ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

**PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2025**

*Luiz Fernando Souza / Eliane Marion*

---

**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

**PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2025**

*Luiz Fernando Souza / Eliane Marion*

---

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);
            }
        }
```

*Luiz Fernando Souza / Eliane Marion*

---

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);
            }
        }
```

*Luiz Fernando Souza / Eliane Marion*

---

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)