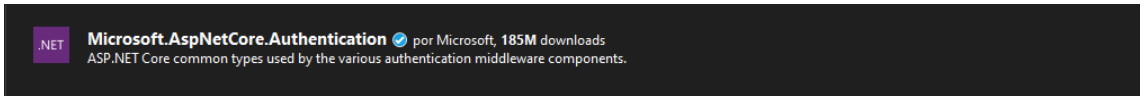
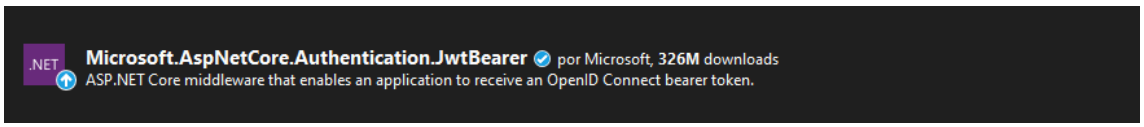
JWT ASP.NET

Instalando pacotes:

Antes de configurarmos o Auth devemos instalar os seguintes pacotes:





Após isso devemos configurar uma forma de “login” pode ser utilizando o .net Identity, alguma entidade no banco de dados ou por codigo mesmo:

```
2 referências
public partial class Admin
{
    2 referências
    public int Id { get; set; }

    1 referência
    public string Nome { get; set; } = null!;

    1 referência
    public string Senha { get; set; } = null!;
}
```

```
4 referências
public partial class TesteContext : DbContext
{
    0 referências
    public TesteContext()
    {
    }

    0 referências
    public TesteContext(DbContextOptions<TesteContext> options)
        : base(options)
    {
    }

    0 referências
    public virtual DbSet<Admin> Admins { get; set; }

    0 referências
```

Configurando a chave secreta:

Bom para gerarmos os tokens devemos primeiro criar uma chave secreta para a aplicação, vamos fazer isso criando uma classe **settings.cs** na raiz do projeto e adicionando a variavel secreta:

```
namespace WebApplication1
{
    0 referências
    public static class Settings
    {
        public static string Secret = "1ff947a49ed0230ca7874c5505dc195b";
    }
}
```

Gerando o Token:

Após esses passos vamos configurar o nosso serviço de geração de token:

```
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Text;
using WebApplication1.Models;
using System.Security.Claims;

namespace WebApplication1.Services
{
    0 referências
    public static class TokenService
    {
        0 referências
        public static string GenerateToken(Admin admin)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(Settings.Secret);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new Claim[]
                {
                    new Claim(ClaimTypes.Name, admin.Nome),
                    new Claim("Id", admin.Id.ToString()),
                }),
                Expires = DateTime.UtcNow.AddHours(8),
                SigningCredentials = new SigningCredentials
                ( new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);

        }
    }
}
```

Basicamente o nosso serviço que ira gerar o token gera um token com base na JwtSecurity, basicamente um token é composto por “subject”, “Expires” e “SigningCredentials”, subject é entendido como informações do usuario e nos podemos escolher quais serão elas, o Expires determina o tempo que o token gerado sera valido e o SigningCredentials é basicamente a chave do sistema que geramos na classe anterior.

Configurando a classe Program.cs:

Devemos configurar a nossa classe do programa com UseAuthentication e UseAuthorization:

```
app.UseAuthentication();
app.UseAuthorization();
```

Após isso devemos configurar o processo de Authentication no Services:

```
});
//pegamos a chave do sistema
var key = Encoding.ASCII.GetBytes(Settings.Secret);
//Configuração de autenticação
builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

    // adicionando as validações pelo JwtBearer
}).AddJwtBearer(x=>
{
    //Desativado a obrigação de https
    x.RequireHttpsMetadata = false;
    //Configurado para salvar o Token porem não vamos persistir ele
    x.SaveToken = true;
    //Aqui vai a configuração de validações do token
    x.TokenValidationParameters = new TokenValidationParameters()
    {
        //ativa a validação por chave
        ValidateIssuerSigningKey = true,
        //passamos a chave do sistema
        IssuerSigningKey = new SymmetricSecurityKey(key),
        //Essas configurações são mais complexas e se referem a Auth
        ValidateIssuer = false,
        ValidateAudience = false,

    };
});
```

Criando um método de login:

Bom após termos configurado os tokens na aplicação podemos criar um método login que ira criar um token para um usuario, primeiro vamos criar o service de Admin e criar o método de login:

```
3 referências
public class AdminService
{
    private readonly TesteContext db;
    1 referência
    public AdminService(TesteContext context)
    {
        db = context;
    }
    1 referência
    public async Task<Admin> Login(string Nome, string Senha)
    {
        return await Task.Run(() => db.Admins.Where(b => b.Nome == Nome && b.Senha == Senha).FirstOrDefault());
    }
}
}
```

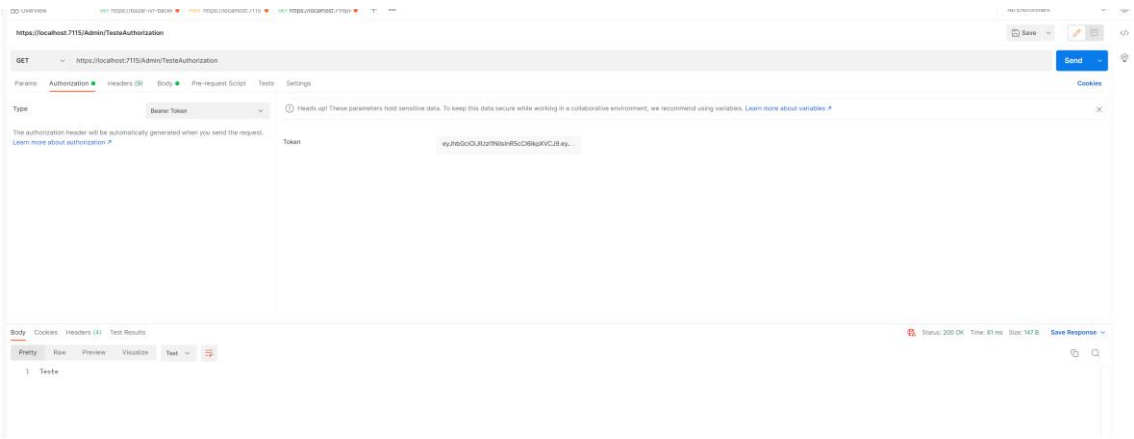
Em seguida vamos criar o seu controller que possui o método Login:

```
[ApiController]
[Route("[controller]")]
1 referência
public class AdminController : ControllerBase
{
    private readonly AdminService adminService;
    0 referências
    public AdminController(TesteContext context)
    {
        adminService = new AdminService(new TesteContext());
    }
    [HttpPost]
    [Route("Login")]
    0 referências
    public async Task<ActionResult<dynamic>> Login([FromBody] Admin usuario)

    {
        Console.WriteLine(usuario.Nome);
       Admin admin = await adminService.Login(usuario.Nome, usuario.Senha);

        if(admin == null)
        {
            return NotFound(new {message = "Usuario ou senha inválidos"});
        }
        else
        {
            var token = TokenService.GenerateToken(admin);
            return Ok(
                new
                {
                    user = admin,
                    token = token
                });
        }

    }
}
```

Aplicando autorizações nos métodos:

Após termos gerado um token podemos adicionar uma verificação nos métodos caso o token passado seja errado ele vai retornar o error 401:

```
[HttpGet]
[Route("TesteAuthorization")]
[Authorize]
0 referências
public string testeAuthorization()
{
    return "Teste";
}
```



Podemos tambem adicionar alguma propriedade do token a ser verificada no Roles, basicamente esse Roles é uma propriedade do token e pode ser uma função do usuario ou estatus:

```
[HttpGet]
[Route("TesteAuthorization")]
[Authorize(Roles = "admin, usuario")]
0 referências
public string testeAuthorization()
{
    return "Teste";
}
```

Melhorando a autorização e adicionando as politicas no ASP.NET 6:

Apartir do .net 6 temos um conceito de “polices” que são politicas de usuarios já definidas:

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("Admin", policy => policy.RequireRole("Admin"));
    options.AddPolicy("Usuario", policy => { policy.RequireRole("Usuario"); policy.RequireRole("Visitante"); });
});
```

Basicamente podemos definir niveis de usuarios com Base no Name do Token ou na Role.

Para isso vamos adicionar um Role na nossa classe Admin e adicionar o Role no Token:

```
6 referências
public class Admin
{
    3 referências
    public int Id { get; set; }

    5 referências
    public string Nome { get; set; } = null!;

    3 referências
    public string Senha { get; set; } = null!;

    0 referências
    public string Role { get; set; } = null!;
}
```

```
    {
        1 referência
        public static string GenerateToken(Admin admin)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(Settings.Secret);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new Claim[]
                {
                    new Claim(ClaimTypes.Name, admin.Nome),
                    new Claim(ClaimTypes.Role, admin.Role),
                    new Claim("Id", admin.Id.ToString()),
                }),
                Expires = DateTime.UtcNow.AddHours(8),
                SigningCredentials = new SigningCredentials
                ( new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);

        }
    }
}
```

Claramente devemos mudar a estrutura dos dados no banco de dados e adicionar os usuarios de exemplo:

```
create table admins (
Id integer auto_increment primary key,
Nome varchar(80) not null,
Senha varchar(80) not null,
Role varchar(80) not null
) default charset utf8mb4;
insert into admins (Nome, Senha, Role) values
("teste","teste", "Admin"),
("teste2","teste2", "Usuario"),
("teste3","teste3", "Visitante");
```

Após isso basta adicionarmos o atributo Policy as nossas anotações “Authorize”:

```
    [HttpGet]
    [Route("TesteAuthorization")]
    [Authorize(Policy = "Admin")]
    0 referências
    public string testeAuthorization()
    {
        return "Teste";
    }
```

Dessa forma os usuarios que possuem uma “Policy” diferente não poderão acessar o método mesmo que tenham um token, as “policies” basicamente são agregadores de permissões no sistema assim não precisamos todas as vezes declarar quais são os usuarios validos para acessar o método basta usarmos a “policy” adequada.