

# ASPNET - 6 - Blog Pessoal - Documentar

1. Introdução
2. Documentando Blog Pessoal - Parte 1
3. Documentando Blog Pessoal - Parte 2

## 1. Introdução

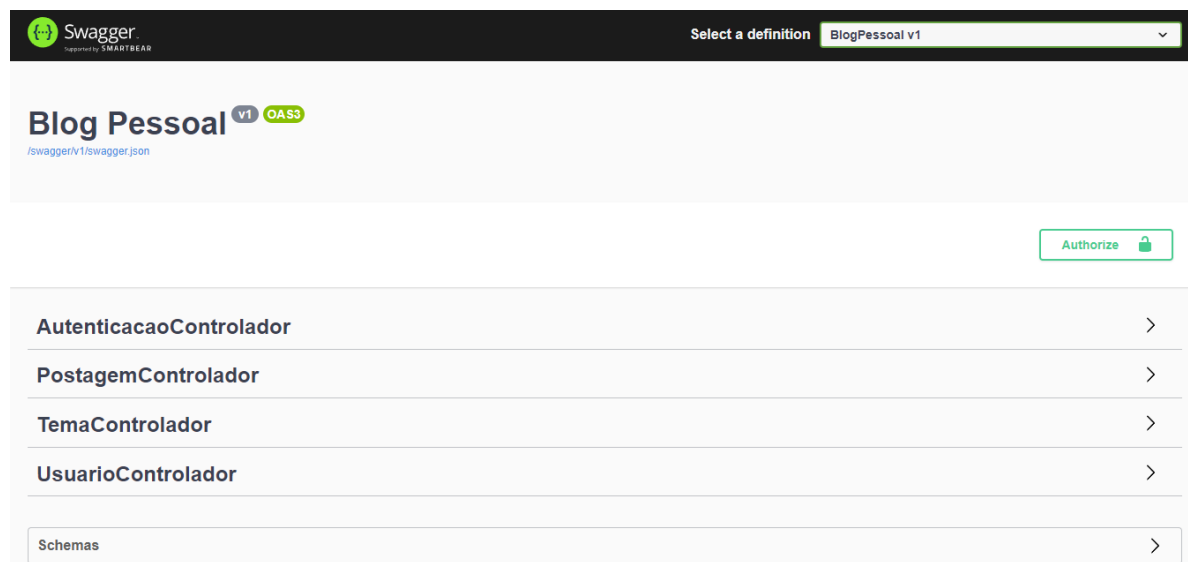
Uma API bem construída deve ser simples de ser consumida, por isso é fundamental documentar a API da maneira correta. O objetivo da Documentação é apresentar uma espécie de “manual de instruções” que permita que qualquer pessoa (Desenvolvedora ou não), que ainda não tenha pleno conhecimento do domínio da sua aplicação, consuma a API de maneira fácil, rápida, eficaz e autônoma.

## OpenApi

A OpenApi, trata-se de uma especificação Open Source, que auxilia as pessoas desenvolvedoras nos processos de definir, criar, documentar e consumir API's REST e RESTful. A OpenApi tem por objetivo padronizar este tipo de integração, descrevendo os recursos que uma API possui, os respectivos endpoints, os dados que serão solicitados nas Requisições, os dados que serão retornados nas Respostas, os Status HTTP, os modelos de dados, os métodos de autenticação, entre outros. APIs REST são frequentemente usadas para a integração de aplicações, seja para consumir serviços de terceiros, seja para prover novos serviços. Para estas APIs, a especificação OpenApi facilita a modelagem, a documentação e a geração de código.

## Swagger

O Swagger é uma poderosa ferramenta que ajuda pessoas Desenvolvedoras a projetar, desenvolver, documentar e consumir serviços web REST e RESTful, de forma interativa e dinâmica, aproveitando ao máximo todos os Recursos da especificação OpenApi. Apesar de ser conhecida principalmente por sua interface Swagger UI, o software inclui suporte para documentação automatizada da API, geração de código e testes.



The screenshot shows the Swagger UI interface for 'Blog Pessoal v1'. At the top, there's a header with the Swagger logo and a dropdown menu to 'Select a definition' with 'BlogPessoal v1' selected. Below the header, the title 'Blog Pessoal' is displayed with 'v1' and 'OAS3' badges, and a link to '/swagger/v1/swagger.json'. An 'Authorize' button is visible on the right. The main content area lists several API endpoints with expandable details:

- AutenticacaoControlador** >
- PostagemControlador** >
- TemaControlador** >
- UsuarioControlador** >
- Schemas** >

## 2. Documentando Blog Pessoal - Parte 1

### Instalando pacote de documentação:

Dentro do diretório de `BlogPessoal/` executar o comando abaixo para instalar o pacote que dá suporte ao *swagger*. O comando deve ser executado no diretório em que se encontra o arquivo *BlogPessoal.csproj*.

```
dotnet add package Swashbuckle.AspNetCore -v 5.6.3
```

### Incluir tags em PropertyGroup (/BlogPessoal.csproj):

Dentro do arquivo *BlogPessoal.csproj*, localizado na raiz do diretório incluir duas `tags` abaixo no dentro da tag `PropertyGroup` :

```
<GenerateDocumentationFile>true</GenerateDocumentationFile>
<NoWarn>$(NoWarn);1591</NoWarn>
```

A `tag` ficara da seguinte forma:

```
<PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <NoWarn>$(NoWarn);1591</NoWarn>
</PropertyGroup>
```

Tags	Descrição
<code>&lt;GenerateDocumentationFile&gt;true&lt;/GenerateDocumentationFile&gt;</code>	Permite gerar arquivo de documentação dentro da pasta de configuração do projeto, utilizando documentação criada dentro do projeto;
<code>&lt;NoWarn&gt;\$(NoWarn);1591&lt;/NoWarn&gt;</code>	Faz com que varios alertas do tipo 1591 desapareçam para não prejudicar o projeto.

### Incluir serviço do SwaggerGen no método ConfigureServices (Startup.cs):

O serviço do Swagger é responsável por gerar a interface cujo controlador será demonstrado. Inserir a configuração dentro do documento *Startup.cs* no método `ConfigureServices` após bloco de configuração do Token, o trecho de código abaixo:

```
// Configuração Swagger
services.AddSwaggerGen(
    s =>
    {
```

```

s.SwaggerDoc("v1", new OpenApiInfo { Title = "Blog Pessoal", Version =
"v1" });

s.AddSecurityDefinition(
    "Bearer",
    new OpenApiSecurityScheme()
    {
        Name = "Authorization",
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer",
        BearerFormat = "JWT",
        In = ParameterLocation.Header,
        Description = "JWT authorization header utiliza: Bearer + JWT
Token",
    }
);

s.AddSecurityRequirement(
    new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            new List<string>()
        }
    }
);

var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
s.IncludeXmlComments(xmlPath);
}
);

```

Para compreender o código podemos pensar nos pequenos blocos de código dentro do serviço `services.AddSwaggerGen`:

Definição	Explicação
<code>s.SwaggerDoc</code>	Define o título e versão do cabeçalho da página;
<code>s.AddSecurityDefinition</code>	Adiciona esquema de Autorização no formato Bearer, e define um botão na página que será utilizado para passar o Token no formato <code>Bearer + Token</code> ;
<code>s.AddSecurityRequirement</code>	Define esquema de como é apresentado a requisição pelo navegador para as requisições;
<code>s.IncludeXmlComments</code>	Inclui documentação especificada em <code>xml</code> no código C# para dentro do Swagger.

## Incluir Swagger no método Configure (Startup.cs):

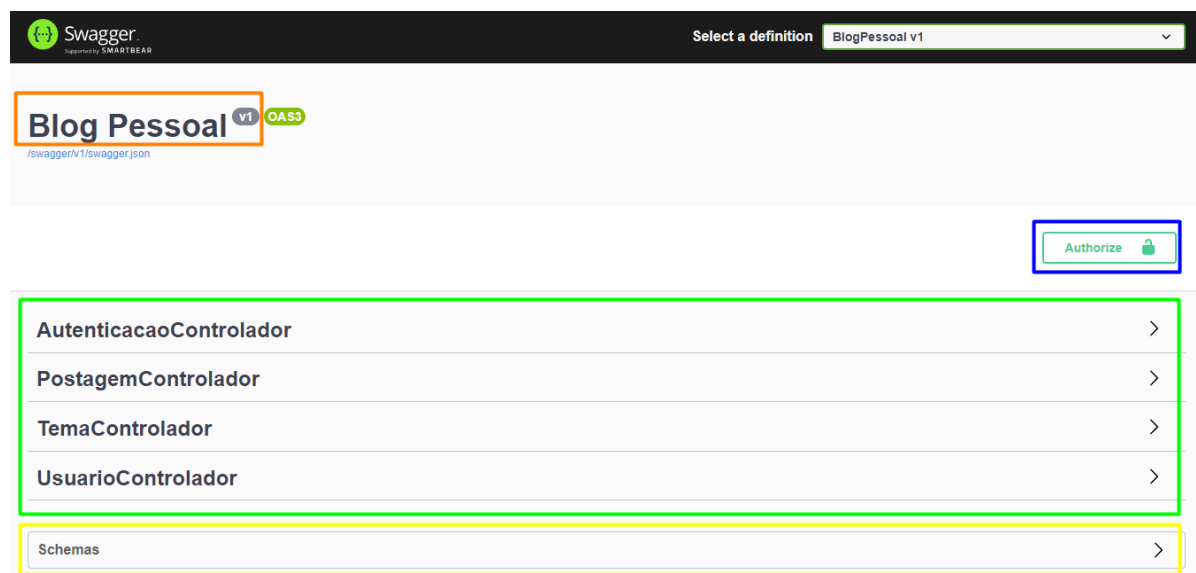
Inserir a configuração dentro do documento *Startup.cs* no método `Configure` dentro do bloco de condição de ambiente (`env.IsDevelopment()`), o trecho de código abaixo:

```
app.UseSwagger();
app.UseSwaggerUI(c => {
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "BlogPessoal v1");
    c.RoutePrefix = string.Empty;
});
```

O bloco de definição de ambiente dentro do método `Configure` ficará da seguinte maneira:

```
// Ambiente de Desenvolvimento
if (env.IsDevelopment())
{
    contexto.Database.EnsureCreated();
    app.UseDeveloperExceptionPage();
    app.UseSwagger();
    app.UseSwaggerUI(c => {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "BlogPessoal v1");
        c.RoutePrefix = string.Empty;
    });
}
```

Com esta configuração já é suficiente para que nossa API seja documentada. Execute a aplicação para conferir o resultado acessando o a url `https://localhost:5001/swagger/index.html`.



Em destaque estão os campos :

Cor	Definição
Laranja	Título e versão de nossa API;
Azul	Botão para fornecer <i>Authorization</i> para as requisições;
Verde	Controladores do sistema;
Amarelo	Os esquemas, entidades ou modelos que nosso sistema representa.

### 3. Documentando Blog Pessoal - Parte 2

Para fornecer uma documentação mais rica é possível incluir ao Swagger a documentação especificada em nossas classe e métodos abaixo segue o exemplo de como documentar Classes, DTOs, Interfaces, Métodos e Endpoints:

Documentando Classes:

Tag de documentação	Usabilidade
<code>&lt;summary&gt;Título&lt;/summary&gt;</code>	É uma boa opção para utilizar como títulos;
<code>&lt;para&gt;Explicação&lt;/para&gt;</code>	Utilizado para definir e explicar algo;
<code>&lt;param name="parametro"&gt;Descrição do parametro&lt;/param&gt;</code>	Aponta um parâmetro de um método;
<code>&lt;return&gt;TipoDeRetorno&lt;/return&gt;</code>	Aponta o tipo de retorno de um método;
<code>&lt;response code="200"&gt;Retorna o usuario&lt;/response&gt;</code>	Aponta o código de resposta de um endpoint;
<code>&lt;remarks&gt;Exemplo de requisição&lt;/remarks&gt;</code>	Fornece espaço para descrever um exemplo de requisição;
<code>[ProducesResponseType(StatusCodes.Status200Ok)]</code>	Aponta um formato para o status de retorno de uma requisição;

Depois de validar o exemplos abaixo, execute novamente a aplicação e confira o que foi alterado na documentação.

### Exemplo de documentação de classe em UsuarioModelo

```
using BlogPessoal.src.utilidades;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.Json.Serialization;
```

```

namespace BlogPessoal.src.modelos
{
    /// <summary>
    /// <para>Resumo: Classe responsavel por representar tb_usuarios no banco.
    </para>
    /// <para>Criado por: Gustavo Boaz</para>
    /// <para>Versão: 1.0</para>
    /// <para>Data: 12/05/2022</para>
    /// </summary>
    [Table("tb_usuarios")]
    public class UsuarioModelo
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required, StringLength(50)]
        public string Nome { get; set; }

        [Required, StringLength(30)]
        public string Email { get; set; }

        [Required, StringLength(30)]
        public string Senha { get; set; }

        public string Foto { get; set; }

        [Required]
        public TipoUsuario Tipo { get; set; }

        [JsonIgnore]
        public List<PostagemModelo> MinhasPostagens { get; set; }
    }
}

```

## Exemplo de documentação de dto em UsuarioDTO

```

using BlogPessoal.src.utilidades;
using System.ComponentModel.DataAnnotations;

namespace BlogPessoal.src.dtos
{
    /// <summary>
    /// <para>Resumo: Classe espelho para criar um novo usuario</para>
    /// <para>Criado por: Gustavo Boaz</para>
    /// <para>Versão: 1.0</para>
    /// <para>Data: 29/04/2022</para>
    /// </summary>
    public class NovoUsuarioDTO
    {
        [Required, StringLength(50)]
        public string Nome { get; set; }

        [Required, StringLength(30)]
        public string Email { get; set; }
    }
}

```

```

        [Required, StringLength(30)]
        public string Senha { get; set; }

        public string Foto { get; set; }

        [Required]
        public TipoUsuario Tipo { get; set; }

        public NovoUsuarioDTO(string nome, string email, string senha, string
foto, TipoUsuario tipo)
        {
            Nome = nome;
            Email = email;
            Senha = senha;
            Foto = foto;
            Tipo = tipo;
        }
    }

    /// <summary>
    /// <para>Resumo: Classe espelho para alterar um novo usuario</para>
    /// <para>Criado por: Gustavo Boaz</para>
    /// <para>Versão: 1.0</para>
    /// <para>Data: 29/04/2022</para>
    /// </summary>
    public class AtualizarUsuarioDTO
    {
        [Required]
        public int Id { get; set; }

        [Required, StringLength(50)]
        public string Nome { get; set; }

        [Required, StringLength(30)]
        public string Senha { get; set; }

        public string Foto { get; set; }

        public AtualizarUsuarioDTO(int id, string nome, string senha, string
foto)
        {
            Id = id;
            Nome = nome;
            Senha = senha;
            Foto = foto;
        }
    }
}

```

## Exemplo de documentação de interface de IUsuario

```

using System.Collections.Generic;
using System.Threading.Tasks;
using BlogPessoal.src.dtos;
using BlogPessoal.src.modelos;

namespace BlogPessoal.src.repositorios

```

```

{
    /// <summary>
    /// <para>Resumo: Responsavel por representar ações de CRUD de
usuario</para>
    /// <para>Criado por: Gustavo Boaz</para>
    /// <para>Versão: 1.0</para>
    /// <para>Data: 29/04/2022</para>
    /// </summary>
    public interface IUsuario
    {
        Task<UsuarioModelo> PegarUsuarioPeloIdAsync(int id);
        Task<List<UsuarioModelo>> PegarUsuariosPeloNomeAsync(string nome);
        Task<UsuarioModelo> PegarUsuarioPeloEmailAsync(string email);
        Task NovoUsuarioAsync(NovoUsuarioDTO usuario);
        Task AtualizarUsuarioAsync(AtualizarUsuarioDTO usuario);
        Task DeletarUsuarioAsync(int id);
    }
}

```

## Exemplo de documentação de métodos em UsuarioRepositorio

```

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using BlogPessoal.src.data;
using BlogPessoal.src.dtos;
using BlogPessoal.src.modelos;
using Microsoft.EntityFrameworkCore;

namespace BlogPessoal.src.repositorios.implementacoes
{
    /// <summary>
    /// <para>Resumo: Classe responsavel por implementar IUsuario</para>
    /// <para>Criado por: Gustavo Boaz</para>
    /// <para>Versão: 1.0</para>
    /// <para>Data: 12/05/2022</para>
    /// </summary>
    public class UsuarioRepositorio : IUsuario
    {
        #region Atributos

        private readonly BlogPessoalContexto _contexto;

        #endregion Atributos

        #region Construtores

        public UsuarioRepositorio(BlogPessoalContexto contexto)
        {
            _contexto = contexto;
        }

        #endregion Construtores

        #region Métodos

```



```

    /// <summary>
    /// <para>Resumo: Método assíncrono para pegar um usuario pelo Id</para>
    /// </summary>
    /// <param name="id">Id do usuario</param>
    /// <return>UsuarioModelo</return>
    public async Task<UsuarioModelo> PegarUsuarioPeloIdAsync(int id)
    {
        return await _contexto.Usuarios.FirstOrDefaultAsync(u => u.Id ==
id);
    }

    /// <summary>
    /// <para>Resumo: Método assíncrono para pegar usuarios pelo nome</para>
    /// </summary>
    /// <param name="nome">Nome do usuario</param>
    /// <return>Lista UsuarioModelo</return>
    public async Task<List<UsuarioModelo>> PegarUsuariosPeloNomeAsync(string
nome)
    {
        return await _contexto.Usuarios
            .Where(u => u.Nome.Contains(nome))
            .ToListAsync();
    }

    /// <summary>
    /// <para>Resumo: Método assíncrono para pegar um usuario pelo
email</para>
    /// </summary>
    /// <param name="email">Email do usuario</param>
    /// <return>UsuarioModelo</return>
    public async Task<UsuarioModelo> PegarUsuarioPeloEmailAsync(string
email)
    {
        return await _contexto.Usuarios.FirstOrDefaultAsync(u => u.Email ==
email);
    }

    /// <summary>
    /// <para>Resumo: Método assíncrono para salvar um novo usuario</para>
    /// </summary>
    /// <param name="usuario">NovoUsuarioDTO</param>
    public async Task NovoUsuarioAsync(NovoUsuarioDTO usuario)
    {
        await _contexto.Usuarios.AddAsync(new UsuarioModelo
        {
            Email = usuario.Email,
            Nome = usuario.Nome,
            Senha = usuario.Senha,
            Foto = usuario.Foto,
            Tipo = usuario.Tipo
        });

        await _contexto.SaveChangesAsync();
    }

    /// <summary>
    /// <para>Resumo: Método assíncrono para atualizar um usuario</para>

```

```

    /// </summary>
    /// <param name="usuario">AtualizarUsuarioDTO</param>
    public async Task AtualizarUsuarioAsync(AtualizarUsuarioDTO usuario)
    {
        var usuarioExistente = await PegarUsuarioPeloIdAsync(usuario.Id);
        usuarioExistente.Nome = usuario.Nome;
        usuarioExistente.Senha = usuario.Senha;
        usuarioExistente.Foto = usuario.Foto;
        _contexto.Usuarios.Update(usuarioExistente);
        await _contexto.SaveChangesAsync();
    }

    /// <summary>
    /// <para>Resumo: Método assíncrono para deletar um usuario</para>
    /// </summary>
    /// <param name="id">Id do usuario</param>
    public async Task DeletarUsuarioAsync(int id)
    {
        _contexto.Usuarios.Remove(await PegarUsuarioPeloIdAsync(id));
        await _contexto.SaveChangesAsync();
    }

    #endregion Métodos
}
}

```

Exemplo de documentação de endpoints do UsuarioControlador

```

using BlogPessoal.src.dtos;
using BlogPessoal.src.modelos;
using BlogPessoal.src.repositorios;
using BlogPessoal.src.servicos;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BlogPessoal.src.controladores
{
    [ApiController]
    [Route("api/Usuarios")]
    [Produces("application/json")]
    public class UsuarioControlador : ControllerBase
    {
        #region Atributos

        private readonly IUsuario _repositorio;
        private readonly IAutenticacao _servicos;

        #endregion

        #region Construtores

        public UsuarioControlador(IUsuario repositorio, IAutenticacao servico)

```

```

{
    _repositorio = repositorio;
    _servicos = servico;
}

#endregion

#region Métodos

/// <summary>
/// Pegar usuario pelo Id
/// </summary>
/// <param name="idUserario">int</param>
/// <returns>ActionResult</returns>
/// <response code="200">Retorna o usuario</response>
/// <response code="404">Usuario não existente</response>
[ProducesResponseType(StatusCodes.Status200Ok, Type =
typeof(UsuarioModelo))]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[HttpGet("id/{idUserario}")]
[Authorize(Roles = "NORMAL,ADMINISTRADOR")]
public async Task<ActionResult<UsuarioModelo>>
PegarUsuarioPeloIdAsync([FromRoute] int idUsuario)
{
    var usuario = await _repositorio.PegarUsuarioPeloIdAsync(idUsuario);

    if (usuario == null) return NotFound();

    return Ok(usuario);
}

/// <summary>
/// Pegar usuario pelo Nome
/// </summary>
/// <param name="nomeUsuario">string</param>
/// <returns>ActionResult</returns>
/// <response code="200">Retorna o usuario</response>
/// <response code="204">Nome não existe</response>
[ProducesResponseType(StatusCodes.Status200Ok, Type =
typeof(UsuarioModelo))]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[HttpGet]
[Authorize(Roles = "NORMAL,ADMINISTRADOR")]
public async Task<ActionResult<List<UsuarioModelo>>>
PegarUsuariosPeloNomeAsync([FromQuery] string nomeUsuario)
{
    var usuarios = await
_repositorio.PegarUsuariosPeloNomeAsync(nomeUsuario);

    if (usuarios.Count < 1) return NoContent();

    return Ok(usuarios);
}

/// <summary>
/// Pegar usuario pelo Email
/// </summary>

```

```

    /// <param name="emailUsuario">string</param>
    /// <returns>ActionResult</returns>
    /// <response code="200">Retorna o usuario</response>
    /// <response code="404">Email não existente</response>
    [ProducesResponseType(StatusCodes.Status200Ok, Type =
typeof(UsuarioModelo))]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    [HttpGet("email/{emailUsuario}")]
    [Authorize(Roles = "NORMAL,ADMINISTRADOR")]
    public async Task<ActionResult<UsuarioModelo>>
PegarUsuarioPeloEmailAsync([FromRoute] string emailUsuario)
    {
        var usuario = await
_repositorio.PegarUsuarioPeloEmailAsync(emailUsuario);

        if (usuario == null) return NotFound();

        return Ok(usuario);
    }

    /// <summary>
    /// Criar novo Usuario
    /// </summary>
    /// <param name="usuario">NovoUsuarioDTO</param>
    /// <returns>ActionResult</returns>
    /// <remarks>
    /// Exemplo de requisição:
    ///
    ///      POST /api/Usuarios
    ///      {
    ///          "nome": "Gustavo Boaz",
    ///          "email": "gustavo@domain.com",
    ///          "senha": "134652",
    ///          "foto": "URLFOTO",
    ///          "tipo": "NORMAL"
    ///      }
    ///
    /// </remarks>
    /// <response code="201">Retorna usuario criado</response>
    /// <response code="400">Erro na requisição</response>
    /// <response code="401">E-mail já cadastrado</response>
    [ProducesResponseType(StatusCodes.Status201Created, Type =
typeof(UsuarioModelo))]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status401Unauthorized)]
    [HttpPost]
    [AllowAnonymous]
    public async Task<ActionResult> NovoUsuarioAsync([FromBody]
NovoUsuarioDTO usuario)
    {
        if(!ModelState.IsValid) return BadRequest();

        try
        {
            await _servicos.CriarUsuarioSemDuplicarAsync(usuario);
            return Created($"api/Usuarios/email/{usuario.Email}", usuario);
        }
        catch (Exception ex)
    }

```

```

        {
            return Unauthorized(ex.Message);
        }
    }

    /// <summary>
    /// Atualizar Usuario
    /// </summary>
    /// <param name="usuario">AtualizarUsuarioDTO</param>
    /// <returns>ActionResult</returns>
    /// <remarks>
    /// Exemplo de requisição:
    ///
    /// PUT /api/Usuarios
    /// {
    ///     "id": 1,
    ///     "nome": "Gustavo Boaz",
    ///     "senha": "134652",
    ///     "foto": "URLFOTO"
    /// }
    ///
    /// </remarks>
    /// <response code="200">Retorna usuario atualizado</response>
    /// <response code="400">Erro na requisição</response>
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [HttpPut]
    [Authorize(Roles = "NORMAL,ADMINISTRADOR")]
    public async Task<ActionResult> AtualizarUsuarioAsync([FromBody]
AtualizarUsuarioDTO usuario)
    {
        if(!ModelState.IsValid) return BadRequest();

        usuario.Senha = _servicos.CodificarSenha(usuario.Senha);

        await _repositorio.AtualizarUsuarioAsync(usuario);
        return Ok(usuario);
    }

    /// <summary>
    /// Deletar usuario pelo Id
    /// </summary>
    /// <param name="idUserario">int</param>
    /// <returns>ActionResult</returns>
    /// <response code="204">Usuario deletado</response>
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [HttpDelete("deletar/{idUserario}")]
    [Authorize(Roles = "ADMINISTRADOR")]
    public async Task<ActionResult> DeletarUsuarioAsync([FromRoute] int
idUserario)
    {
        await _repositorio.DeletarUsuarioAsync(idUsuario);
        return NoContent();
    }

    #endregion
}

```

