

Atividade

Criar método de login e CORS

Nesta atividade, você continuará a implementação dos códigos no projeto Web API da empresa **ExoApi**. Nesta etapa, sua tarefa será a implementação do método de login e CORS no projeto.

O método de login necessitará da implementação de um mecanismo de token, para que somente usuários que se autenticarem possam efetuar algumas operações no sistema, como deletar ou atualizar um usuário.

Os testes nesta atividade estão diferentes da anterior, pois serão realizados no aplicativo **Insomnia** ou **Postman**.

Importante

Para realizar este tutorial, é necessário baixar e descompactar o arquivo **ATIVIDADE-06.zip** disponível no AVA.



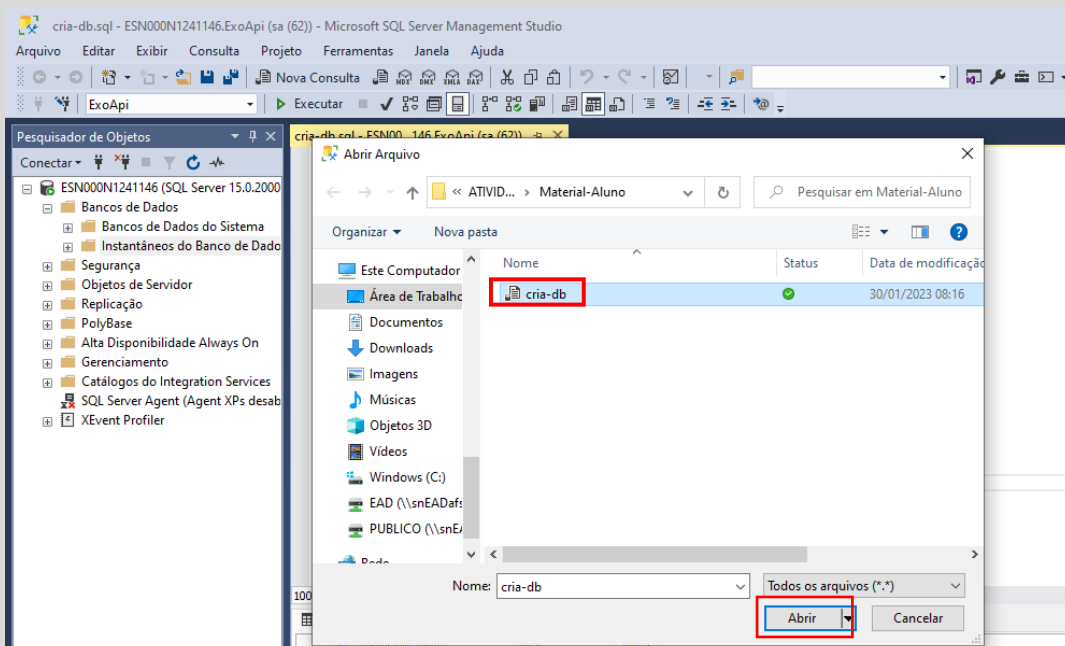
Criando o banco no SSMS

Importante

Os passos a seguir devem ser realizados caso você tenha apagado o banco da atividade anterior ou queira criar um novo banco. Caso já esteja com o banco criado, pule essa etapa e vá para [Preparação dos arquivos no VSCode](#).



1. Abra o SQL Server Management Studio (SSMS). Clique em **Arquivo > Abrir Arquivo...** e localize a pasta baixada para realizar a atividade. Na pasta **Material-Aluno**, selecione o script **cria-db.sql** e clique em **Abrir**.

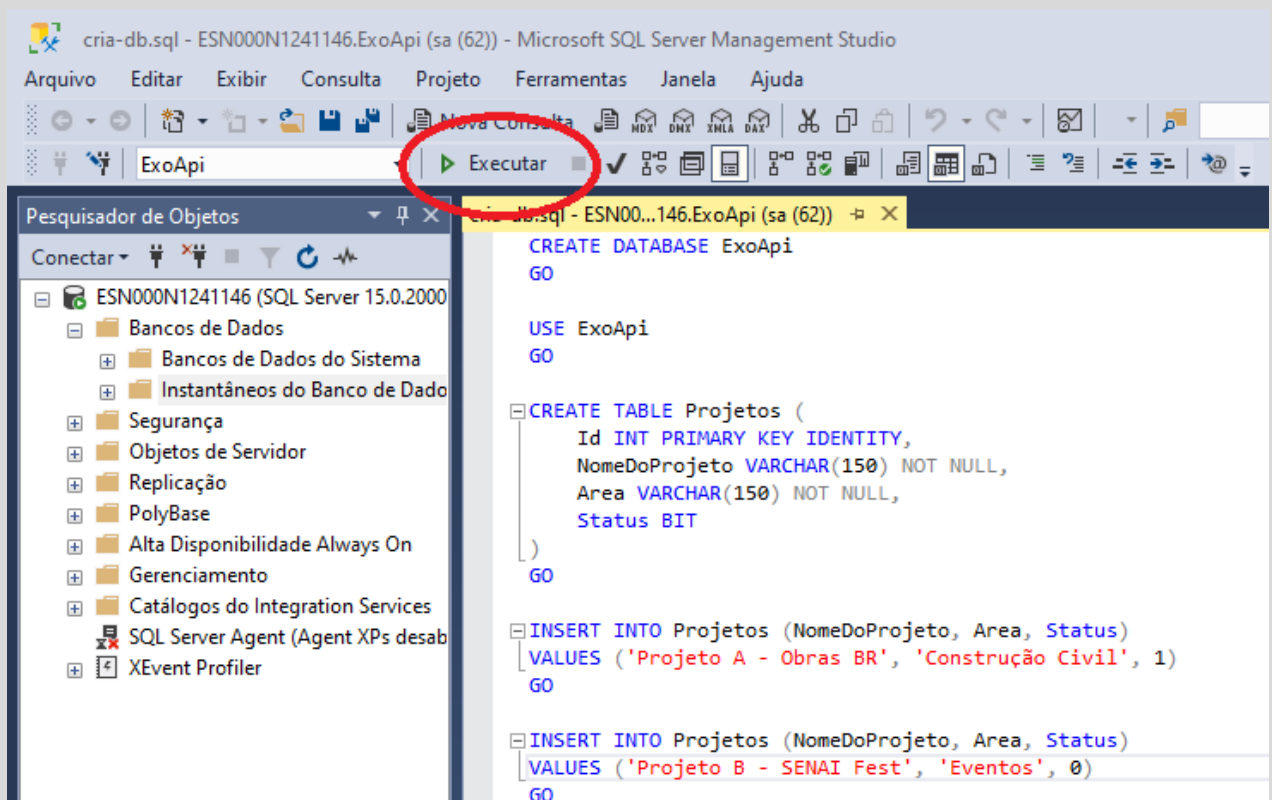


Dica!

Caso já exista um banco de dados e você queira criar um novo, será necessário deletar o existente antes de executar esse script.



2. Com o arquivo **cria-db.sql** aberto, clique em **Executar** para executar o banco e criar os usuários.



Preparação dos arquivos no VSCode

Importante

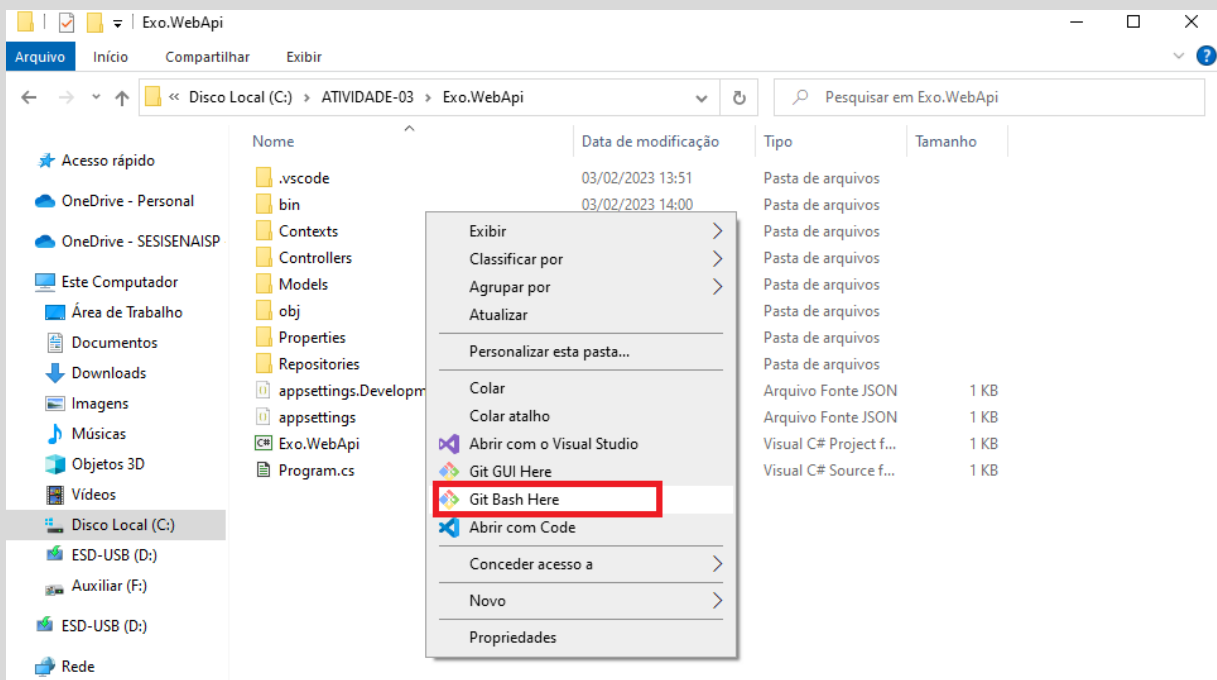
O projeto de API dessa atividade será o mesmo da antecedente. Na etapa anterior, foram realizadas algumas implementações e continuaremos com esse mesmo projeto.

Caso seja necessário, os arquivos estão anexados à atividade (na parte em que paramos).

Caso você queira usar o seu projeto, pule a próxima etapa e vá diretamente para [Inserindo o método Login](#).



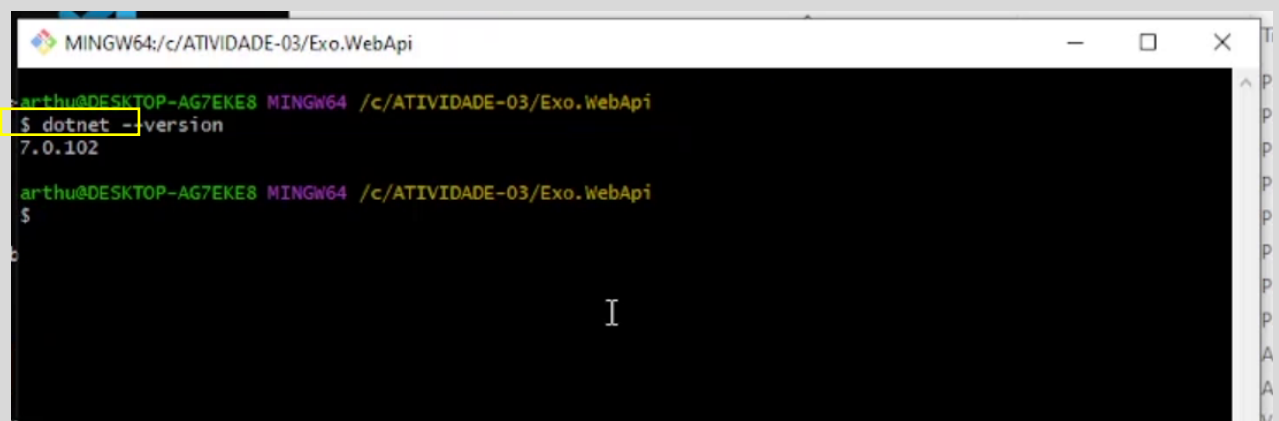
1. Abra a pasta que você usou para baixar a atividade e localize a **Exo.WebApi**. Clique com o botão direito dentro dela e escolha **Git Bash Here** para abrir o terminal.



2. No terminal que será aberto, digite o comando abaixo e dê **Enter** para verificar a versão do dotnet instalada em sua máquina.

```
dotnet --version
```

3. Certifique-se de que sua versão do dotnet seja a 6 ou superior. No nosso caso, a versão é 7.0.102, como mostra a imagem.

A screenshot of a terminal window titled 'MINGW64:/c/ATIVIDADE-03/Exo.WebApi'. The prompt is 'arthu@DESKTOP-AG7EKE8 MINGW64 /c/ATIVIDADE-03/Exo.WebApi'. The command '\$ dotnet --version' is entered, and the output '7.0.102' is displayed. The prompt '\$' is shown again on the next line. A cursor is visible on the line following the prompt.

```
arthu@DESKTOP-AG7EKE8 MINGW64 /c/ATIVIDADE-03/Exo.WebApi
$ dotnet --version
7.0.102
arthu@DESKTOP-AG7EKE8 MINGW64 /c/ATIVIDADE-03/Exo.WebApi
$
```

4. Agora, digite o comando abaixo e dê **Enter** para abrir o VSCode com o projeto já aberto.

```
code .
```

Inserindo o método login

1. Na classe **UsuarioRepository.cs** já disponibilizamos o método login. Acesse a classe e confira se está tudo certo e se ele está implementado. Caso não esteja, insira o código a seguir:

```
public Usuario Login(string email, string senha)
{
    return _context.Usuarios.FirstOrDefault(u => u.Email ==
email && u.Senha == senha);
}
```

2. No arquivo **Exo.WebApi.csproj** também tínhamos adiantado o processo e inserido a referência ao pacote JwtBearer. Então, acesse o arquivo e confira se está tudo certo e se consta essa linha de código. Caso não, insira o código abaixo:

```
<PackageReference
Include="Microsoft.AspNetCore.Authentication.JwtBearer"
Version="6.0.0"/>
```

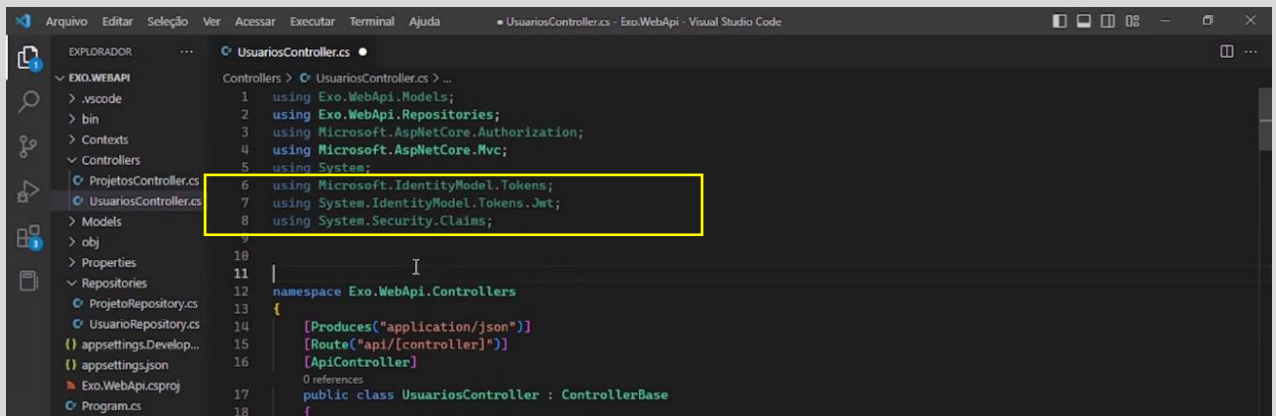
3. Abra o terminal e digite o comando abaixo para consolidar todos os pacotes:

```
Dotnet restore
```

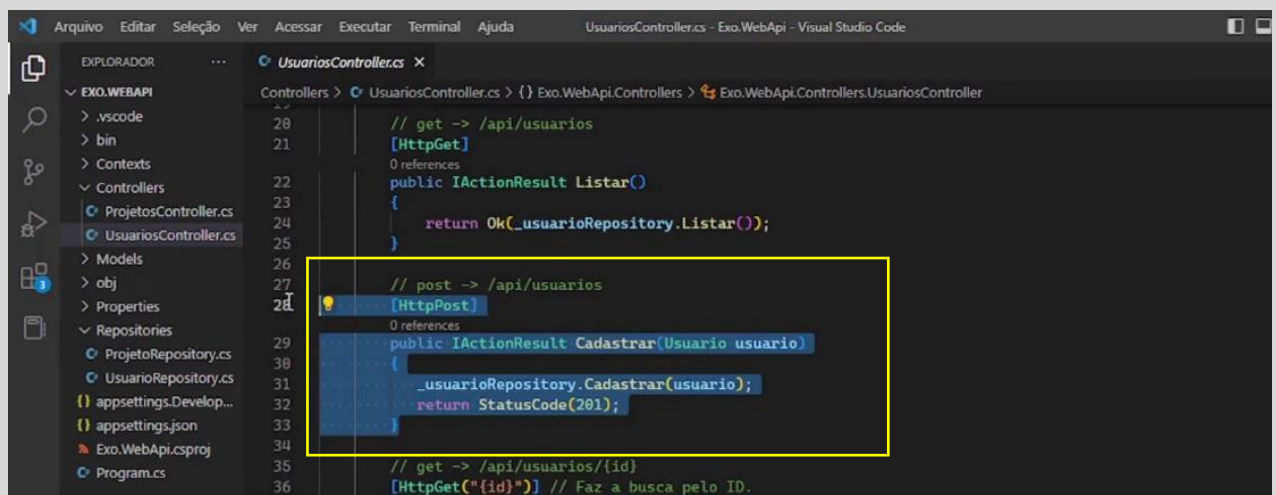
Inserindo o novo método POST

1. Na classe **UsuariosController.cs**, insira as linhas abaixo no local indicado para realizar as novas importações.

```
using Microsoft.IdentityModel.Tokens;  
using System.IdentityModel.Tokens.Jwt;  
using System.Security.Claims;
```



2. Na mesma classe **UsuariosController.cs**, selecione as linhas do método POST conforme o seguinte e utilize o atalho de teclado **Ctrl + K + C** para comentá-las.



3. Logo abaixo das linhas que você acabou de comentar, insira o novo código para o método POST.

```
// Novo código POST para auxiliar o método de Login.
public IActionResult Post(Usuario usuario)
{
    Usuario usuarioBuscado = _usuarioRepository.Login(usuario.Email,
usuario.Senha);
    if (usuarioBuscado == null)
    {
        return NotFound("E-mail ou senha inválidos!");
    }

    // Se o usuário for encontrado, segue a criação do token.

    // Define os dados que serão fornecidos no token - Payload.
    var claims = new[]
    {
        // Armazena na claim o e-mail usuário autenticado.
        new Claim(JwtRegisteredClaimNames.Email, usuarioBuscado.Email),

        // Armazena na claim o id do usuário autenticado.
        new Claim(JwtRegisteredClaimNames.Jti,
usuarioBuscado.Id.ToString()),
    };

    // Define a chave de acesso ao token.
    var key = new
SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("exoapi-chave-
autenticacao"));

    // Define as credenciais do token.
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    // Gera o token.
    var token = new JwtSecurityToken(
        issuer: "exoapi.webapi", // Emissor do token.
        audience: "exoapi.webapi", // Destinatário do token.
        claims: claims, // Dados definidos acima.
        expires: DateTime.Now.AddMinutes(30), // Tempo de expiração.
        signingCredentials: creds // Credenciais do token.
    );

    // Retorna ok com o token.
    return Ok(
        new { token = new JwtSecurityTokenHandler().WriteToken(token) }
    );
}

// Fim do novo código POST para auxiliar o método de Login.
```


Inserindo o Authorize (autorização)

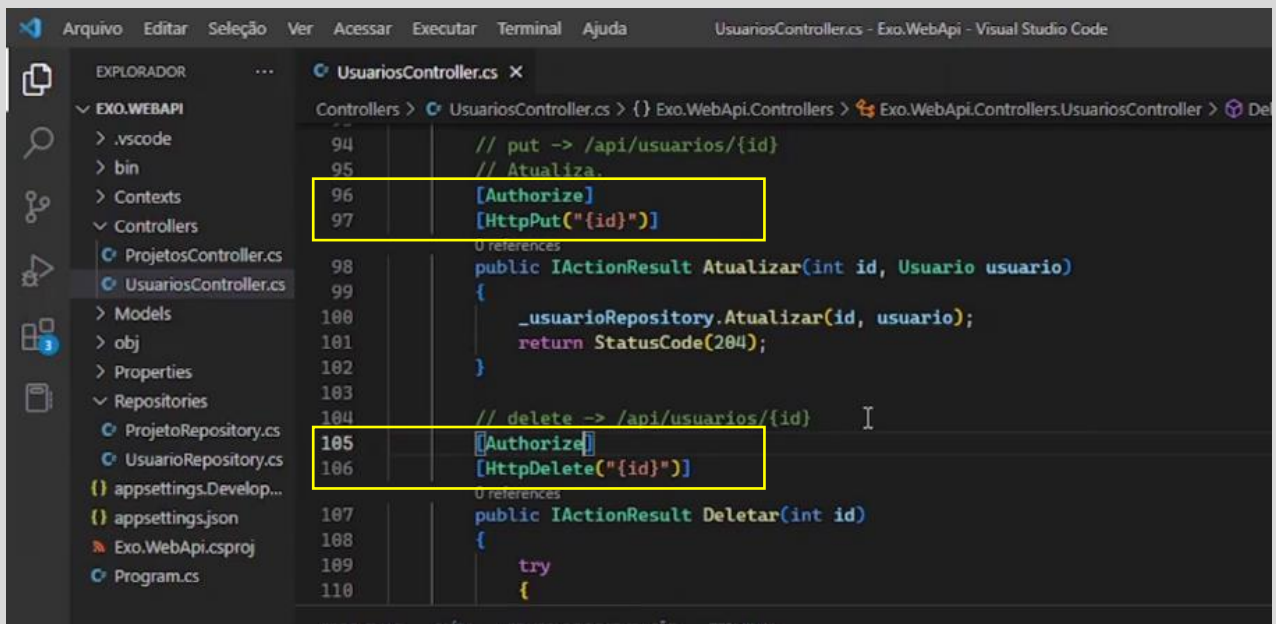
1. Em **UsuariosController.cs**, localize o comentário **// Atualiza** que está próximo da linha 95 e insira o seguinte código logo abaixo dessa linha:

```
[Authorize]
[HttpPut("{id}")]
```

2. Da mesma forma, localize o comentário **// delete -> /api/usuarios/{id}** na linha 105, aproximadamente, e insira o seguinte código logo abaixo dessa linha:

```
[Authorize]
[HttpDelete("{id}")]
```

As linhas citadas anteriormente foram adicionadas no **UsuarioController.cs** conforme a imagem abaixo.



Importante

Para substituir a sua classe **UsuariosController.cs** por completo, em vez de fazer as inserções individuais, disponibilizamos o código desse model nas [páginas finais deste PDF](#).



3. Na classe **Program.cs**, substitua todo o código existente pelo código abaixo.

```
using Microsoft.IdentityModel.Tokens;
```

4. No início desse código, localize **builder.Services.AddControllers();** na linha 8, e logo depois dela, insira o código a seguir:

```
// Forma de autenticação.
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = "JwtBearer";
    options.DefaultChallengeScheme = "JwtBearer";
})
// Parâmetros de validação do token.
.AddJwtBearer("JwtBearer", options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        // Valida quem está solicitando.
        ValidateIssuer = true,
        // Valida quem está recebendo.
        ValidateAudience = true,
        // Define se o tempo de expiração será validado.
        ValidateLifetime = true,
        // Criptografia e validação da chave de autenticação.
        IssuerSigningKey = new
            SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("exoapi-chave-
autenticacao")),
        // Valida o tempo de expiração do token.
        ClockSkew = TimeSpan.FromMinutes(30),
        // Nome do issuer, da origem.
        ValidIssuer = "exoapi.webapi",
        // Nome do audience, para o destino.
        ValidAudience = "exoapi.webapi"
    };
});
```

5. Feito isso, localize **app.UseRouting();** na linha 45, aproximadamente, e logo depois dela, insira o código a seguir.

```
// Habilita a autenticação.  
app.UseAuthentication();  
  
// Habilita a autorização.  
app.UseAuthorization();
```

Importante

Para substituir a sua classe **Program.cs** por completo, em vez de fazer as inserções individuais, disponibilizamos o código desse model nas [páginas finais deste PDF](#).



6. Abra o terminal (menu **Terminal > Novo Terminal**) e, nele, digite o comando abaixo para restaurar o projeto e consolidar a instalação:

```
dotnet restore
```

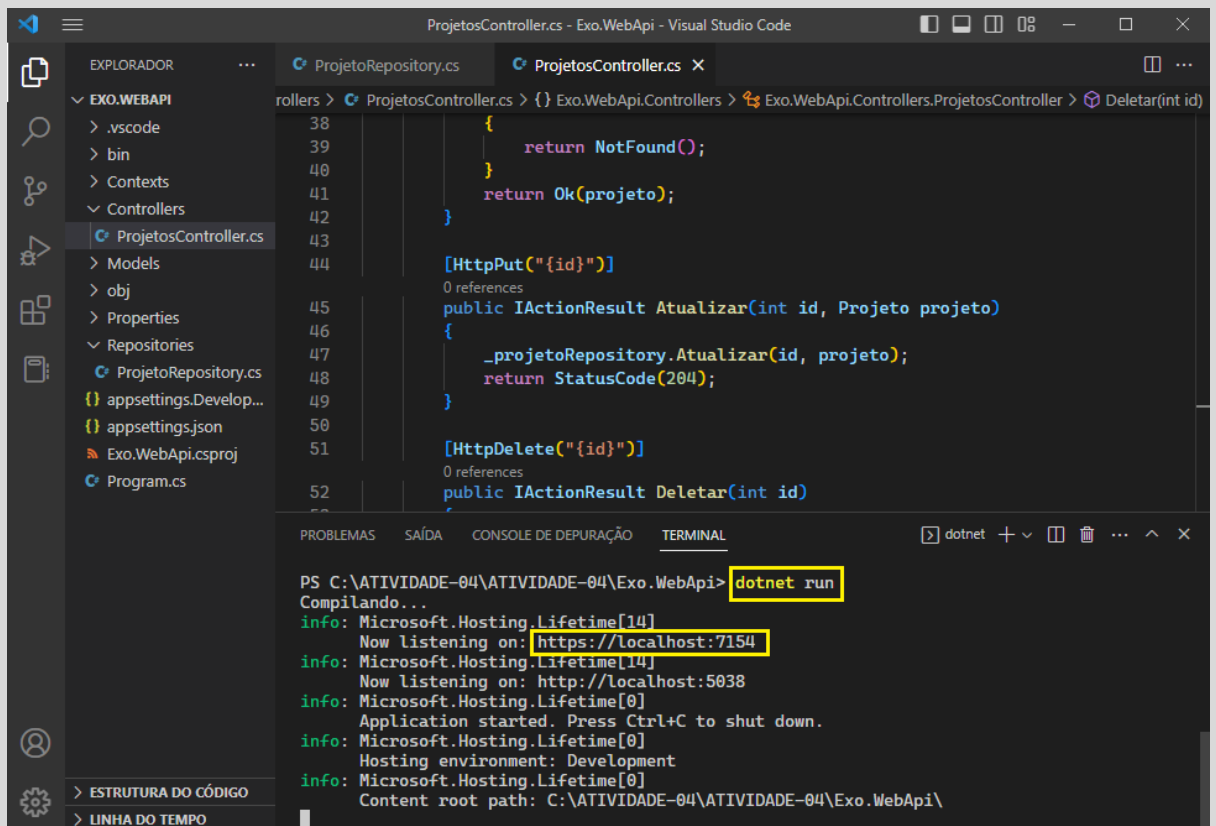
7. Digite o comando abaixo para compilar a aplicação.

```
dotnet build
```

8. Digite o próximo comando para executar o projeto.

```
dotnet run
```

9. Anote o endereço que apareceu no terminal. Utilizaremos esse link, seguido do sufixo **api/projetos**, para realizar a próxima etapa da atividade.



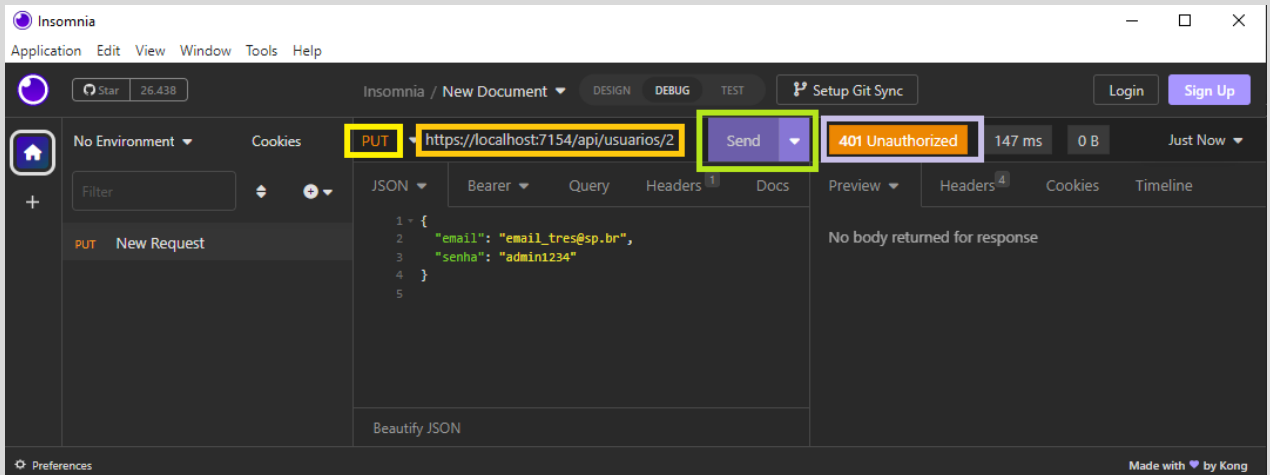
The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** The file tree on the left shows the project structure, including `EXO.WEBAPI`, `Controllers`, `Models`, `obj`, `Properties`, `Repositories`, `ProjetoRepository.cs`, `appsettings.Develop...`, `appsettings.json`, `Exo.WebApi.csproj`, and `Program.cs`.
- Code Editor:** The main editor shows the `ProjetoController.cs` file. The code includes a `Deletar(int id)` method that returns `NotFound()` or `Ok(projeto)`, an `Atualizar(int id, Projeto projeto)` method that calls `_projetoRepository.Atualizar(id, projeto)` and returns `StatusCode(204)`, and a `Deletar(int id)` method that returns `NotFound()`.
- Terminal:** The terminal at the bottom shows the output of the `dotnet run` command. The command is highlighted in yellow. The output shows the application starting on `https://localhost:7154` and `http://localhost:5038`. The terminal text is as follows:

```
PS C:\ATIVIDADE-04\ATIVIDADE-04\Exo.WebApi> dotnet run
Compilando...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7154
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5038
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\ATIVIDADE-04\ATIVIDADE-04\Exo.WebApi\
```

Utilizando o Insomnia (com token de acesso)

1. Inicie os testes pelo método atualizar (PUT). No Insomnia, escolha o método PUT, insira a url **https://localhost:7154/api/usuarios/2** para tentar modificar a senha do usuário 2 e clique em **SEND**. O resultado será o código **401 Unauthorized** na janela lateral, conforme a imagem.

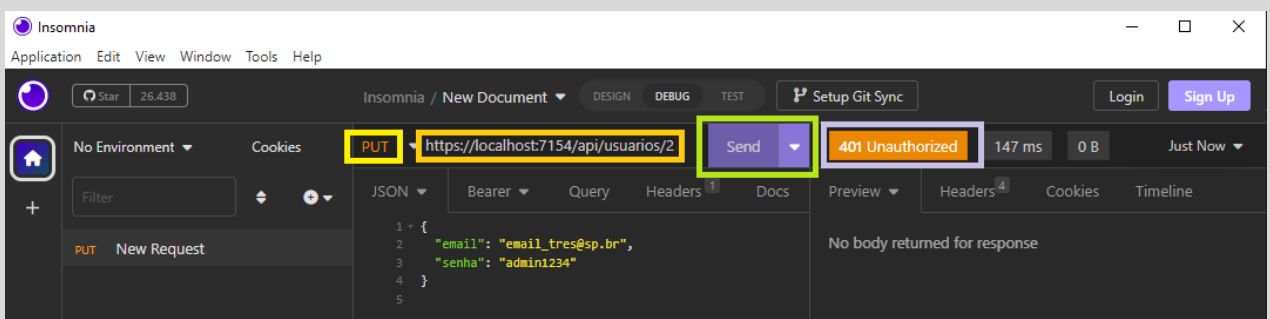


Você sabia?

401 Unauthorized é o código de resposta da requisição não autorizada. Isso acontece porque inserimos a anotação **[Authorized]** no método Atualizar da classe **UsuariosController.cs**.



2. Se tentarmos deletar o usuário, receberemos a mesma mensagem **401 Unauthorized**, pois existe também a anotação **[Authorized]** no método Deletar.



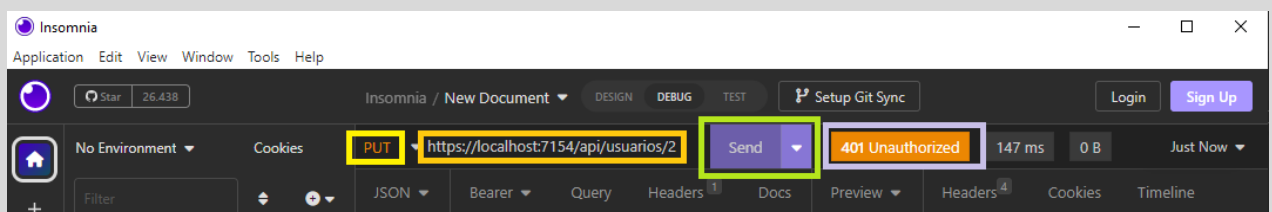
Utilizando o Insomnia (sem token de acesso)

Saiba mais...

Para acessar essas áreas que requerem autorização, vamos precisar gerar **um token de acesso**. Esse token será obtido pelo método **POST** de usuários e, por meio da autenticação Bearer, teremos acesso a essas operações restritas, como explicaremos na próxima parte deste tutorial.



1. Para criar o token, envie os dados de acesso via método **POST**. Escolha o método POST, preencha a url **https://localhost:7154/api/usuarios** e escolha a opção do corpo da requisição como **JSON**, conforme a imagem abaixo:



2. Preencha os **dados de usuário e senha** e clique em **SEND**.

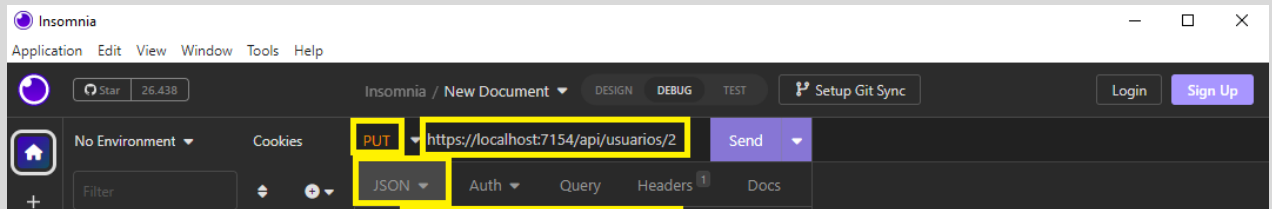
```
{
  "email": "email@sp.br",
  "senha": "1234"
}
```

Importante

O retorno da requisição será o **token** de autorização para acesso às áreas restritas. Esse token está configurado para expirar em 30 minutos, mas você pode mudar essa configuração lá na classe **UsuariosController.cs**.



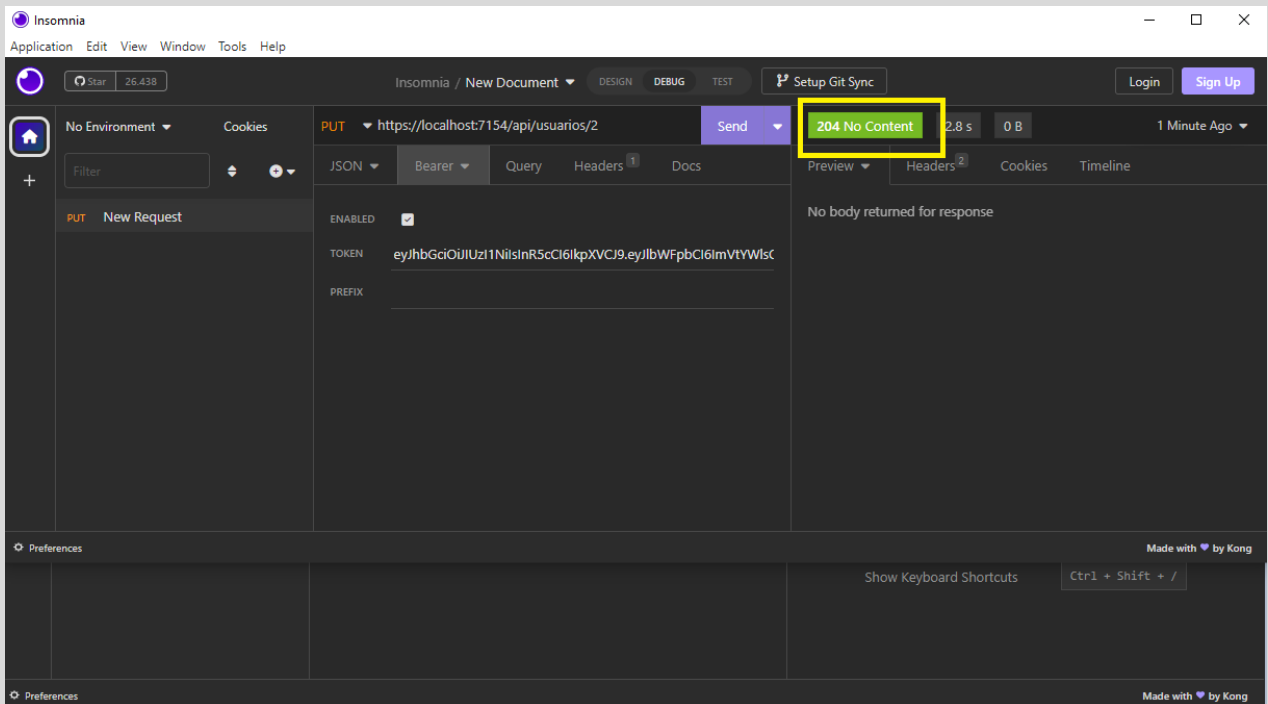
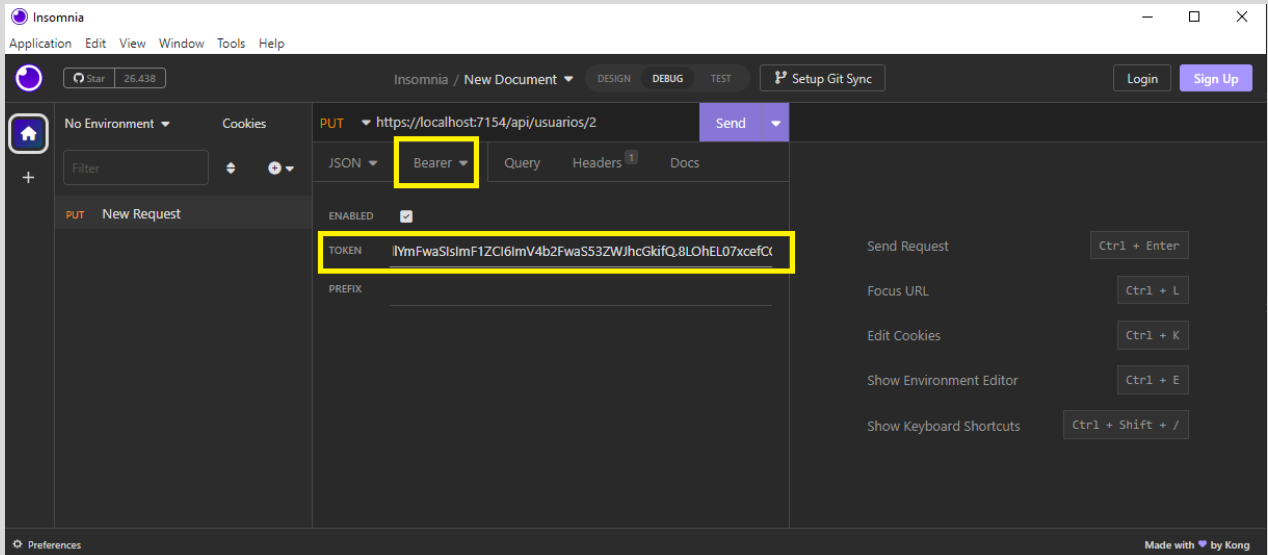
3. Copie o token (somente o código que está entre aspas). Agora vamos novamente fazer uma operação restrita, porém, utilizando o token dessa vez. Tente atualizar o usuário 2. No **Insomnia**, escolha o método PUT e preencha a url **https://localhost:7154/api/usuarios/2**.



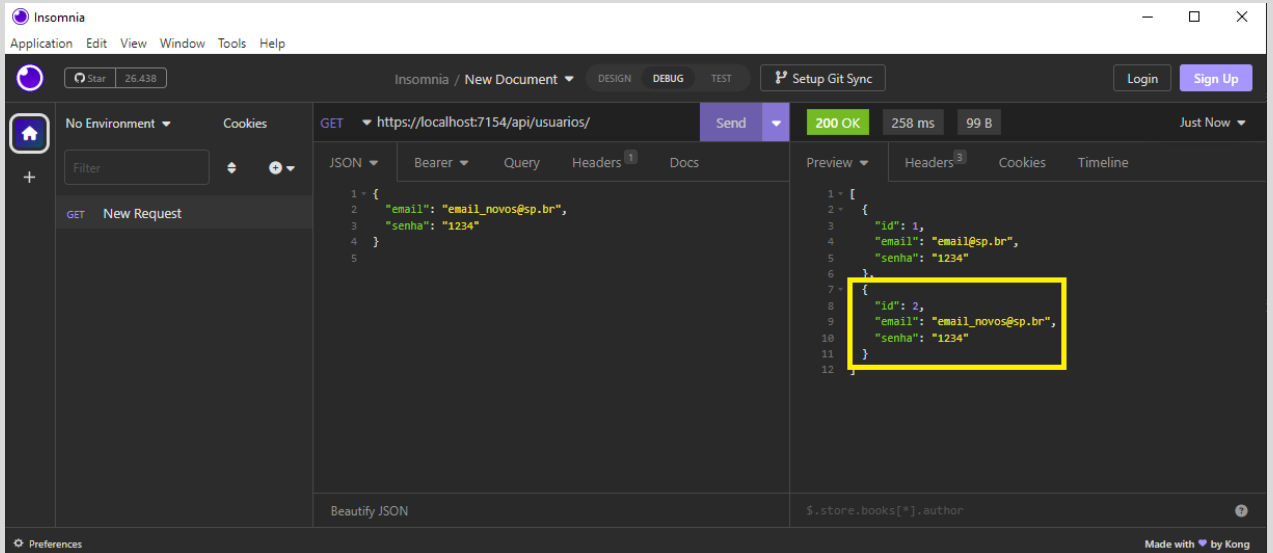
4. Preencha o corpo da requisição **JSON** com os novos dados de usuário:

```
{
  "email": "email@sp.br",
  "senha": "aaaaa1234"
}
```

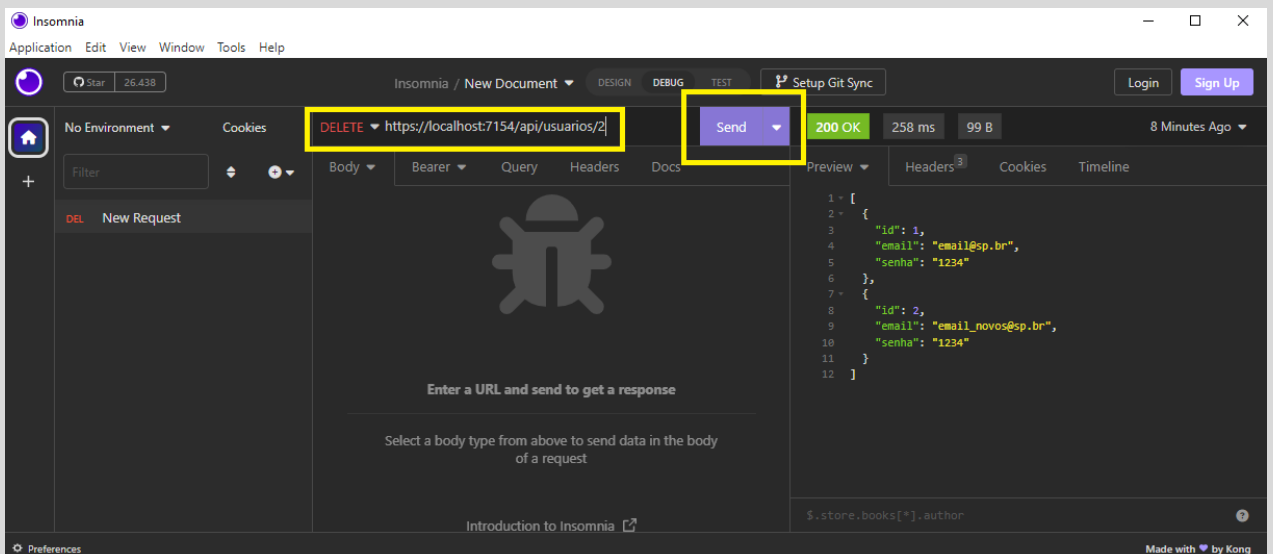
5. Clique na guia **Auth** escolha **Bearer Token** e preencha o campo token com o valor gerado na operação anterior. O resultado será **204**, código de resposta da requisição bem-sucedida.



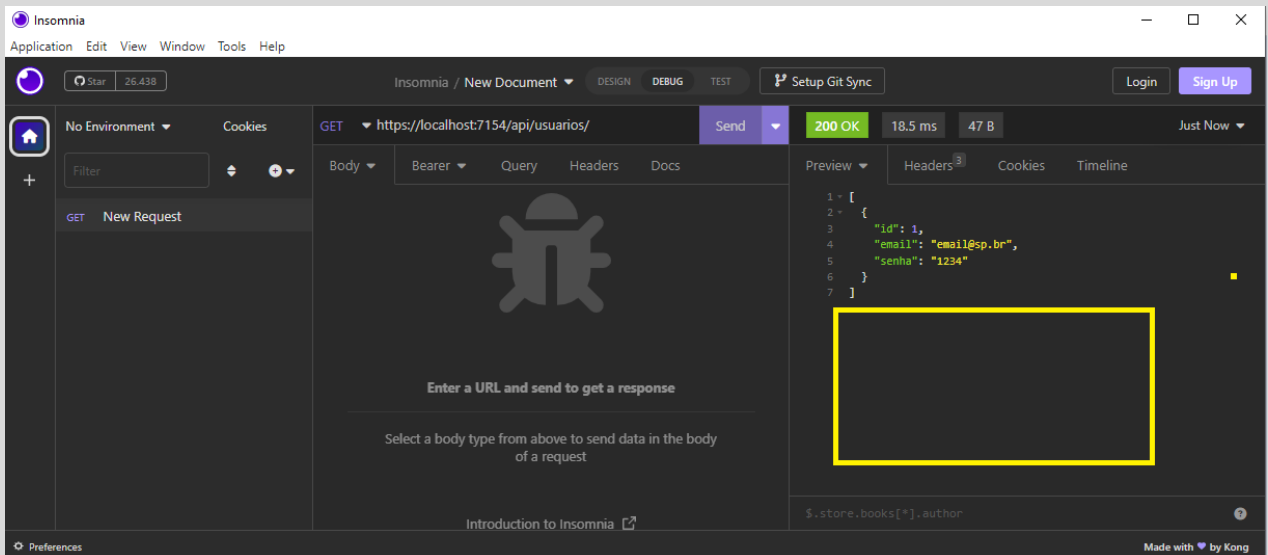
6. Se executarmos um **GET** na url **https://localhost:7154/api/usuarios/**, o resultado será o e-mail do usuário 2 alterado:



7. Por último, faremos o teste com o método Delete para ver se conseguimos deletar o usuário de id 2. Como o token já está no **Auth/Bearer Token**, basta mudar o método para **DELETE**, colocar a url **https://localhost:7154/api/usuarios/2** e clicar em **SEND**:



8. Se executarmos um **GET** na url **https://localhost:7154/api/usuarios/**, o resultado mostrará que o usuário 2 foi excluído:



Importante

Nas próximas páginas, constam os códigos completos para substituição nos arquivos citados anteriormente para [Program.cs](#) e [UsuariosController.cs](#).



Classe UsuariosController.cs

```
using Exo.WebApi.Models;
using Exo.WebApi.Repositories;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;

namespace Exo.WebApi.Controllers
{
    [Produces("application/json")]
    [Route("api/[controller]")]
    [ApiController]
    public class UsuariosController : ControllerBase
    {
        private readonly UsuarioRepository _usuarioRepository;
        public UsuariosController(UsuarioRepository
usuarioRepository)
        {
            _usuarioRepository = usuarioRepository;
        }

        // get -> /api/usuarios
        [HttpGet]
        public IActionResult Listar()
        {
            return Ok(_usuarioRepository.Listar());
        }

        // Novo código POST para auxiliar o método de Login.
        public IActionResult Post(Usuario usuario)
        {
            Usuario usuarioBuscado =
            _usuarioRepository.Login(usuario.Email, usuario.Senha);
            if (usuarioBuscado == null)
            {
                return NotFound("E-mail ou senha inválidos!");
            }
        }
    }
}
```

Continuação do código >>>

>>> continuação do código

```

        // Se o usuário for encontrado, segue a criação do token.

        // Define os dados que serão fornecidos no token - Payload.
        var claims = new[]
        {
            // Armazena na claim o e-mail usuário autenticado.
            new Claim(JwtRegisteredClaimNames.Email,
usuarioBuscado.Email),

            // Armazena na claim o id do usuário autenticado.
            new Claim(JwtRegisteredClaimNames.Jti,
usuarioBuscado.Id.ToString()),
        };

        // Define a chave de acesso ao token.
        var key = new
SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("exoapi-chave-
autenticacao"));

        // Define as credenciais do token.
        var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);

        // Gera o token.
        var token = new JwtSecurityToken(
            issuer: "exoapi.webapi", // Emissor do token.
            audience: "exoapi.webapi", // Destinatário do token.
            claims: claims, // Dados definidos acima.
            expires: DateTime.Now.AddMinutes(30), // Tempo de
expiração.
            signingCredentials: creds // Credenciais do token.
        );

        // Retorna ok com o token.
        return Ok(
            new { token = new
JwtSecurityTokenHandler().WriteToken(token) }
        );
    }
    // Fim do novo código POST para auxiliar o método de Login.

```

Continuação do código >>>

>>> continuação do código

```
// get -> /api/usuarios/{id}
[HttpGet("{id}")] // Faz a busca pelo ID.
public IActionResult BuscarPorId(int id)
{
    Usuario usuario = _usuarioRepository.BuscarPorId(id);
    if (usuario == null)
    {
        return NotFound();
    }
    return Ok(usuario);
}

// put -> /api/usuarios/{id}
// Atualiza.
[Authorize]
[HttpPut("{id}")]
public IActionResult Atualizar(int id, Usuario usuario)
{
    _usuarioRepository.Atualizar(id, usuario);
    return StatusCode(204);
}

// delete -> /api/usuarios/{id}
[Authorize]
[HttpDelete("{id}")]
public IActionResult Deletar(int id)
{
    try
    {
        _usuarioRepository.Deletar(id);
        return StatusCode(204);
    }
    catch (Exception e)
    {
        return BadRequest();
    }
}

}
```

Classe Program.cs

```
using Exo.WebApi.Contexts;
using Exo.WebApi.Repositories;
using Microsoft.IdentityModel.Tokens;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddScoped<ExoContext, ExoContext>();
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Forma de autenticação.
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = "JwtBearer";
    options.DefaultChallengeScheme = "JwtBearer";
})
// Parâmetros de validação do token.
.AddJwtBearer("JwtBearer", options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        // Valida quem está solicitando.
        ValidateIssuer = true,
        // Valida quem está recebendo.
        ValidateAudience = true,
        // Define se o tempo de expiração será validado.
        ValidateLifetime = true,
        // Criptografia e validação da chave de autenticação.
        IssuerSigningKey = new
        SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("exoapi-
chave-autenticacao")),
    }
});
```

Continuação do código >>>

>>> continuação do código

```
        // Valida o tempo de expiração do token.
        ClockSkew = TimeSpan.FromMinutes(30),
        // Nome do issuer, da origem.
        ValidIssuer = "exoapi.webapi",
        // Nome do audience, para o destino.
        ValidAudience = "exoapi.webapi"
    };
});

builder.Services.AddTransient<ProjetoRepository,
ProjetoRepository>();
builder.Services.AddTransient<UsuarioRepository,
UsuarioRepository>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseRouting();

// Habilita a autenticação.
app.UseAuthentication();

// Habilita a autorização.
app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});

app.Run();
```