Projeto de Exemplo

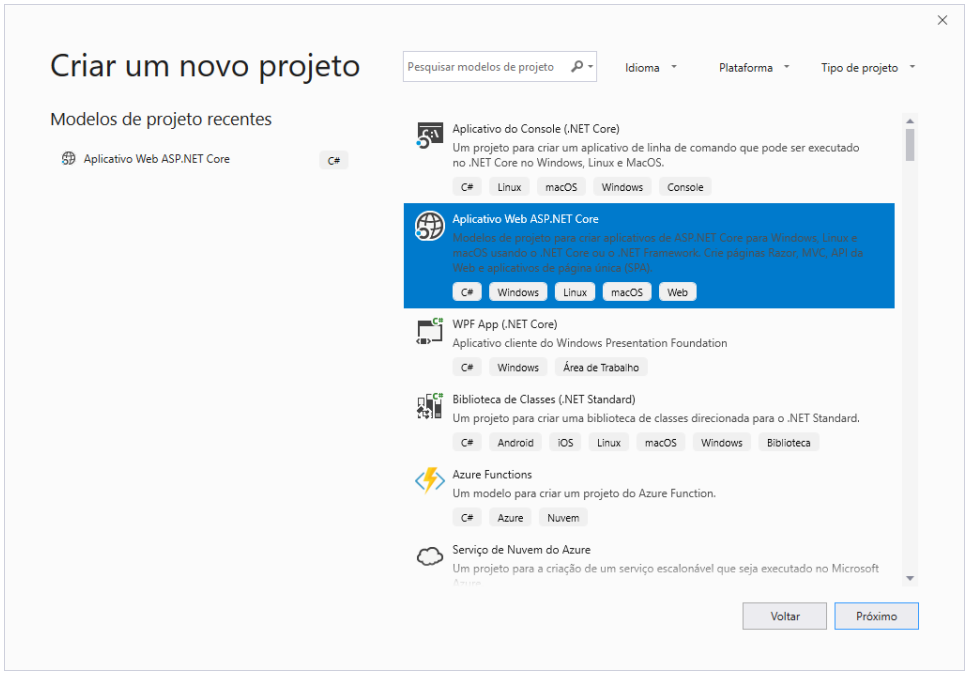
Projeto de exemplo de como utilizar autenticação e autorização e buscar os dados do token. Este projeto será feito utilizando o SqlClient e com um repositório de exemplo utilizando o EFCore.

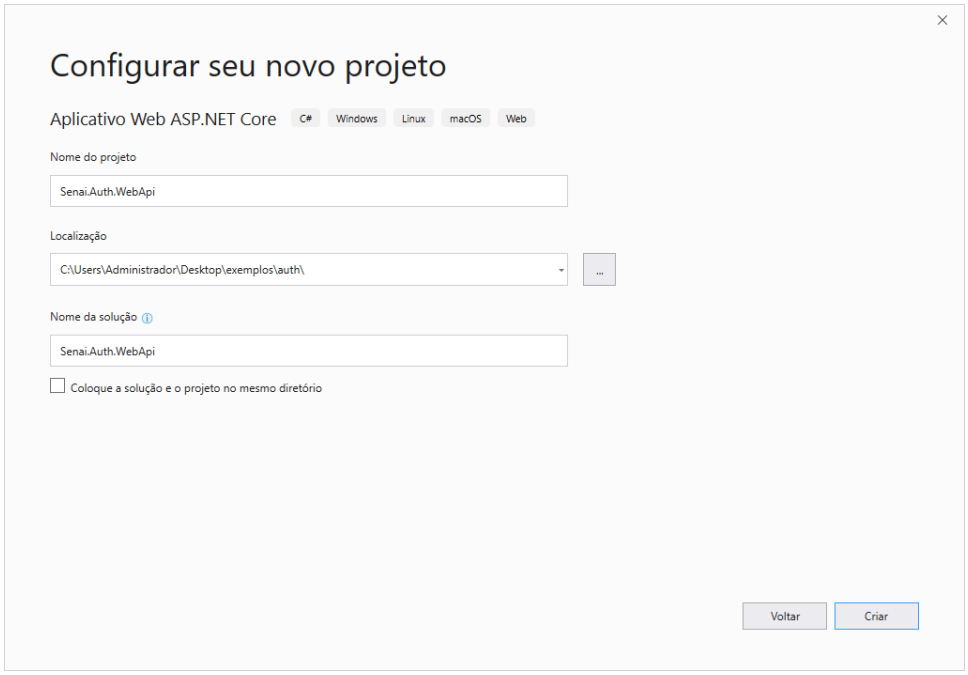
Banco de Dados

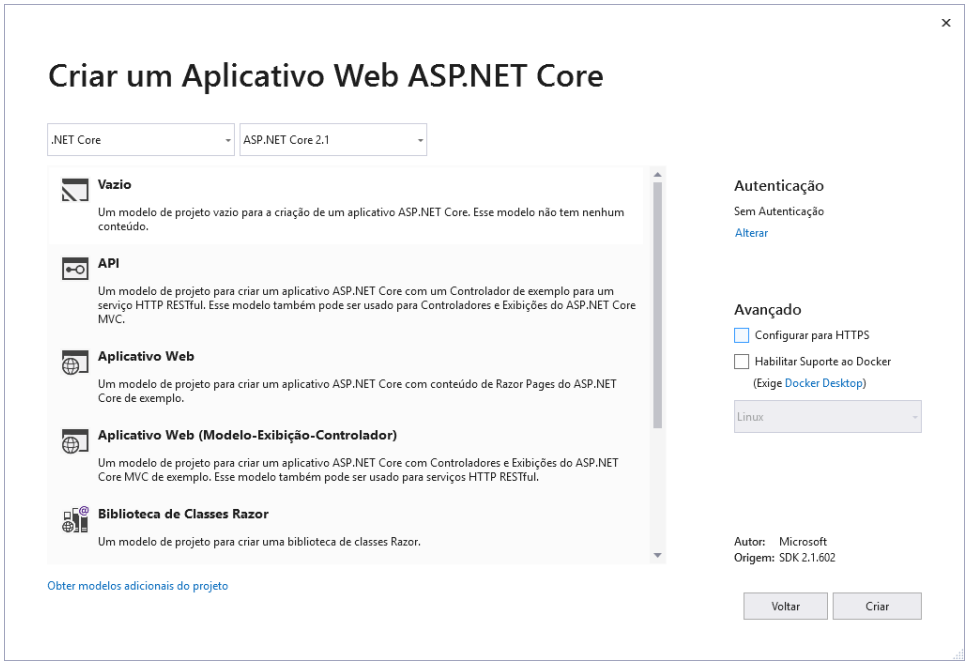
Executar o script e_auth_script.sql.

BackEnd

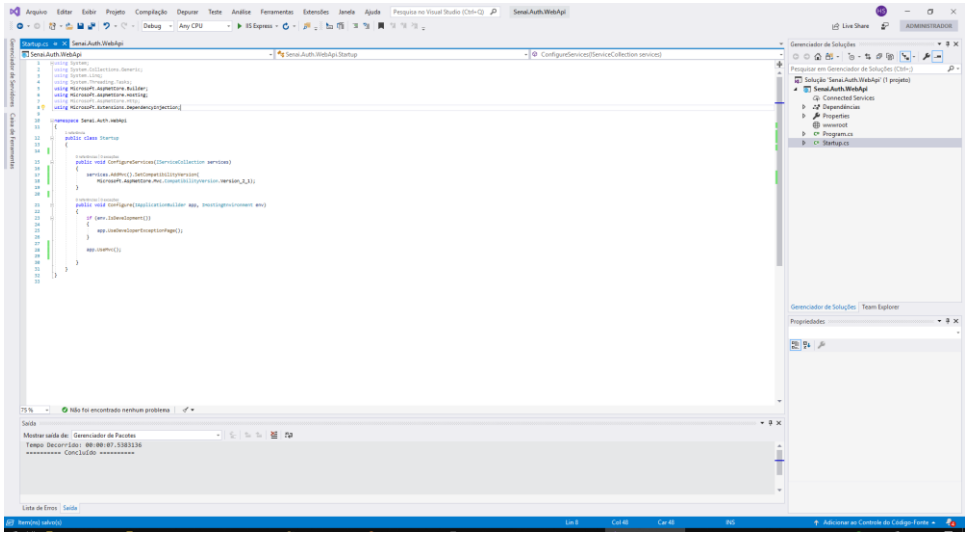
Criar o projeto Senai.Auth.WebApi.







Configurar o mvc no Startup.cs.



Gerenciar pacotes do NuGet e adicionar as seguintes dependências:

Token - JWT

System.IdentityModel.Tokens.Jwt (5.5.0) - criar e validar o jwt

Microsoft.AspNetCore.Authentication.JwtBearer (2.1.1) - integrar a parte de autenticação

EFCore

Microsoft.EntityFrameworkCore.SqlServer (2.1.11)

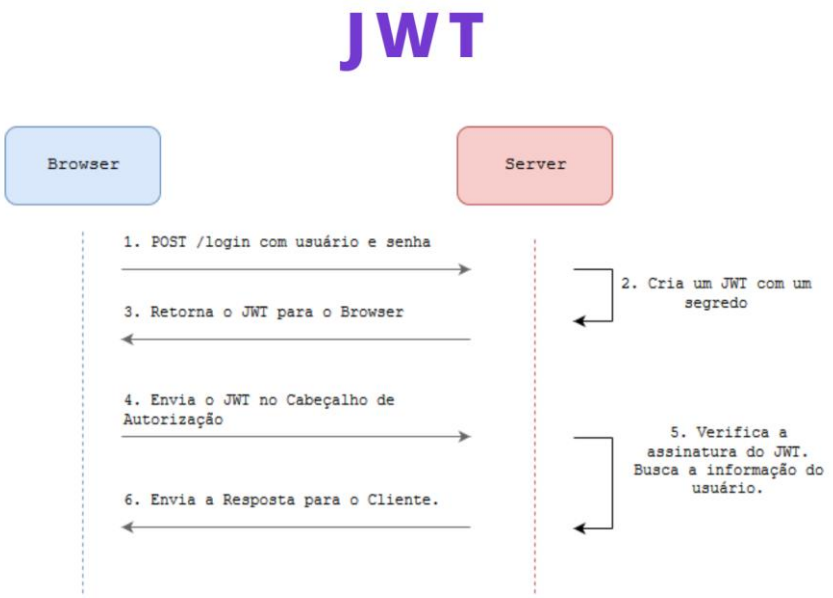
Microsoft.EntityFrameworkCore.SqlServer.Design (1.1.6)

Microsoft.EntityFrameworkCore.Tools (2.1.11)

SqlClient

System.Data.SqlClient (4.6.1)

No processo de autenticação e autorização, os seguintes passos ocorrem:



Vamos supor que eu queria restringir o acesso do endpoint:

GET /api/mensagens

O que eu preciso fazer para isto acontecer?

1. Configurar o Startup.cs.
2. Incluir o [Authorize] no endpoint que eu desejo restringir o acesso.
3. Gerar o token.
    a. Criar um controller para gerar o token;
    b. Criar um método no repositório para buscar o usuário por email e senha.
4. Na próxima requisição GET /api/mensagens, enviar o token gerado no cabeçalho da requisição.

Neste cenário, como o intuito é mostrar a parte de autorização e autenticação, vamos realizar o Scaffold para ganharmos tempo na montagem do processo.

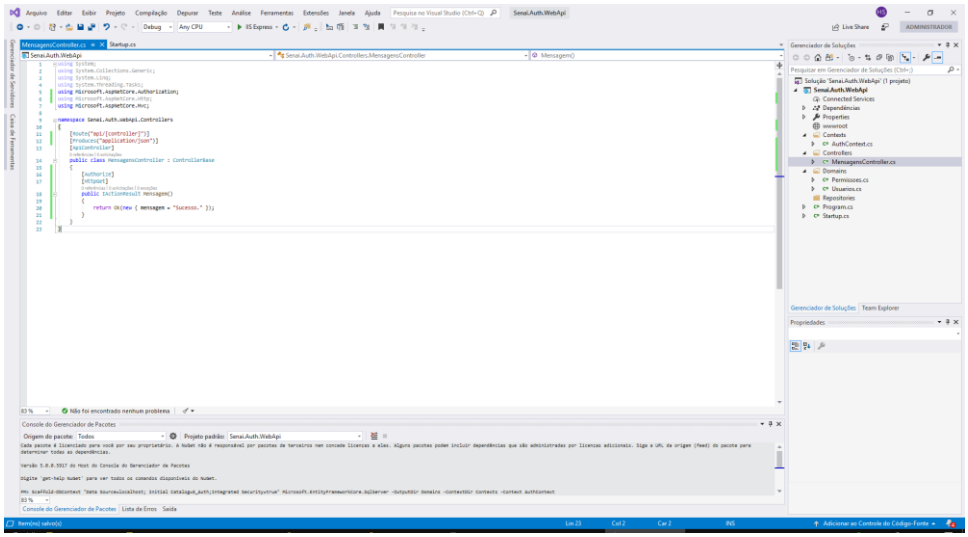
Scaffold-DbContext "Data Source=localhost; Initial Catalog=E_Auth;Integrated Security=true" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Domains -ContextDir Contexts -Context AuthContext

Configurar o startup.

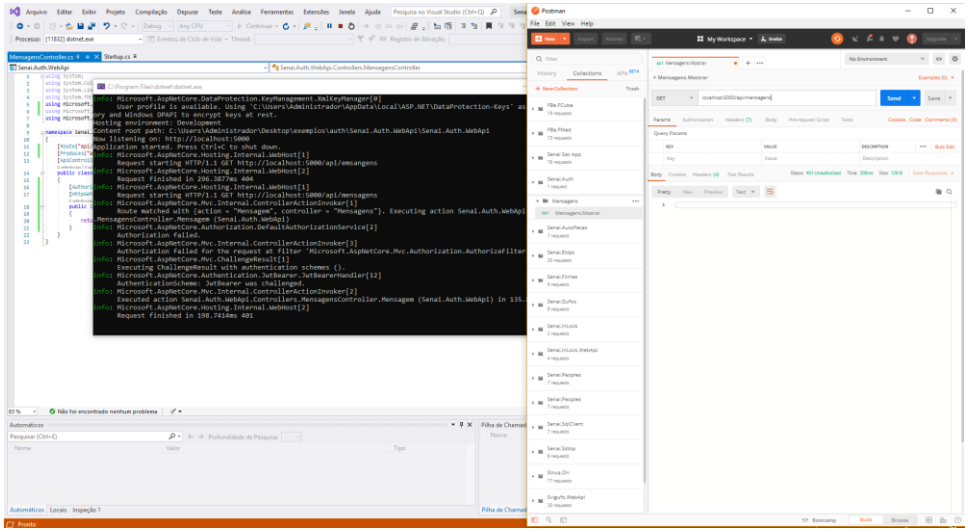
```
0 referências | 0 exceções
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
        .AddJsonOptions
        (
        options => {
            options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;
            options.SerializerSettings.NullValueHandling = Newtonsoft.Json.NullValueHandling.Ignore;
        })
        .SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_1);


    // configurar - adicionar o middleware
    services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = "JwtBearer";
        options.DefaultChallengeScheme = "JwtBearer";
    }).AddJwtBearer("JwtBearer", options =>
    {
        options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters
        {
            // emissor
            ValidateIssuer = true,
            // destinatario do token, representa a aplicacao que ira utiliza-lo
            ValidateAudience = true,
            // tempo
            ValidateLifetime = true,
            // chave de autenticacao
            IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("auth-chave-autenticacao")),
            // tempo de expiracao
            ClockSkew = TimeSpan.FromMinutes(30),
            // emissor valido
            ValidIssuer = "Auth.WebApi",
            // destinatario valido
            ValidAudience = "Auth.WebApi"
        };
    });

    // adicionar o middleware de documentacao do swagger
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Swashbuckle.AspNetCore.Swagger.Info { Title = "Auth API", Version = "v1" });
    });
}
```

```
0 referências | 0 exceções
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    // autenticacao
    app.UseAuthentication();

    // documentacao - registrar
    app.UseSwagger();

    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Auth API V1");
    });

    app.UseMvc();

}
```

Criar um novo controller chamado MensagensController.cs.



A partir do momento que eu restrinjo o acesso com o Authorize, ao tentar realizar uma solicitação no postman, não enviando o token, eu receberei uma mensagem de não autorizado.

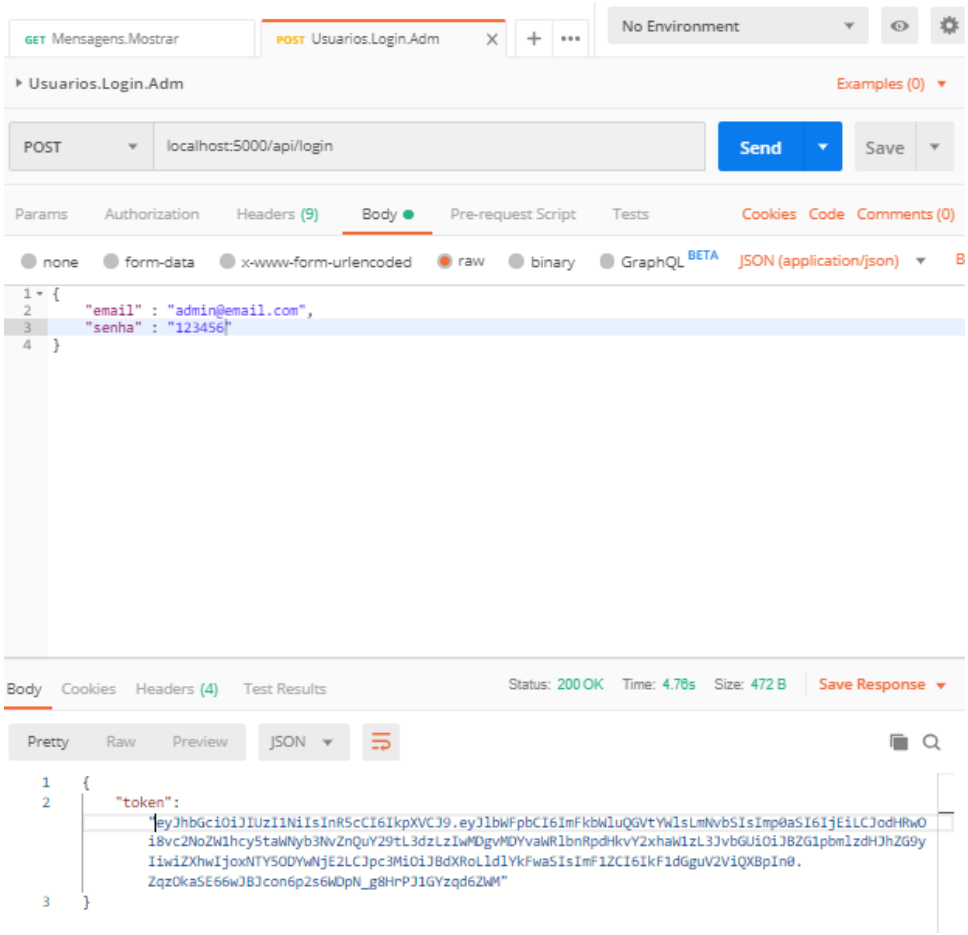


O que precisamos fazer é encaminhar o token na solicitação, mas antes disso, precisamos gerá-lo. De que maneira?

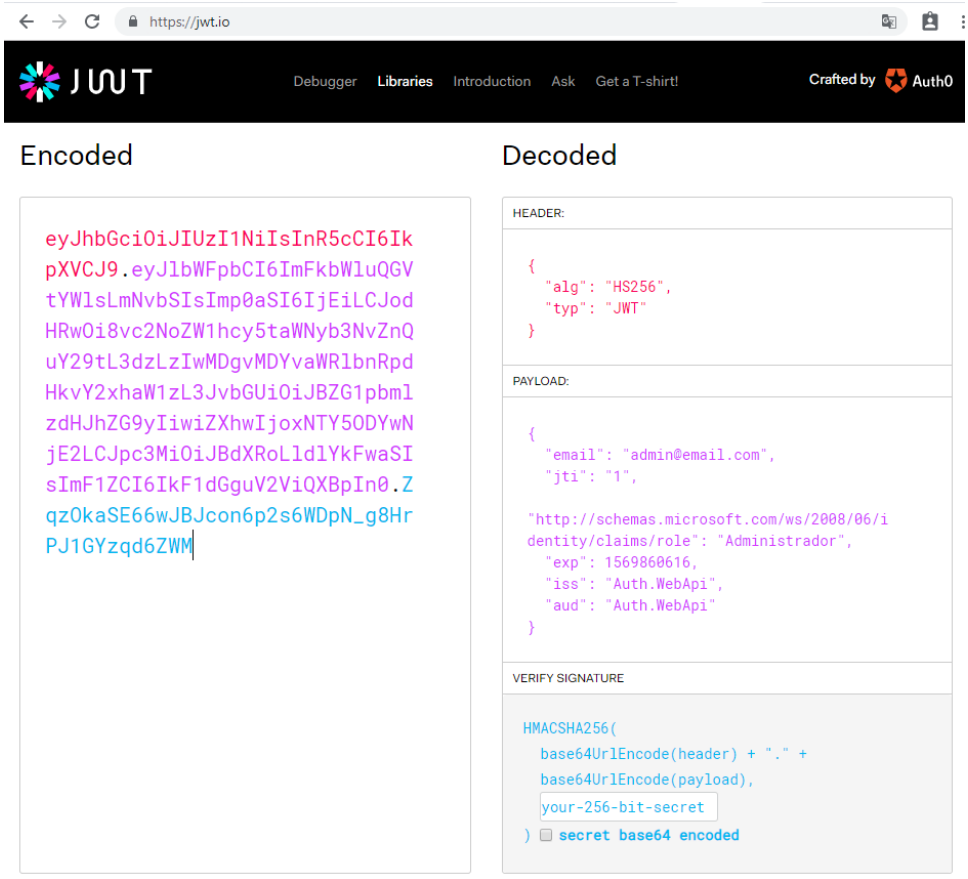
Ao enviarmos um email e senha, verificamos se eles existem no banco de dados, e caso exista, geramos o token com as informações do usuário. Como, por exemplo, email, id e permissão. Não é muito bom incluir a senha, pois este token pode ser descriptografado.

```
UsuarioRepository UsuarioRepository = new UsuarioRepository();

[HttpPost]
0 referências | 0 solicitações | 0 exceções
public IActionResult Login(LoginViewModel login)
{
    try
    {
        Usuarios Usuario = UsuarioRepository.BuscarPorEmailESenha(login);
        if (Usuario == null)
            return NotFound(new { mensagem = "Email ou senha inválidos." });

        // informacoes do usuario
        var claims = new[]
        {
            // email
            new Claim(JwtRegisteredClaimNames.Email, Usuario.Email),
            // id
            new Claim(JwtRegisteredClaimNames.Jti, Usuario.UsuarioId.ToString()),
            // é a permissão do usuário
            new Claim(ClaimTypes.Role, Usuario.Permissao.Nome),
        };

        // chave que tambem esta configurada no startup
        var key = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("auth-chave-autenticacao"));

        // criptografia
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        // eh o proprio token
        var token = new JwtSecurityToken(
            // quem esta mandando e quem esta validando
            issuer: "Auth.WebApi",
            audience: "Auth.WebApi",
            // sao as informacoes do usuario
            claims: claims,
            // data de expiracao
            expires: DateTime.Now.AddDays(30),
            // eh a chave
            signingCredentials: creds);

        // gerar a chave pra vocês
        return Ok(new
        {
            token = new JwtSecurityTokenHandler().WriteToken(token)
        });

    }
    catch (Exception ex)
    {
        return BadRequest(new { mensagem = "Erro." + ex.Message });
    }
}
```

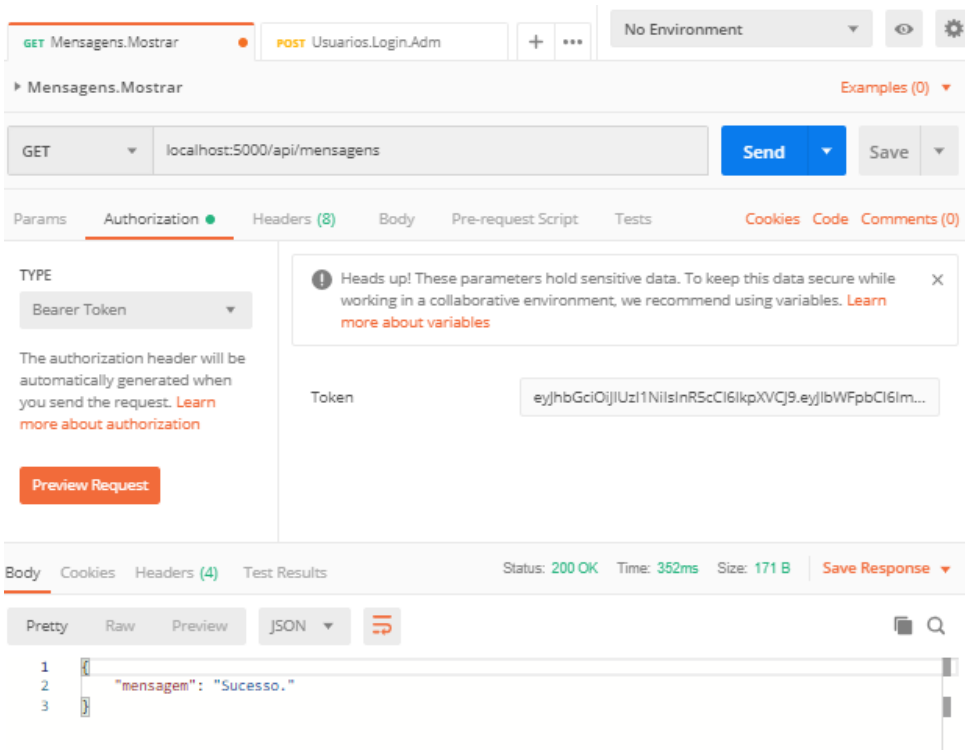
Após realizar o login, teremos o token em mãos.



No site, jwt.io, conseguimos visualizar as informações inseridas dentro dele.

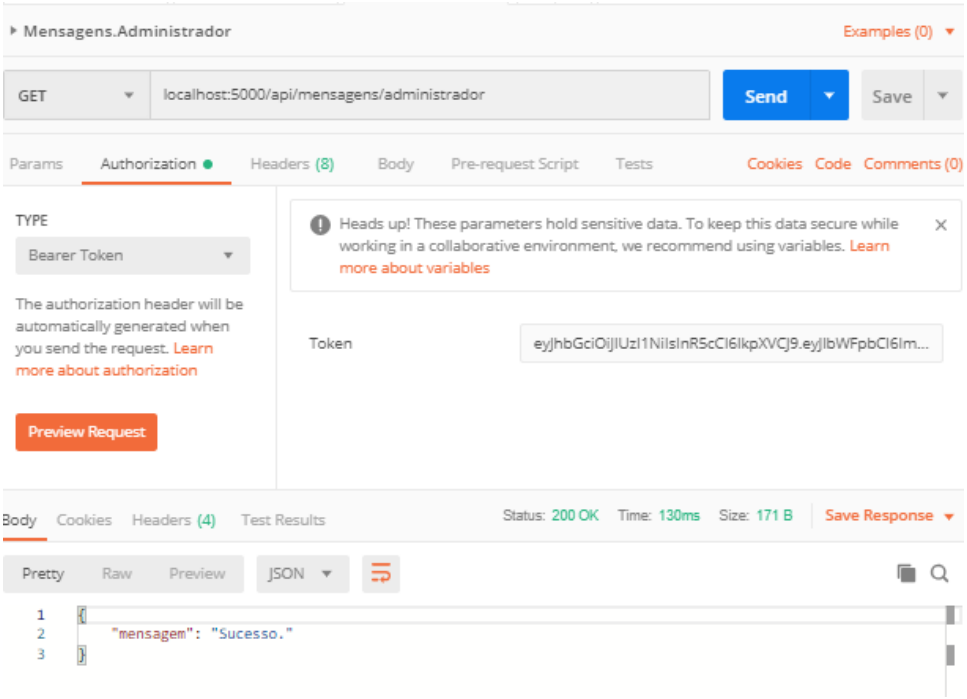


Após termos o token em mãos, ao realizar a próxima requisição para GET /api/mensagens, este usuário terá permissão de visualizar.

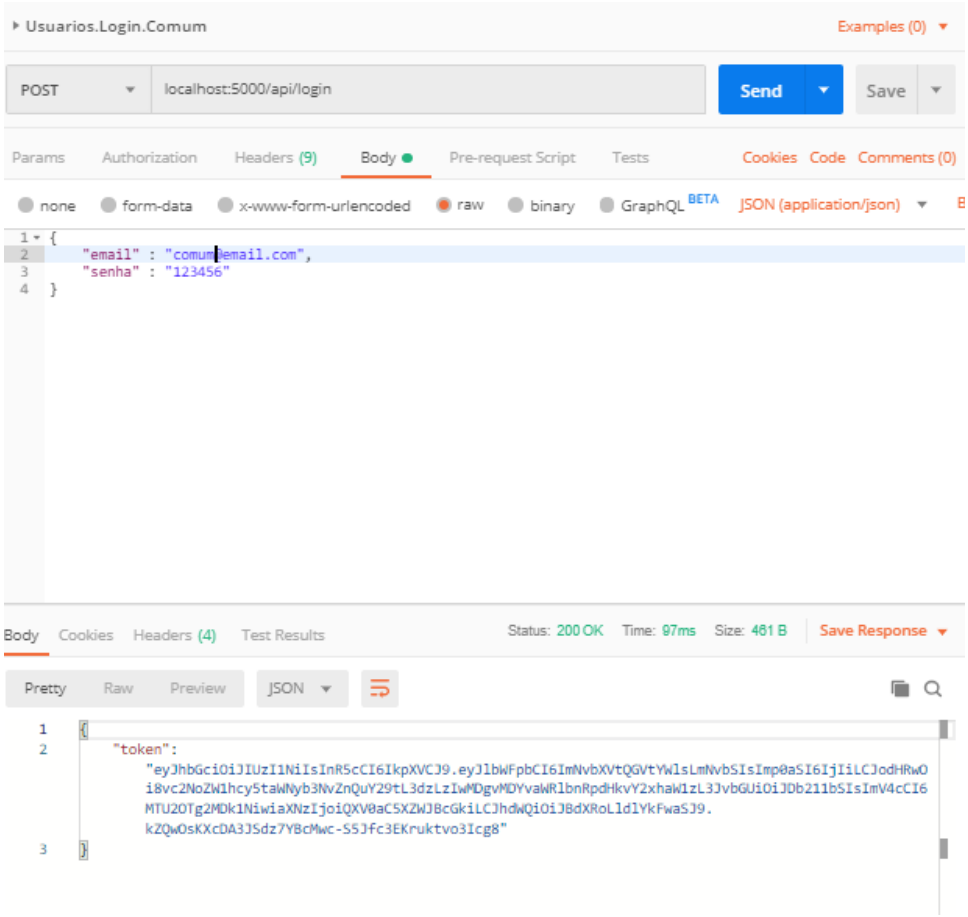


Caso queiramos restringir uma rota de acesso para somente algumas permissões, podemos incluir a permissão da seguinte maneira:

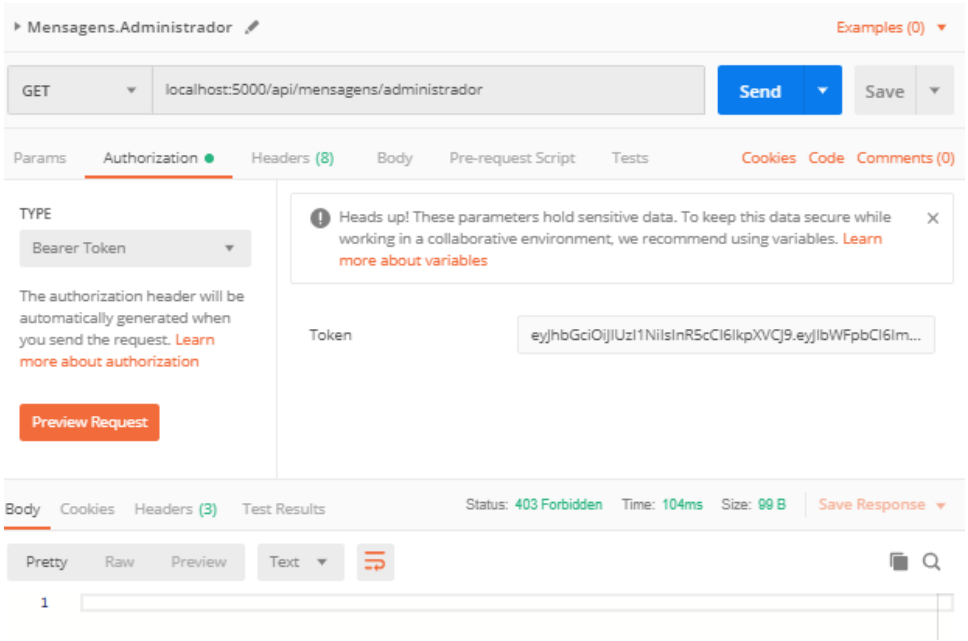
```
[Authorize(Roles = "Administrador")]
[HttpGet("administrador")]
0 referências | 0 solicitações | 0 exceções
public IActionResult Administrador()
{
    return Ok(new { mensagem = "Sucesso." });
}
```



Caso eu tente realizar a passagem de um token com o usuário comum, uma mensagem irá ser mostrada.



Apesar do token ser válido, o usuário não possui autorização para acessar a informação solicitada.

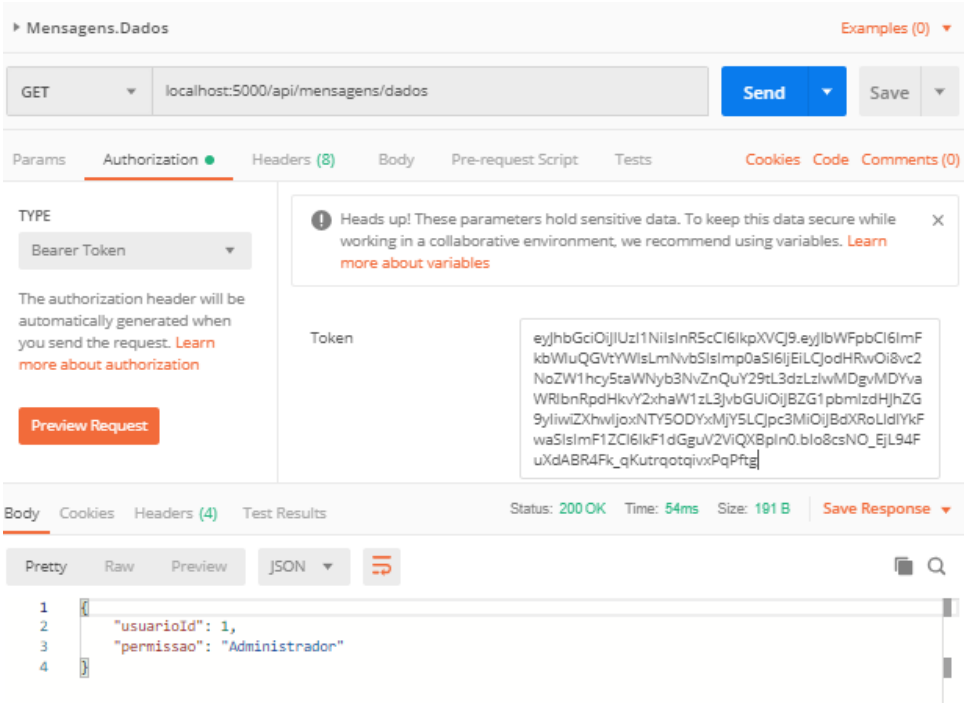


Buscar os dados do token.

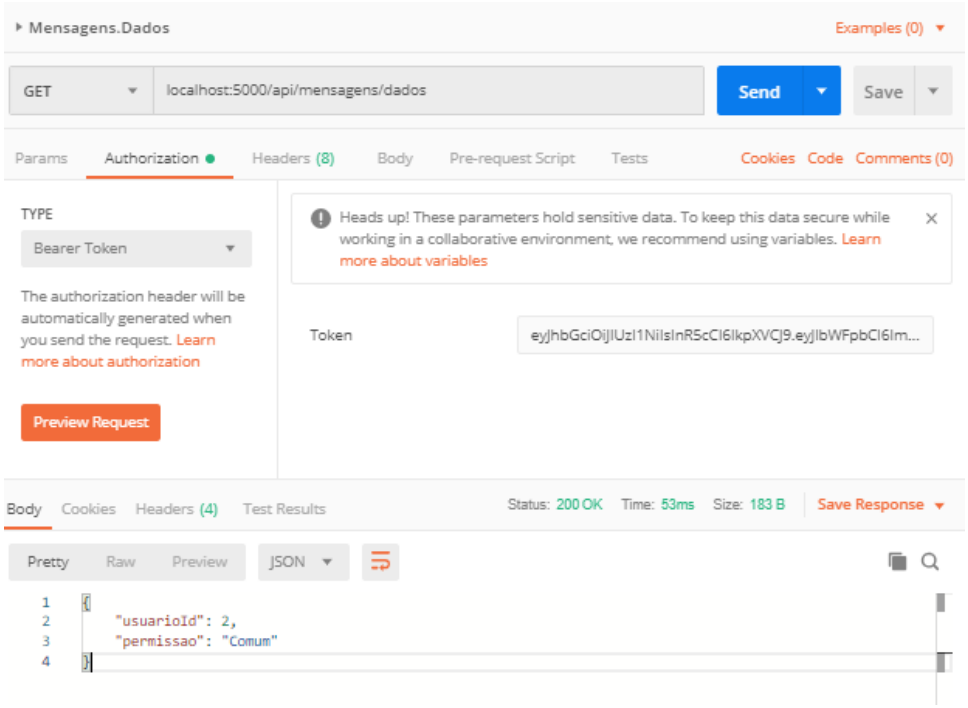
```
[Authorize]
[HttpGet("dados")]
0 referências | 0 solicitações | 0 exceções
public IActionResult DadosDoToken()
{
    int UsuarioId = Convert.ToInt32(HttpContext.User.Claims.First(x => x.Type == JwtRegisteredClaimNames.Jti).Value);
    string UsuarioPermissao = HttpContext.User.Claims.First(x => x.Type == ClaimTypes.Role).Value;
    return Ok(new { UsuarioId = UsuarioId, Permissao = UsuarioPermissao });
}
```

Dessa maneira, após obter os dados do token (lembrando: essas informações foram inseridas no LoginController, na hora de gerar o token), você pode trabalhar com eles da maneira que lhe convir dentro do seu contexto.

Passando os dados de um usuário administrador.



Passando os dados de um usuário comum.



Swagger do projeto.