

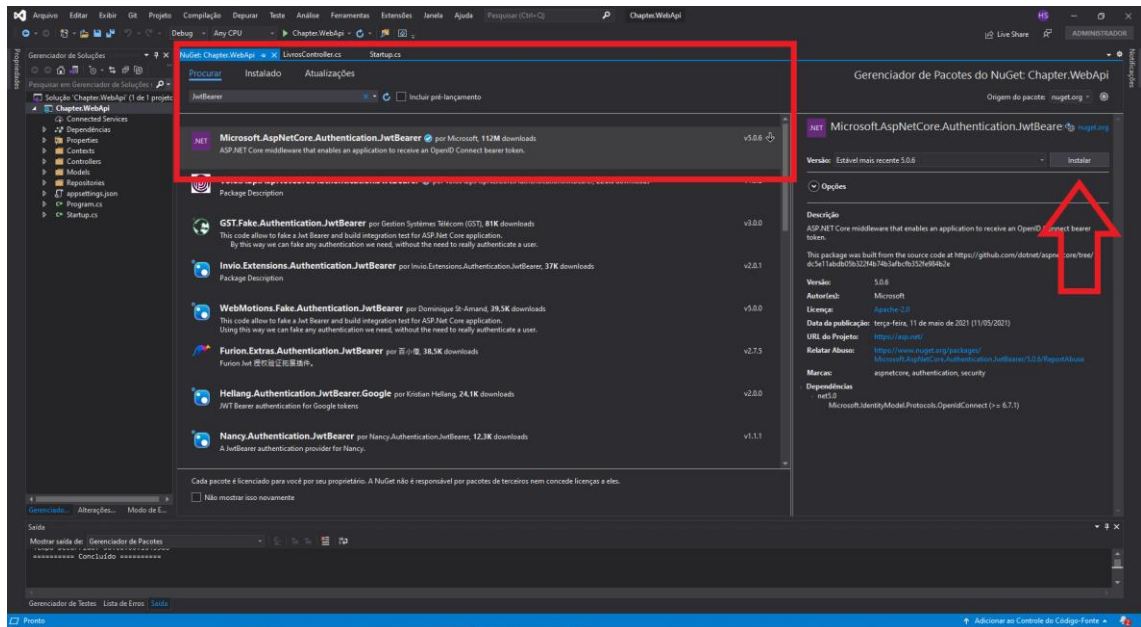
Restringindo o acesso e usando token

Para atingir o objetivo de restringir o acesso ao DELETE, faremos as etapas:

- 1) Adicionar o pacote do JwtBearer para habilitar o *middleware* (*software* que fornece recursos comuns a aplicações, como autenticação e gerenciamento de APIs);
- 2) Adicionar no Startup.cs as configurações necessárias para habilitar a autenticação;
- 3) Adicionar a restrição no *endpoint* de `/api/livros/{id}` de deleção para usuários autenticados;
- 4) Criar um repositório de usuário para validar se o e-mail e senha informados existem no banco de dados;
- 5) Criar um *endpoint* de login que recebe e-mail e senha para gerar o token;
- 6) Enviar o token da requisição do DELETE `/api/livros/{id}`.

Adicionando o pacote JwtBearer

Sobre o projeto, clique com o botão direito → Gerenciar pacotes do *nuget* e pesquise pelo pacote de referência: *JwtBearer*. Clique em instalar.



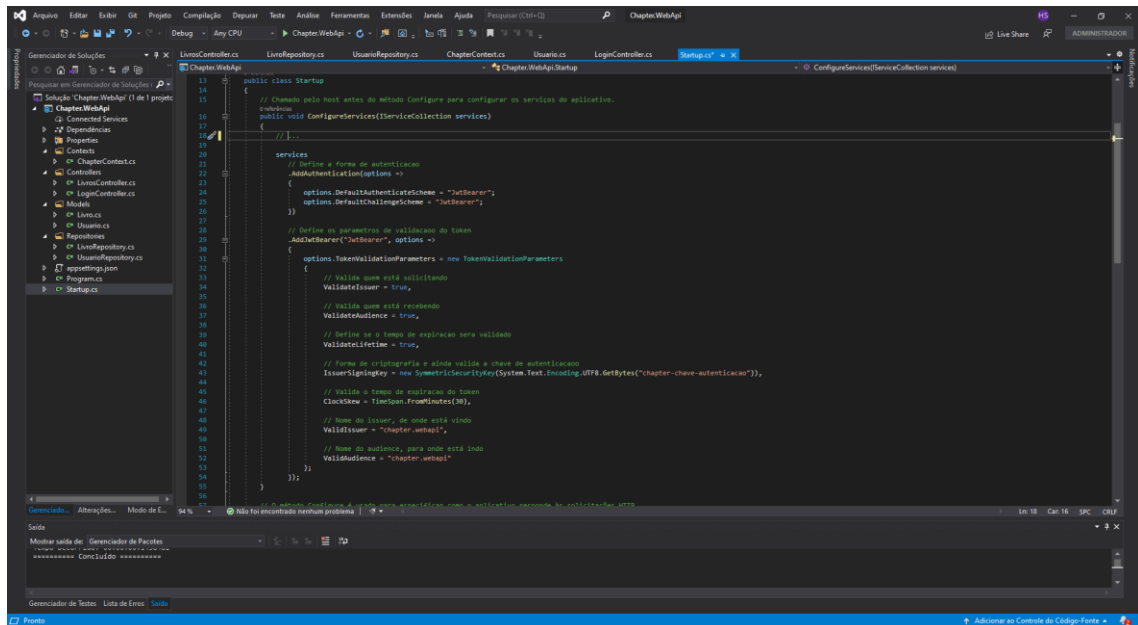
Saiba mais

A API do servidor NuGet é um conjunto de pontos de extremidade HTTP que podem ser usados para baixar pacotes, buscar metadados, publicar novos pacotes e executar a maioria das outras operações disponíveis nos clientes NuGet oficiais.



Adicionando a configuração no Startup.cs

Adicione a configuração no projeto Startup.cs.



```
13 public class Startup
14 {
15     // Chamado pelo host antes do método Configure para configurar os serviços do aplicativo.
16     // Dependências
17     public void ConfigureServices(IServiceCollection services)
18     {
19         // ...
20
21         services
22             .AddAuthentication(options =>
23             {
24                 options.DefaultAuthenticateScheme = "JwtBearer";
25                 options.DefaultChallengeScheme = "JwtBearer";
26             })
27             .AddJwtBearer("JwtBearer", options =>
28             {
29                 // Define os parâmetros de validação do token
30                 options.TokenValidationParameters = new TokenValidationParameters
31                 {
32                     // Valida quem está solicitando
33                     ValidateIssuer = true,
34
35                     // Valida quem está recebendo
36                     ValidateAudience = true,
37
38                     // Define se o tempo de expiração será validado
39                     ValidateLifetime = true,
40
41                     // Define se o algoritmo e chave serão validados
42                     IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("chapter-chave-autenticacao")),
43
44                     // Valida o tempo de expiração do token
45                     ClockSkew = TimeSpan.FromMinutes(10),
46
47                     // Nome do issuer, de onde está vindo
48                     ValidateIssuer = "chapter.webapi",
49
50                     // Nome da audiência, para onde está indo
51                     ValidateAudience = "chapter.webapi"
52                 };
53             });
54     }
55 }
```

```
services
    // Define a forma de autenticação
    .AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = "JwtBearer";
        options.DefaultChallengeScheme = "JwtBearer";
    })

    // Define os parametros de validacao 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,

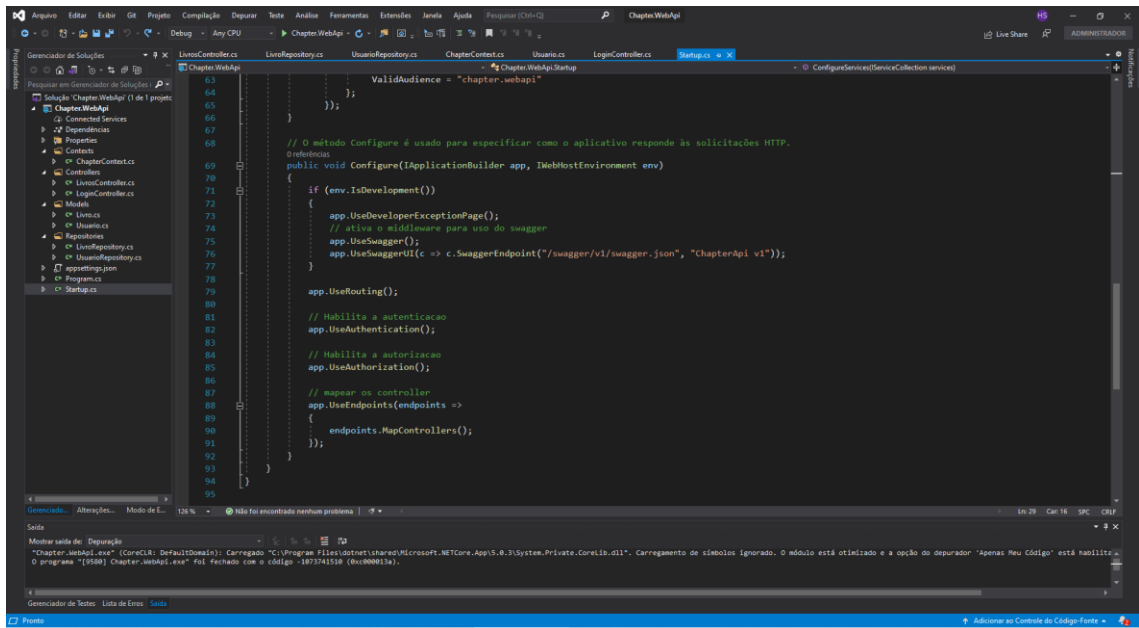
            // criptografia e validação da chave de autenticação
            IssuerSigningKey = new
SymmetricSecurityKey(System.Text.Encoding.UTF8.GetByte
s("chapter-chave-autenticação")),

            // Valida o tempo de expiração do token
            ClockSkew = TimeSpan.FromMinutes(30),

            // Nome do issuer, de onde está vindo
            ValidIssuer = "chapter.webapi",

            // Nome do audience, para onde está indo
            ValidAudience = "chapter.webapi"
        };
    });
```

Banco de dados

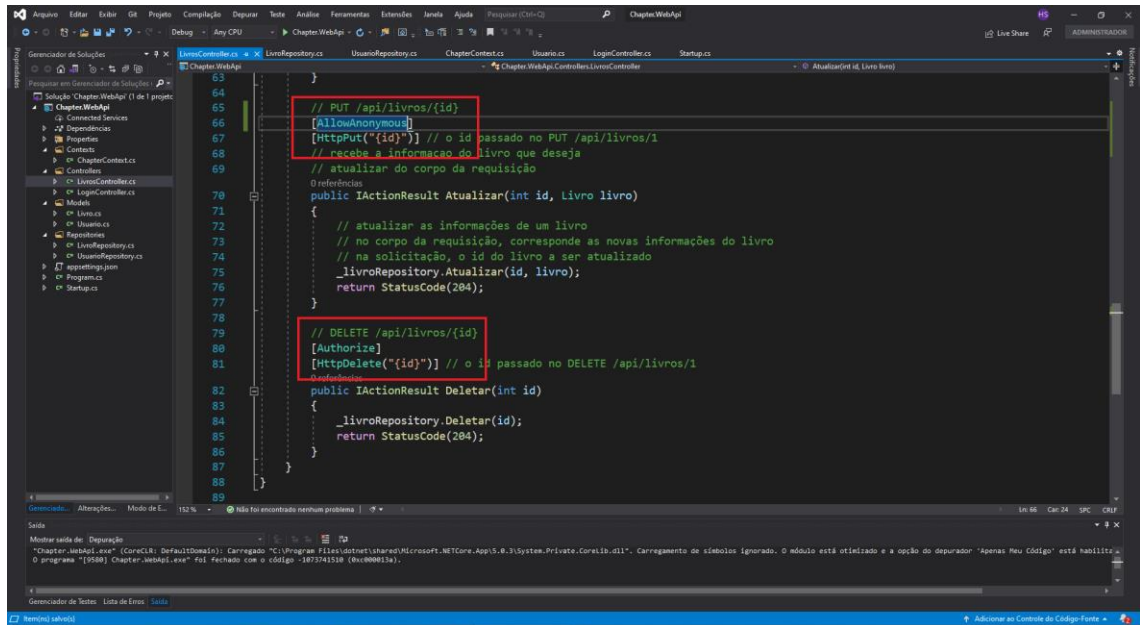


Na sessão de Configure, adicione o código para habilitar a autenticação e autorização do nosso projeto:

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

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

No controller de livros, adicionaremos a restrição no *endpoint* para deletar. Através do atributo [Authorize], somente usuários autenticados (com o token) poderão realizar requisição. Enquanto o atributo [AllowAnonymous] permite que qualquer usuário solicite as informações.



```
63 }
64
65 // PUT /api/livros/{id}
66 [AllowAnonymous]
67 [HttpPut("{id}")] // o id passado no PUT /api/livros/1
68 // recebe a informacao do livro que deseja
69 // atualizar do corpo da requisicao
70 public IActionResult Atualizar(int id, Livro livro)
71 {
72     // atualizar as informacoes de um livro
73     // no corpo da requisicao, corresponde as novas informacoes do livro
74     // na solicitacao, o id do livro a ser atualizado
75     _livroRepository.Atualizar(id, livro);
76     return StatusCode(204);
77 }
78
79 // DELETE /api/livros/{id}
80 [Authorize]
81 [HttpDelete("{id}")] // o id passado no DELETE /api/livros/1
82 // deleta o livro
83 public IActionResult Deletar(int id)
84 {
85     _livroRepository.Deletar(id);
86     return StatusCode(204);
87 }
88
89 }
```

```
// PUT /api/livros/{id}
[AllowAnonymous]
[HttpPut("{id}")] // o id passado no PUT /api/livros/1
// recebe a informacao do livro que deseja
// atualizar do corpo da requisição
public IActionResult Atualizar(int id, Livro livro)
{
    // atualizar as informações de um livro
    // no corpo da requisição, corresponde as novas
informações do livro
    // na solicitação, o id do livro a ser atualizado
    _livroRepository.Atualizar(id, livro);
    return StatusCode(204);
}

// DELETE /api/livros/{id}
[Authorize]
[HttpDelete("{id}")] // o id passado no DELETE /api/livros/1
public IActionResult Deletar(int id)
{
    try
    {
        _livroRepository.Deletar(id);
        return StatusCode(204);
    } catch (Exception e)
    {
        return BadRequest();
    }
}
```

Banco de dados

Rode a aplicação, abra o Insomnia e tente realizar a requisição para a URL a DELETE /api/livros/{id}. Você receberá o status 401, não autorizado.

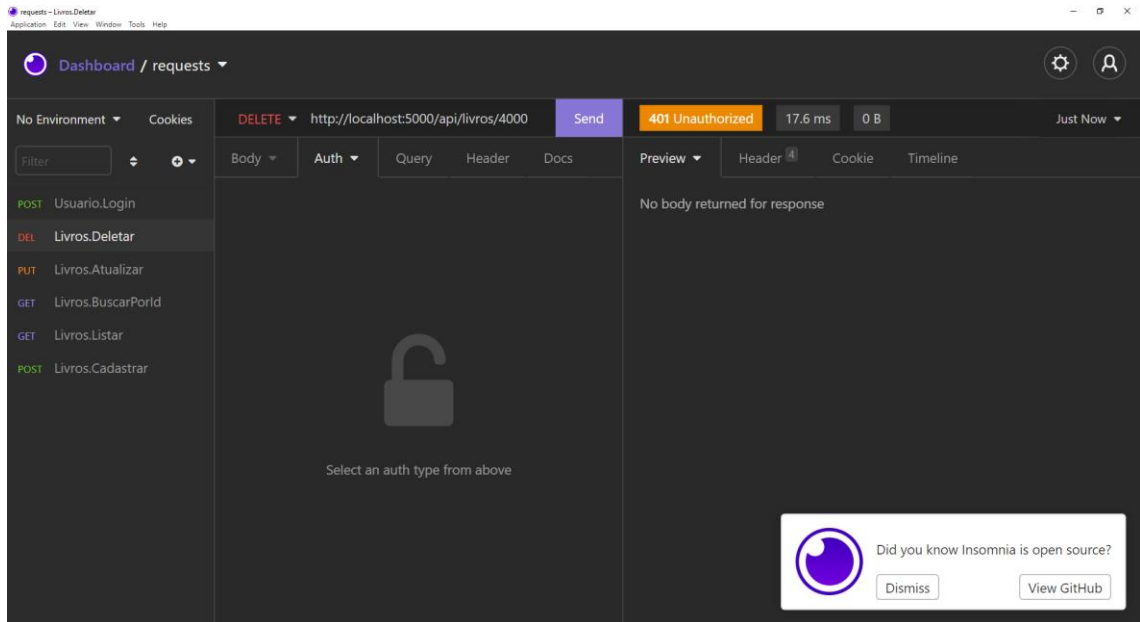
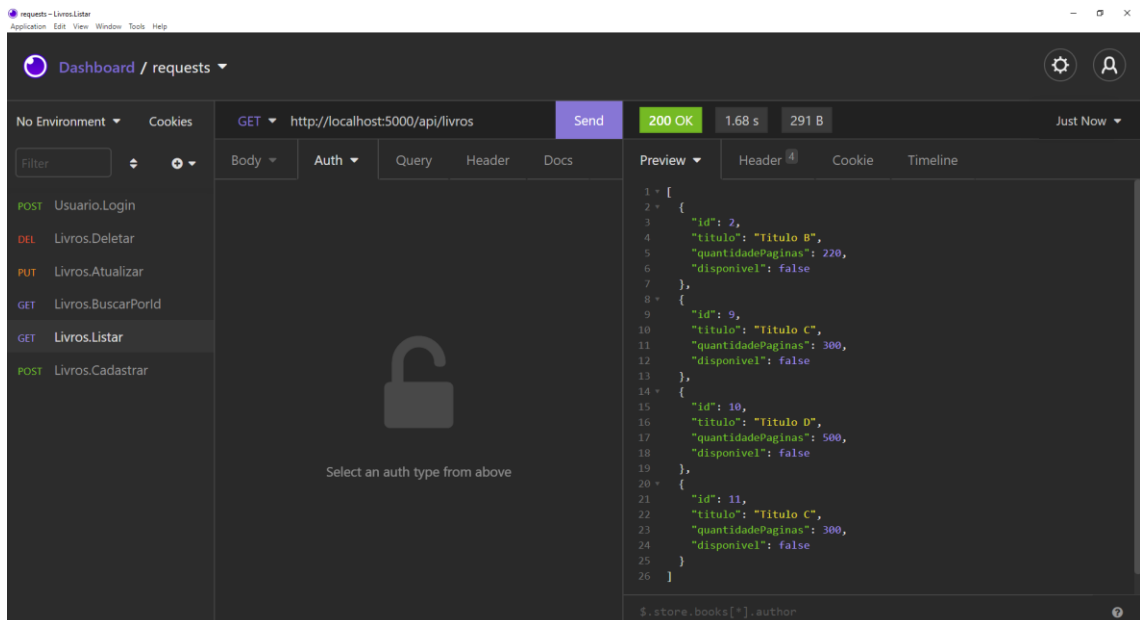


Imagem representando uma requisição a lista.



O acesso está restrito, mas o token ainda não foi gerado na aplicação, não sendo possível ter a chave para enviar nessa requisição e ter acesso ao recurso mencionado.

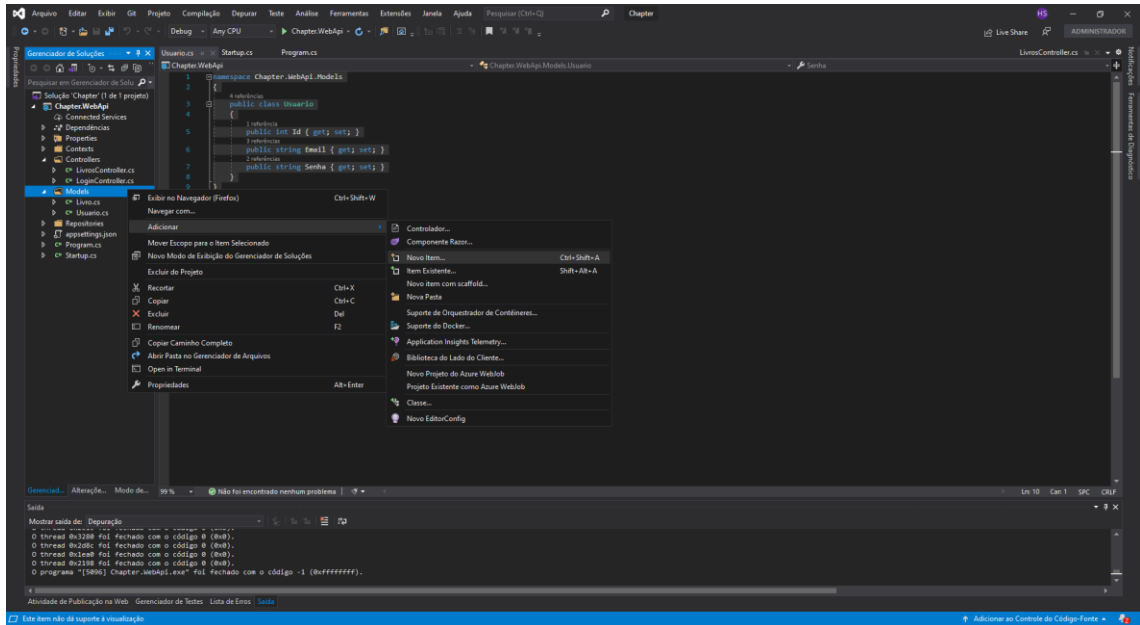
Para a criação do token, são duas etapas:

- criar o modelo de usuário e repositório que dado um e-mail e senha,
- verificar no banco de dados se essas informações existem.

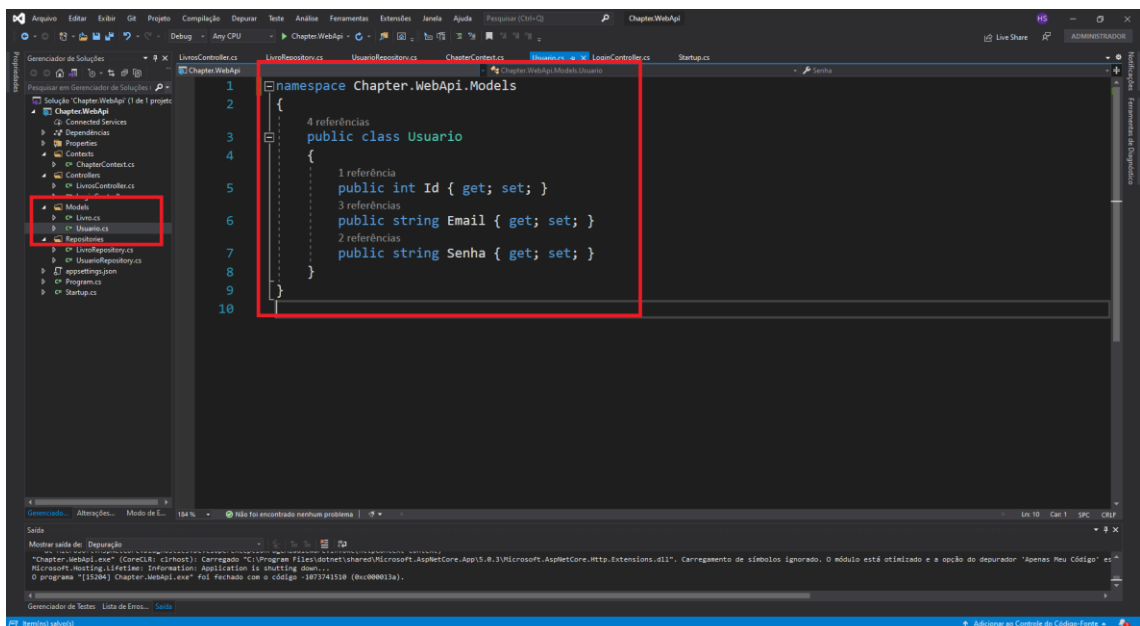
Ao criar um endpoint `/api/login` para receber as informações na requisição de e-mail e senha, verifique se o usuário existe e criar o token de acesso.

Criando o modelo de Usuario

Clique com o botão direito sob a pasta Models, e adicione um novo Item.



Coloque o nome do arquivo como Usuario.cs.



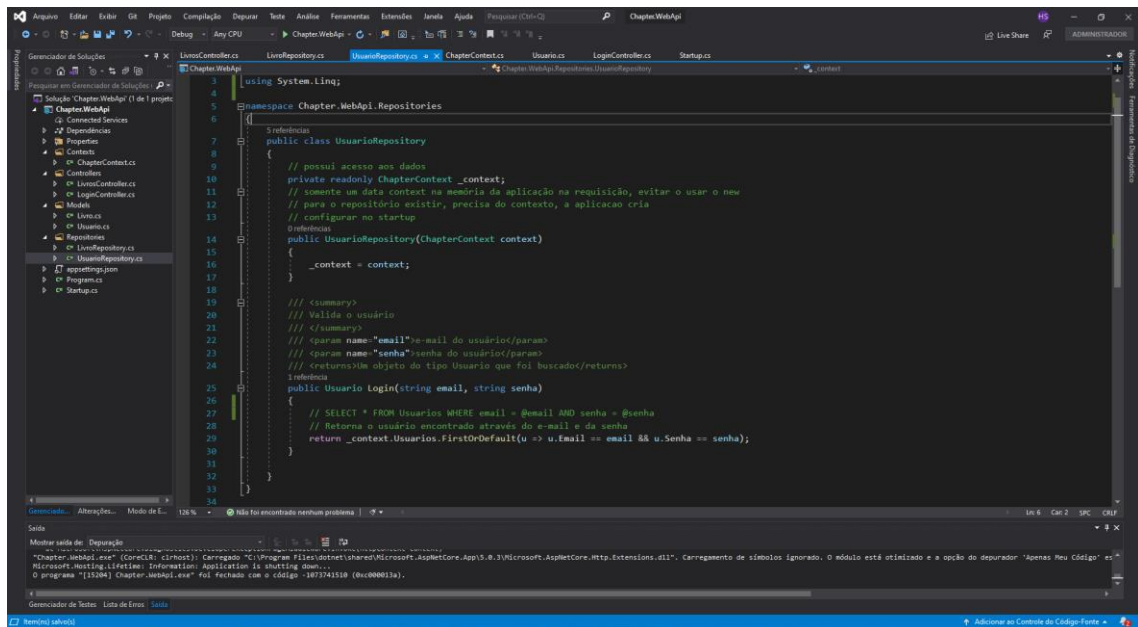
Código da classe do Usuario.cs.

```
namespace Chapter.WebApi.Models
{
    public class Usuario
    {
        public int Id { get; set; }
        public string Email { get; set; }
        public string Senha { get; set; }
    }
}
```

Criando o repositório de Usuario

Adicione um arquivo `UsuarioRepository.cs` que fará a consulta dos dados do usuário através do e-mail e senha para verificar se as informações conferem com as informações adicionadas no banco de dados.

Imagem representando a busca de usuário por e-mail e senha.



Código do UsuarioRepository.cs.

```

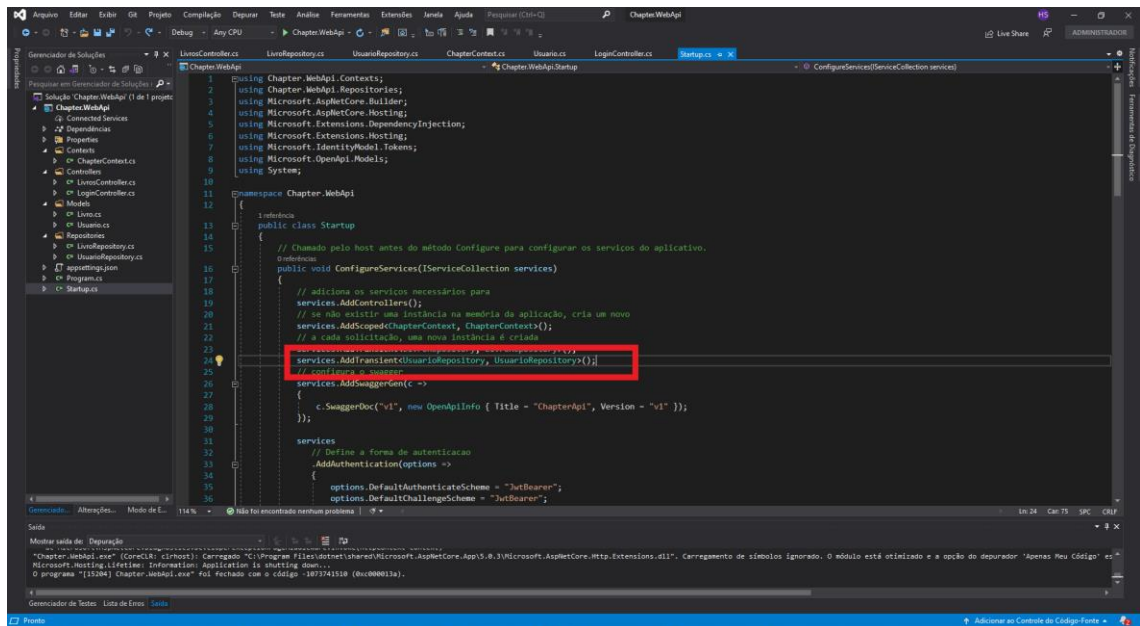
using Chapter.WebApi.Contexts;
using Chapter.WebApi.Models;
using System.Linq;

namespace Chapter.WebApi.Repositories
{
    public class UsuarioRepository
    {
        // possui acesso aos dados
        private readonly ChapterContext _context;
        // somente um data context na memória da aplicação na
        // requisição, evitar o usar o new
        // para o repositório existir, precisa do contexto, a
        // aplicacao cria
        // configurar no startup
        public UsuarioRepository(ChapterContext context)
        {
            _context = context;
        }

        /// <summary>
        /// Valida o usuário
        /// </summary>
        /// <param name="email">e-mail do usuário</param>
        /// <param name="senha">senha do usuário</param>
        /// <returns>Um objeto do tipo Usuario que foi
        buscado</returns>
        public Usuario Login(string email, string senha)
        {
            // SELECT * FROM Usuarios WHERE email = @email AND senha
            // Retorna o usuário encontrado através do e-mail e da
            // senha
            return _context.Usuarios.FirstOrDefault(u => u.Email ==
            email && u.Senha == senha);
        }
    }
}

```

Adicionar no Startup.cs a configuração do repositório.



```
services.AddTransient<UsuarioRepository, UsuarioRepository>();
```

Criando o controller de login

Para realizar a criação do token, precisaremos de um endpoint que receba o e-mail e senha, verifique se esses dados existem na base e caso exista, crie o token.

Método	Endpoint	Descrição	Corpo da solicitação	Corpo da resposta
GET	/api/livros	Obter todos os livros	Nenhum	Lista de livros
GET	/api/livros/{id}	Obter um livro por um identificador	Nenhum	Um livro
POST	/api/livros	Adicionar um livro	Dados do livro	Nenhum
PUT	/api/livros/{id}	Atualizar os dados de um livro existente	Dados do livro	Nenhum
DELETE	/api/livros/{id}	Excluir um livro	Nenhum	Nenhum
POST	/api/login	Criar um token	Email e senha	token

Crie o arquivo LoginController na pasta de Controllers, que ficam responsáveis pelas requisições em nossa API Web.

```

namespace Chapter.WebApi.Controllers
{
    /// <summary>
    /// Controller responsável pelos endpoints referentes ao login
    /// </summary>
    [Produces("application/json")]
    [Route("api/NameController/login")]
    [ApiController]
    public class LoginController : ControllerBase
    {
        private readonly IRepository _userRepository;

        public LoginController(IRepository userRepository)
        {
            _userRepository = userRepository;
        }

        /// <summary>
        /// Valida o usuário
        /// </summary>
        /// <param name="login">Objeto login que contém o e-mail e a senha do usuário/parâmetro</param>
        /// <returns>Retorna um token com as informações do usuário/retorno</returns>
        [HttpPost]
        public IActionResult Post([FromBody] Login login)
        {
            // Busca o usuário pelo e-mail e senha
            var usuarioEncontrado = _userRepository.Login(login.Email, login.Senha);
            // Caso não encontre nenhum usuário com o e-mail e senha informados
            if (usuarioEncontrado == null)
            {
                // Retorna NotFound com uma mensagem de erro
                return NotFound("E-mail ou senha inválidos!");
            }
        }
    }
}

```

```

return NotFound("E-mail ou senha inválidos!");
}

// Caso o usuário seja encontrado, prossegue para a criação do token
// Dependências
// Criar a validação de JWT: System.IdentityModel.Tokens.Jwt
// Integrar a autenticação: Microsoft.AspNetCore.Authentication.JwtBearer (versão compatível com o .NET do projeto)
// Define os dados que serão fornecidos no token - Payload
var claims = new[]
{
    // Adiciona na Claim o e-mail do usuário autenticado
    new Claim(JwtRegisteredClaimNames.Email, usuarioEncontrado.Email),
    // Adiciona na Claim o ID do usuário autenticado
    new Claim(JwtRegisteredClaimNames.Sub, usuarioEncontrado.Id.ToString()),
};

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

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

// Gera o token
var token = new JwtSecurityToken(
    issuer: "chapter-webapi", // Emissor do token
    audience: "chapter-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)
});
}
}

```


Código completo da classe LoginController.cs.

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

namespace Chapter.WebApi.Controllers
{
    /// <summary>
    /// Controller responsável pelos endpoints referentes ao Login
    /// </summary>

    // Define que o tipo de resposta da API será no formato JSON
    [Produces("application/json")]

    // Define que a rota de uma requisição será no formato
    domínio/api/NomeController
    // http://localhost:5000/api/login
    [Route("api/[controller]")]
    // Define que é um controlador de API
    [ApiController]
    public class LoginController : ControllerBase
    {
        private readonly UsuarioRepository _usuarioRepository;
        public LoginController(UsuarioRepository usuarioRepository)
        {
            _usuarioRepository = usuarioRepository;
        }
        /// <summary>
        /// Valida o usuário
        /// </summary>
        /// <param name="login">Objeto login que contém o e-mail e a
        senha do usuário</param>
        /// <returns>Retorna um token com as informações do
        usuário</returns>
    }
}
```

```

// dominio/api/Login
[HttpPost]
public IActionResult Post(Usuario login)      {
    // Busca o usuário pelo e-mail e senha
    Usuario usuarioBuscado =
_usuarioRepository.Login(login.Email, login.Senha);
    // Caso não encontre nenhum usuário com o e-mail e senha
informados
    if (usuarioBuscado == null)
    {
        // Retorna NotFound com uma mensagem de erro
        return NotFound("E-mail ou senha inválidos!");
    }
    // Caso o usuário seja encontrado, prossegue para a
criação do token
    /*
        Dependências
        Criar e validar o JWT:
System.IdentityModel.Tokens.Jwt
        Integrar a autenticação:
Microsoft.AspNetCore.Authentication.JwtBearer (versão compatível com
o .NET do projeto)
    */
    // Define os dados que serão fornecidos no token - Payload
    var claims = new[]
    {
        // Armazena na Claim o e-mail do 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("chapter-
chave-autenticacao"));

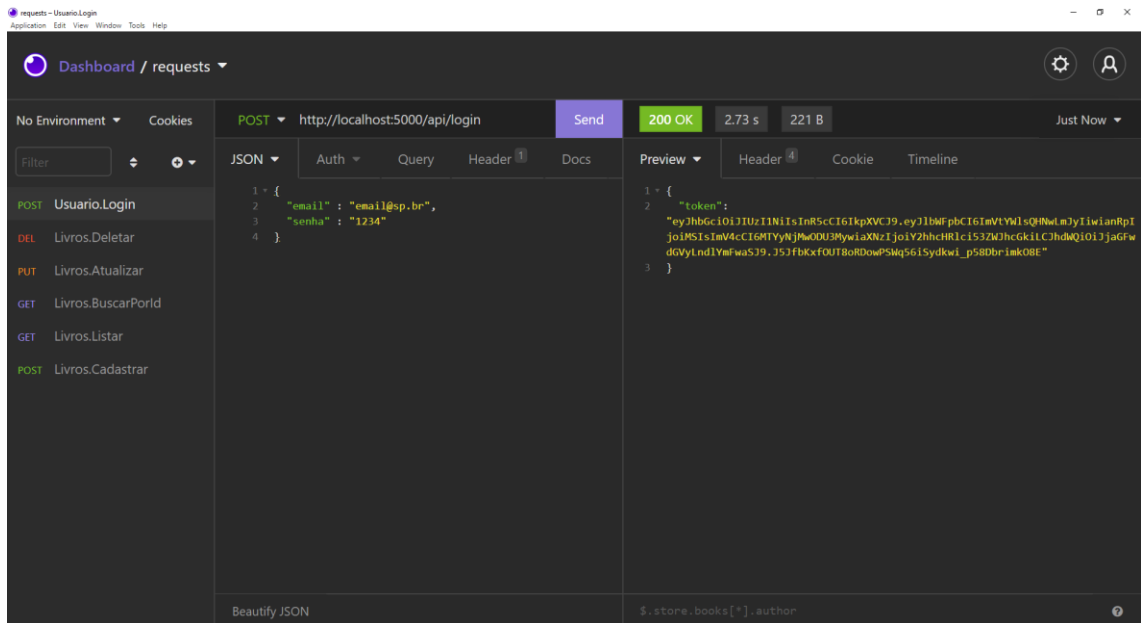
        // Define as credenciais do token
        var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);
        // Gera o token
        var token = new JwtSecurityToken(
            issuer: "chapter.webapi",                // emissor
do token
            audience: "chapter.webapi",              //
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)
        });
    }
}

```

Criando o token

Para criar o token, abra o insomnia e adicione uma requisição em que faremos a operação de POST para enviar no corpo da solicitação o e-mail e a senha do usuário, para a url: <http://localhost:5000/api/login>



Banco de dados

Selecione o valor do token na solicitação da criação do token, e na requisição do DELETE, adicione essa informação no cabeçalho da requisição.

O cabeçalho será enviada no cabeçalho de Authorization quando solicitado um recurso protegido.

Authorization: Bearer <token>

