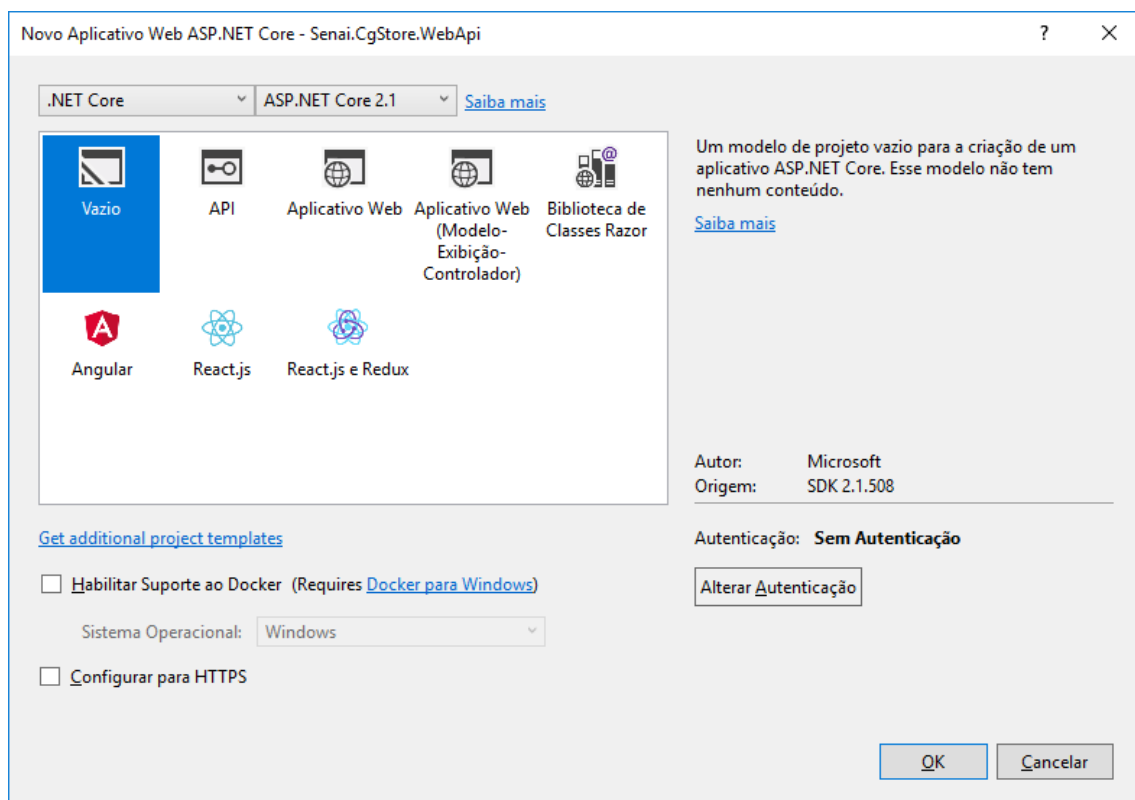
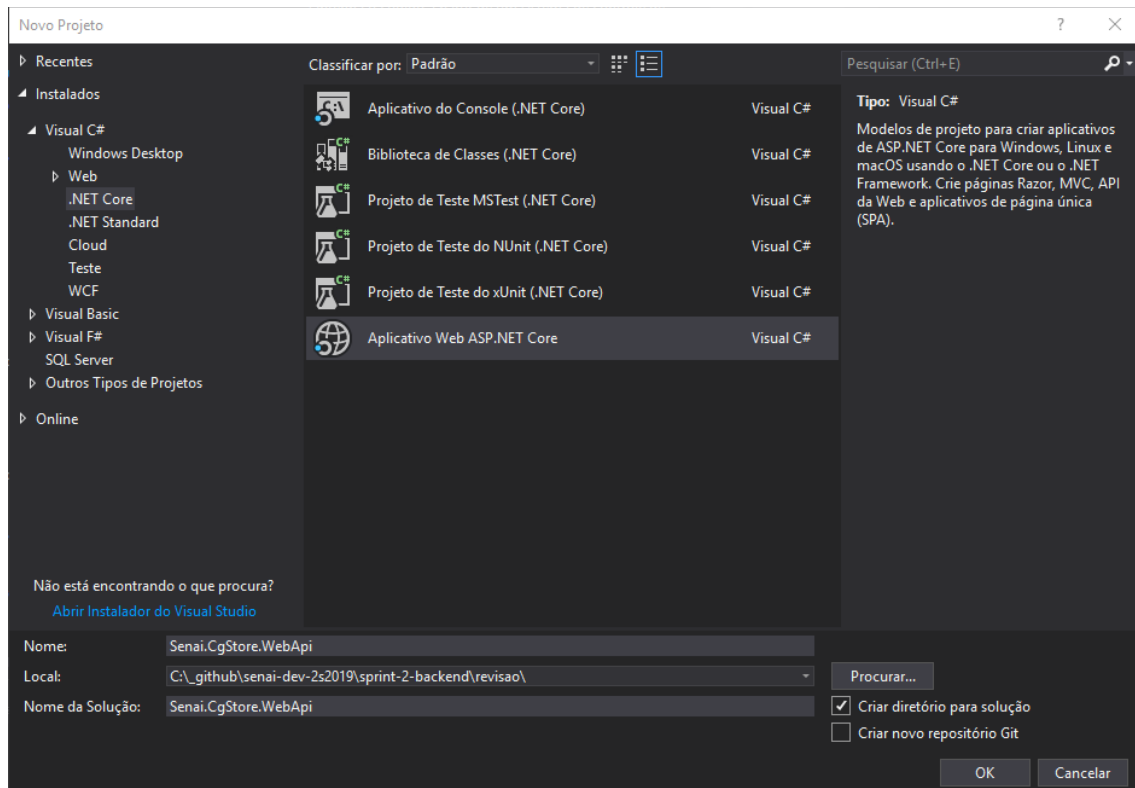


Executar o script para a criação do banco de dados e tabelas:

cgstore_revisao.sql

Criar Projeto



Gerenciar Pacotes do NuGet e instalar as 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)

Swagger

Swashbuckle.AspNetCore (4.0.1)

Configurar o startup o MVC e o Swagger.

```
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);

    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Swashbuckle.AspNetCore.Swagger.Info { Title = "CgStore API", Version = "v1" });
    });
}
```

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

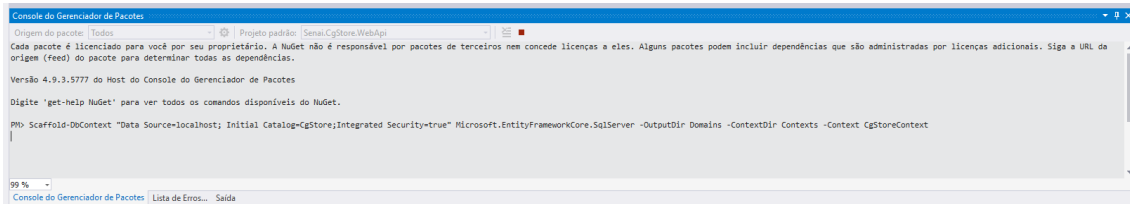
    app.UseSwagger();

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

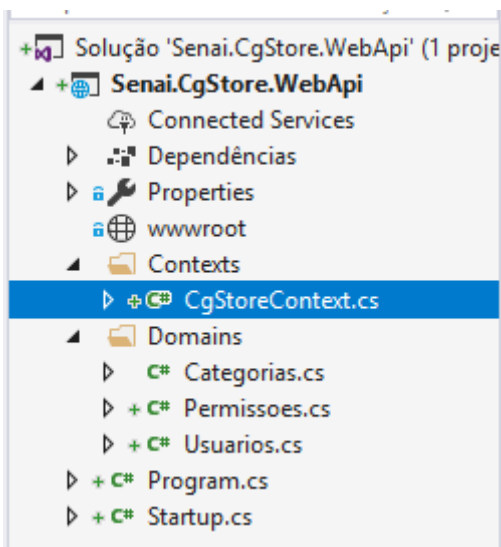
    app.UseMvc();
}
```

Como estamos utilizando o EFCore, vamos realizar o Scaffold para dar início aos modelos da nossa aplicação.

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



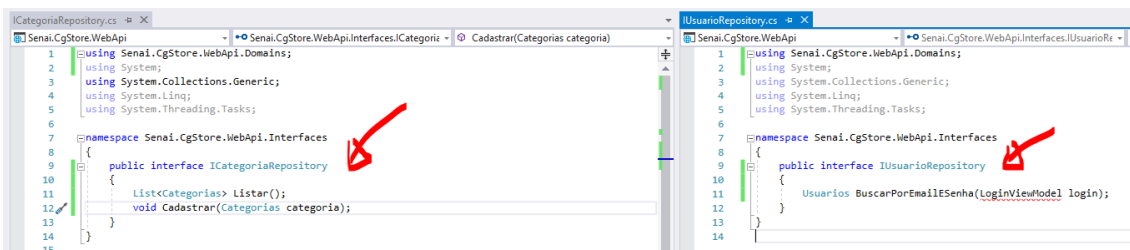
Solução com o projeto.



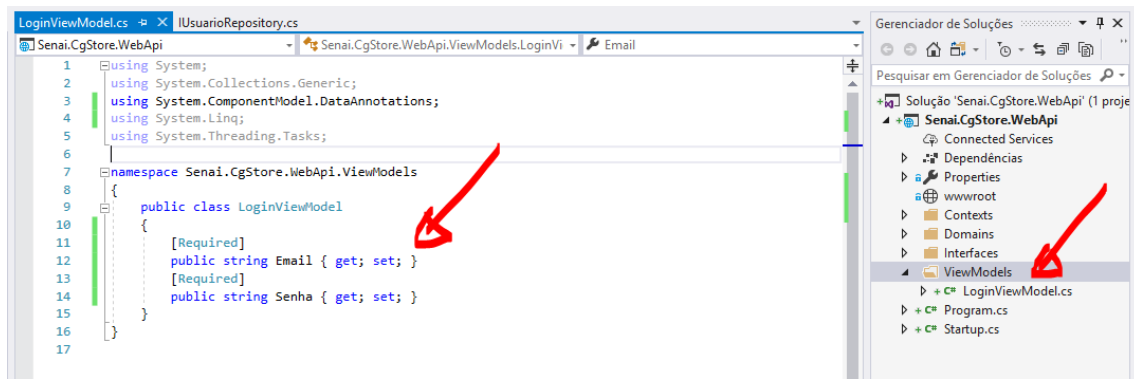
Criar as interfaces antes de criar os repositórios, bem como antes de criar os controllers. Para já definir o que será utilizado.

ICategoriaRepository.cs

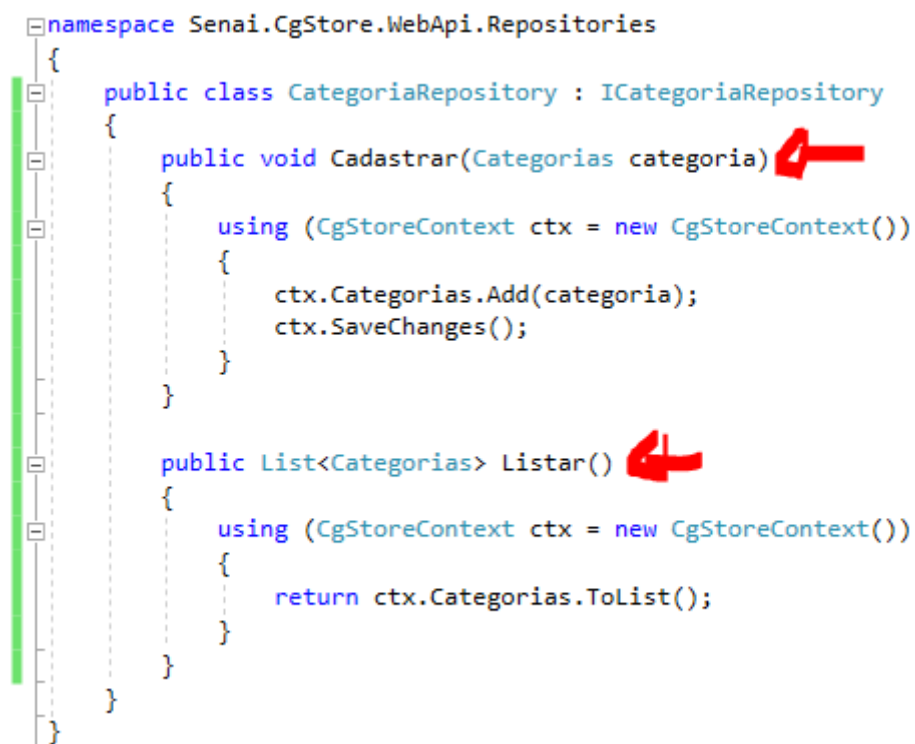
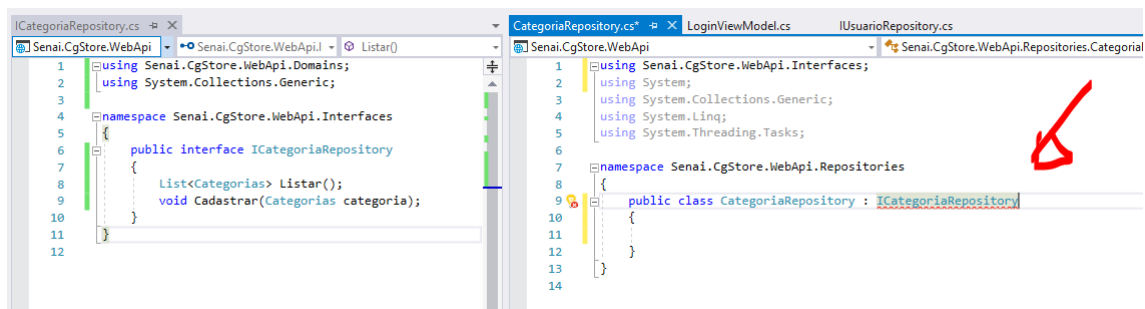
IUsuarioRepository.cs



Criar o LoginViewModel e realizar o using na interface.



Criar uma classe CategoriaRepository e realizar a chamada a interface.



Criar o Controller de Categorias para realizar a listagem.

CategoriasController.cs

```

namespace Senai.CgStore.WebApi.Controllers
{
    [Route("api/[controller]")]
    [Produces("application/json")]
    [ApiController]
    public class CategoriasController : ControllerBase
    {
        private ICategoriaRepository CategoriaRepository { get; set; }

        public CategoriasController()
        {
            CategoriaRepository = new CategoriaRepository();
        }

        [HttpGet]
        public IActionResult Listar()
        {
            return Ok(CategoriaRepository.Listar());
        }
    }
}

```

Lembrar de: launchSettings.json, alterar a opção do navegador para falso.

Lembrar de incluir os comentários no controller e no repositório

Postman

Criar uma coleção: Senai.CgStore e uma pasta Categorias.

Criar uma requisição Categorias.Listar.

Após apresentar a lista, vamos criar o login para gerar o token e restringir o acesso ao cadastro de novas categorias.

Mas antes disso, vamos criar o repositório para ter acesso ao banco de dados.

```

namespace Senai.CgStore.WebApi.Repositories
{
    public class UsuarioRepository : IUsuarioRepository
    {
        public Usuarios BuscarPorEmailESenha(LoginViewModel login)
        {
            using (CgStoreContext ctx = new CgStoreContext())
            {
                Usuarios usuario = ctx.Usuarios.Include(x => x.Permissao).FirstOrDefault(x => x.Email == login.Email && x.Senha == login.Senha);
                if (usuario == null)
                {
                    return null;
                }
                return usuario;
            }
        }
    }
}

```

Para configurar toda a parte do token e gerá-lo, precisamos de duas configurações.

Primeira: Configurar o startup.cs.

ConfigureServices

```

services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = "JwtBearer";
    options.DefaultChallengeScheme = "JwtBearer";
}).AddJwtBearer("JwtBearer", options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,

        ValidateAudience = true,

        ValidateLifetime = true,

        IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("cgstore-chave-autenticacao")),

        ClockSkew = TimeSpan.FromMinutes(30),

        ValidIssuer = "CgStore.WebApi",

        ValidAudience = "CgStore.WebApi"
    };
});

```

Configure

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseAuthentication();

    app.UseSwagger();

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

    app.UseMvc();
}

```



Após isto, precisamos criar o controller para gerar o token.

LoginController.cs

```
//UsuarioRepository UsuarioRepository = new UsuarioRepository();

private IUsuarioRepository UsuarioRepository { get; set; }

public LoginController()
{
    UsuarioRepository = new UsuarioRepository();
}

[HttpPost]
public IActionResult Login(LoginViewModel login)
{
    try
    {
        Usuarios usuarioBuscado = UsuarioRepository.BuscarPorEmailESenha(login);
        if (usuarioBuscado == null)
            return NotFound(new { mensagem = "Eita, deu ruim." });

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

        var key = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("cgstore-chave-autenticacao"));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: "CgStore.WebApi",
            audience: "CgStore.WebApi",
            claims: claims,
            expires: DateTime.Now.AddMinutes(30),
            signingCredentials: creds);

        return Ok(new
        {
            token = new JwtSecurityTokenHandler().WriteToken(token)
        });
    }
    catch (System.Exception ex)
    {
        return BadRequest(new { mensagem = ex.Message });
    }
}
}
```

Validar no postman e no jwt.io se o token está válido com as informações solicitadas.

Realizar o cadastro no CategoriasController.cs.


```
[Authorize(Roles = "Administrador")]
[HttpPost]
public IActionResult Cadastrar(Categorias categoria)
{
    try
    {
        int UsuarioId = Convert.ToInt32(HttpContext.User.Claims.First(x => x.Type == JwtRegisteredClaimNames.Jti).Value);
        categoria.UsuarioId = UsuarioId;
        categoria.DataCriacao = DateTime.Now;
        CategoriaRepository.Cadastrar(categoria);
        return Ok();
    }
    catch (System.Exception ex)
    {
        return BadRequest(new { mensagem = ex.Message });
    }
}
}
```


Validar o Swagger.

Na inclusão de um novo item, como BuscarPorId, primeiro atualizar a interface e depois realizar a chamada no repositório.

Depois, incluir no controller.

```
public interface ICategoriaRepository
{
    /// <summary>
    /// Listar todos as categorias
    /// </summary>
    /// <returns>Lista de Categorias</returns>
    List<Categorias> Listar();
    /// <summary>
    /// Cadastrar uma Categoria
    /// </summary>
    /// <param name="categoria">Categorias</param>
    void Cadastrar(Categorias categoria);

    Categorias BuscarPorId(int id); 
```

```
public class CategoriaRepository : ICategoriaRepository
{
    public Categorias BuscarPorId(int id)
    {
        using (CgStoreContext ctx = new CgStoreContext())
        {
            return ctx.Categorias.Find(id); 
        }
    }
}
```

```
[HttpGet("{id}")]
public IActionResult BuscarPorId(int id)
{
    try
    {
        Categorias categoria = CategoriaRepository.BuscarPorId(id);
        if (categoria == null)
            return NotFound();
        return Ok();
    }
    catch (Exception ex)
    {
        return BadRequest(new { mensagem = ex.Message });
    }
}
```