasAsync()
{
    try
    {
        List<Personagem> lista =
        await _context.TB_PERSONAGENS.ToListAsync();

        foreach (Personagem p in lista)
        {
            await ZerarRankingAsync(p);
            await RestaurarPontosVidaAsync(p);
        }
        return Ok();
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

Copie e adicione na Classe controller **PersonagensController** o método para buscar os personagens de acordo com o Id de um usuário informado

```
[HttpGet("GetByUser/{userId}")]
public async Task<IActionResult> GetByUserAsync(int userId)
{
    try
    {
        List<Personagem> lista = await _context.TB_PERSONAGENS
                    .Where(u => u.Usuario.Id == userId)
                    .ToListAsync();

        return Ok(lista);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2025/2

Luiz Fernando Souza / Quitéria Danno

Copie e adicione na Classe controller **PersonagensController** um método para verificar o perfil do usuário, trazendo todos se ele for administrador, ou trazendo só os que ele cadastrou caso seja Jogador.

```
        [HttpGet("GetByPerfil/{userId}")]
        public async Task<IActionResult> GetByPerfilAsync(int userId)
        {
            try
            {
                Usuario usuario = await _context.TB_USUARIOS
                   .FirstOrDefaultAsync(x => x.Id == userId);

                List<Personagem> lista = new List<Personagem>();
                if (usuario.Perfil == "Admin")
                    lista = await _context.TB_PERSONAGENS.ToListAsync();
                else
                    lista = await _context.TB_PERSONAGENS
                            .Where(p => p.Usuario.Id == userId).ToListAsync();
                return Ok(lista);
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

Copie e adicione na Classe controller **PersonagensController** um método para busca aproximada do nome do personagem.

```
[HttpGet("GetByNomeAproximado/{nomePersonagem}")]
        public async Task<IActionResult> GetByNomeAproximado(string nomePersonagem)
        {
            try
            {
                List<Personagem> lista = await _context.TB_PERSONAGENS
                    .Where(p => p.Nome.ToLower().Contains(nomePersonagem.ToLower()))
                    .ToListAsync();

                return Ok(lista);
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2025/2

Luiz Fernando Souza / Quitéria Danno

Copie e adicione na Classe controller UsuariosController.cs os métodos para buscar o usuário por Id e por login, e para atualizar a geolocalização e atualizar o e-mail

```
        [HttpGet("{usuarioId}")]
        public async Task<IActionResult> GetUsuario(int usuarioId)
        {
            try
            {
                //List exigirá o using System.Collections.Generic
                Usuario usuario = await _context.TB_USUARIOS //Busca o usuário no banco através do Id
                   .FirstOrDefaultAsync(x => x.Id == usuarioId);

                return Ok(usuario);
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

```
        [HttpGet("GetByLogin/{login}")]
        public async Task<IActionResult> GetUsuario(string login)
        {
            try
            {
                //List exigirá o using System.Collections.Generic
                Usuario usuario = await _context.TB_USUARIOS //Busca o usuário no banco através do login
                   .FirstOrDefaultAsync(x => x.Username.ToLower() == login.ToLower());

                return Ok(usuario);
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

```
        //Método para alteração da geolocalização
        [HttpPut("AtualizarLocalizacao")]
        public async Task<IActionResult> AtualizarLocalizacao(Usuario u)
        {
            try
            {
                Usuario usuario = await _context.TB_USUARIOS //Busca o usuário no banco através do Id
                   .FirstOrDefaultAsync(x => x.Id == u.Id);

                usuario.Latitude = u.Latitude;
                usuario.Longitude = u.Longitude;

                var attach = _context.Attach(usuario);
                attach.Property(x => x.Id).IsModified = false;
                attach.Property(x => x.Latitude).IsModified = true;
                attach.Property(x => x.Longitude).IsModified = true;

                int linhasAfetadas = await _context.SaveChangesAsync(); //Confirma a alteração no banco
                return Ok(linhasAfetadas); //Retorna as linhas afetadas (Geralmente sempre 1 linha msm)
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

```
[HttpPut("AtualizarEmail")]
public async Task<IActionResult> AtualizarEmail(Usuario u)
{
    try
    {
        Usuario usuario = await _context.TB_USUARIOS //Busca o usuário no banco através do Id
           .FirstOrDefaultAsync(x => x.Id == u.Id);

        usuario.Email = u.Email;

        var attach = _context.Attach(usuario);
        attach.Property(x => x.Id).IsModified = false;
        attach.Property(x => x.Email).IsModified = true;

        int linhasAfetadas = await _context.SaveChangesAsync(); //Confirma a alteração no banco
        return Ok(linhasAfetadas); //Retorna as linhas afetadas (Geralmente sempre 1 linha msm)
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

```
        //Método para alteração da foto
        [HttpPut("AtualizarFoto")]
        public async Task<IActionResult> AtualizarFoto(Usuario u)
        {
            try
            {
                Usuario usuario = await _context.TB_USUARIOS
                   .FirstOrDefaultAsync(x => x.Id == u.Id);

                usuario.Foto = u.Foto;

                var attach = _context.Attach(usuario);
                attach.Property(x => x.Id).IsModified = false;
                attach.Property(x => x.Foto).IsModified = true;

                int linhasAfetadas = await _context.SaveChangesAsync();
                return Ok(linhasAfetadas);
            }
            catch (System.Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2025/2

Luiz Fernando Souza / Quitéria Danno

Validação ao salvar uma arma: Alterar o método que adiciona uma arma no contexto do banco de dados.

```
[HttpPost]
0 references
public async Task<IActionResult> Add(Arma novaArma)
{
    try
    {
        if(novaArma.Dano == 0)
          throw new Exception("O Dano da arma não pode ser 0");

        Personagem? p = await _context.TB_PERSONAGENS.FirstOrDefaultAsync(p => p.Id == novaArma.PersonagemId);

        if(p == null)
            throw new Exception("Não existe personagem com o Id informado.");

        Arma buscaArma = await _context.TB_ARMAS
A           .FirstOrDefaultAsync(a => a.PersonagemId == novaArma.PersonagemId);

        if(buscaArma != null)
B           throw new Exception("O Personagem selecionado já contém uma arama atribuída a ele.");

        await _context.TB_ARMAS.AddAsync(novaArma);
        await _context.SaveChangesAsync();

        return Ok(novaArma.Id);
    }
    catch (System.Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

(A) Busca na tabela de Armas uma arma com o Id de personagem informado.
(B) Se achar, quer dizer que o personagem não pode ter mais que uma arma (relacionamento 1 para 1)

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2025/2

Luiz Fernando Souza / Quitéria Danno

Autenticação via Token com JSON Web Tokens (JWT)

Nossas aplicações até o momento tem todos os métodos programados com acesso livre na controller, mas podemos fazer com que os métodos das controllers sejam acessados apenas se houver uma autenticação válida, porém, esse processo acarretaria a necessidade de entrar com as credenciais a cada método solicitado.

Para facilitar os procedimentos, podemos utilizar um token que pode ficar armazenado no navegador de uma aplicação ou de um dispositivo, sendo que este token conterá informações essenciais de identificação de um usuário, mas não informações sensíveis que podem ser roubadas, além disso, o token pode ser criado já com uma data de validade atribuída.

JSON Web Tokens é um recurso muito útil para tudo isso que falamos até então e faremos a aplicação prática na API, mas para que conheça um pouco da teoria do que estamos falando, segue um vídeo para introdução sobre Token e JWT:

Token: https://youtu.be/LtVb9rhU41c

JWT: https://youtu.be/Gyq-yeot8qM

Artigo: https://www.brunobrito.net.br/jwt-cookies-oauth-bearer/

O JWT nada mais é do que uma série de caracteres que contém especificações sobre um usuário, chamamos essas especificações de Claims (Reivindicações), ou seja, através desses dados é possível saber quais os tipos de acesso que determinado usuário poderá ter numa API, por exemplo, ou simplesmente resgar informações armazenadas no token gerado.

1. Abra o projeto RpgApi, vá até o arquivo appsettings.json e insira a codificação que será a chave para gerar o token

```
{
  "ConfiguracaoToken":{
    "Chave":"minha chave super secreta minha chave super secreta minha chave super secreta"
  },

  "ConnectionStrings": {
    "ConexaoLocal": "Data Source=localhost; Initial Catalog=DB-DS-DS_2023_2; User Id=sa; Pass
```

2. Utilize o terminal para instalar os pacotes necessários para programação do método que vai gerar o Token:
   - ➔ dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer (-v 8.0.8 se for versão .net 8)
   - ➔ dotnet add package System.IdentityModel.Tokens.Jwt (- v 8.1.0 se for versão .net 8)
   - ➔ dotnet add package Microsoft.IdentityModel.Tokens (- v 8.1.0 se for versão .net 8)

   Perceba que ao abrir o arquivo RpgApi.csProj, as referências aos pacotes estarão dentro da tag **ItemGroup** conforme a seguir:

```
<PackageReference Include="Microsoft.IdentityModel.Tokens" Version="8.1.0"/>
<PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="8.1.0"/>
<PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="8.0.8"/>
```