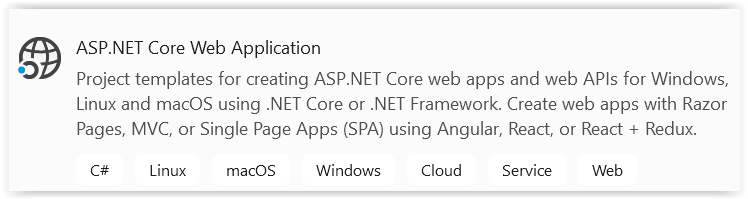
**IMPLEMENTAR SEGURIDAD OAUTH 2 EN WEB API CORE CONSUMO CLIENTE CORE**

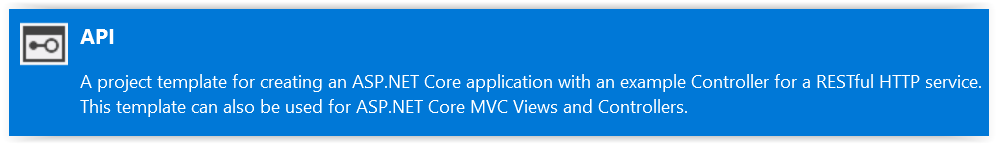
* Vamos a realizar una aplicación Web Api en la que implementaremos seguridad Web Api utilizando OAuth 2.0.
* Dicha seguridad se basa en tokens generados por el servidor y que tienen que ser incluidos en las cabeceras de los métodos seguros para incluir acceso.
* Dicha seguridad se conoce como Bearer Token OAuth 2.0.
* Es la seguridad más completa para acceso a Api, ya que no se depende de Cookies, ni de elementos Web, simplemente de peticiones y ahí es donde entra su importancia, ya que los dispositivos móviles son la clave para su demanda.
* El token no es fijo en el tiempo, sino que podemos ofrecerle un tiempo de duración determinado.
* La seguridad de los tokens se establece en dos partes:
  + Por un lado, el token es generado en una petición post a una url del servidor. En dicha llamada se envía el usuario y password y se genera dicho token.
  + Por otro lado, teniendo ya el token, se envía en la llamada al método que se necesita, validando dicho token con la petición anterior, que es una combinación de tiempo, url de petición y otras características.

Si se realiza una petición desde otro servidor (sniffer) o con otras credenciales a la inicial, se bloquea la petición al método Get de acceso.



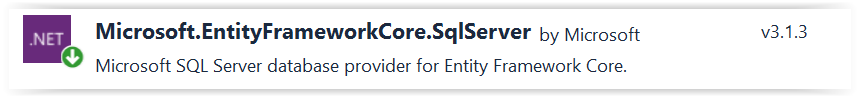
Para probar la seguridad, nos crearemos un nuevo proyecto **MVC Core** llamado **ApiEmpleadosCoreOAuth** de tipo **API**

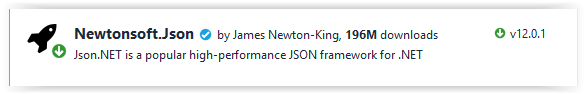




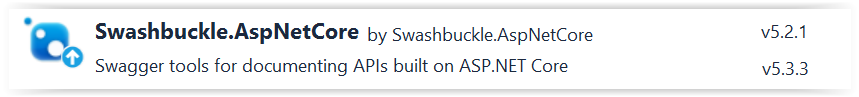
Sobre el proyecto, agregamos el NuGet de **Entity Framework Core**, **Entity Framework Core Sql Server** y **NewtonSoft.json**.







Agregamos también el NuGet **Swashbuckle.AspNetCore** para poder documentar el servicio Api con **Swagger**.



Sobre la carpeta **Models** y agregamos la clase **Empleado**

**EMPLEADO**

[Table("EMP")]

**public** **class** Empleado

{

    [Key]

    [DatabaseGenerated(DatabaseGeneratedOption**.**None)]

    [Column("EMP\_NO")]

**public** **int** IdEmpleado { **get**; **set**; }

    [Column("APELLIDO")]

**public** String Apellido { **get**; **set**; }

    [Column("OFICIO")]

**public** String Oficio { **get**; **set**; }

    [Column("SALARIO")]

**public** **int** Salario { **get**; **set**; }

    [Column("DIR")]

**public** **int** Director { **get**; **set**; }

}

Sobre el proyecto, creamos la carpeta **Data** y la clase **EmpleadosContext**

**EMPLEADOSCONTEXT**

**public** **class** EmpleadosContext : DbContext

{

**public** EmpleadosContext(DbContextOptions<EmpleadosContext> options):

**base**(options) { }

**public** DbSet<Empleado> Empleados { **get**; **set**; }

}

Creamos la carpeta **Repositories** y la clase **RepositoryEmpleados**

**REPOSITORYEMPLEADOS**

using ApiEmpleadosCoreOAuth.Data;

using ApiEmpleadosCoreOAuth.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ApiEmpleadosCoreOAuth.Repositories

{

    public class RepositoryEmpleados

    {

        EmpleadosContext context;

        public RepositoryEmpleados(EmpleadosContext context)

        {

            this.context = context;

        }

        public List<Empleado> GetEmpleados()

        {

            return this.context.Empleados.ToList();

        }

        public Empleado BuscarEmpleado(int idempleado)

        {

            return this.context.Empleados

                .SingleOrDefault(x => x.IdEmpleado == idempleado);

        }

        public Empleado ExisteEmpleado(String apellido, int empno)

        {

            return this.context.Empleados.SingleOrDefault

                (x => x.Apellido == apellido && x.IdEmpleado == empno);

        }

        public List<Empleado> GetEmpleadosSubordinados(int idempleado)

        {

            var consulta = from datos in context.Empleados

                           where datos.Director == idempleado

                           select datos;

            return consulta.ToList();

        }

    }

}

A continuación, vamos a trabajar sobre el archivo **Settings** de la aplicación.

Incluiremos una cadena de conexión a la base de datos SQL Server Azure y unas claves que necesitaremos posteriormente.



Tendremos que incluir los siguientes parámetros para la generación del token:

* **Issuer**: Se utiliza para poner la URL de nuestro servidor. Este parámetro es opcional.
* **Audience**: Sirve para dar el nombre a nuestro API.
* **SecretKey**: valor único que se utiliza para para proteger nuestro token y generarlo dinámicamente.

**Nota:** El Secret Key debe ser superior a 8 caracteres. Deberíamos incluir letras, números y símbolos.

**APPSETTINGS.JSON**

{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft": "Warning",

      "Microsoft.Hosting.Lifetime": "Information"

    }

  },

  "AllowedHosts": "\*",

  "ConnectionStrings": {

    "hospitalazurexamarin": "Data Source=sqlxamarin.database.windows.net;Initial Catalog=HOSPITALXAMARIN;Persist Security Info=True;User ID=adminsql;Password=Admin123"

  },

  "ApiAuth": {

    "Issuer": "https://localhost:44375/",

    "Audience": "ApiEmpleadosCoreOAuth",

    "SecretKey": "MvcCor€Cl@veS€cretKe8@Tok€N"

  }

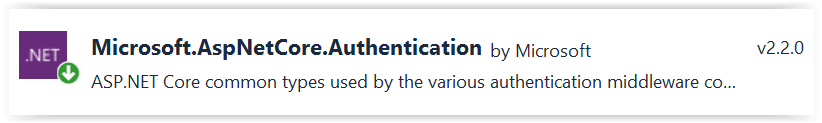
}

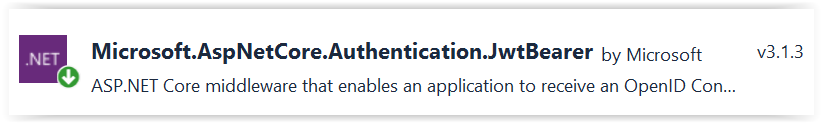
A continuación, es el momento de implementar la seguridad de la aplicación API mediante un Token.

Tendremos métodos que contendrán seguridad y otros métodos que serán libres.

Los métodos seguros deberán recibir un Token en el API para poder ser utilizados.

Debemos agregar los NuGet que controlan la seguridad para el Token:







Para hacer el código más limpio, vamos a crearnos una clase que será la encargada de crear las opciones de autentificación de la aplicación y las opciones del Token.

Creamos una nueva carpeta llamada **Token** y dentro una clase llamada **HelperToken**

**HELPERTOKEN**

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ApiEmpleadosCoreOAuth.Token

{

    public class HelperToken

    {

        public String Issuer { get; set; }

        public String Audience { get; set; }

        public String Secretkey { get; set; }

        public HelperToken(IConfiguration configuration)

        {

            this.Issuer = configuration["ApiAuth:Issuer"];

            this.Audience = configuration["ApiAuth:Audience"];

            this.Secretkey = configuration["ApiAuth:SecretKey"];

        }

        //CREAMOS UN METODO PARA GENERAR UNA CLAVE

        //SIMETRICA A PARTIR DE NUESTRO SecretKey

        public SymmetricSecurityKey GetKeyToken()

        {

            byte[] data =

                System.Text.Encoding.UTF8.GetBytes(this.Secretkey);

            return new SymmetricSecurityKey(data);

        }

        //METODO PARA CONFIGURAR LAS OPCIONES DE SEGURIDAD DEL TOKEN

        //LOS METODOS DE CONFIGURACION SON Action

        public Action<JwtBearerOptions> GetJwtOptions()

        {

            Action<JwtBearerOptions> jwtoptions =

                new Action<JwtBearerOptions>(options => {

                    options.TokenValidationParameters =

                    new TokenValidationParameters()

                    {

ValidateActor = true, ValidateAudience= true, ValidateLifetime = true

, ValidateIssuerSigningKey = true

, ValidIssuer = this.Issuer

, ValidAudience = this.Audience

, IssuerSigningKey = this.GetKeyToken()

                    };

                });

            return jwtoptions;

        }

        //METODO PARA VALIDAR LA AUTENTIFICACION

        public Action<AuthenticationOptions> GetAuthOptions()

        {

            Action<AuthenticationOptions> authoptions =

                new Action<AuthenticationOptions>(options =>

                {

                    options.DefaultAuthenticateScheme

                    = JwtBearerDefaults.AuthenticationScheme;

                    options.DefaultScheme =

                    JwtBearerDefaults.AuthenticationScheme;

                    options.DefaultChallengeScheme =

                    JwtBearerDefaults.AuthenticationScheme;

                });

            return authoptions;

        }

    }

}

Ahora vamos a irnos a la clase **Startup** para configurar la seguridad de nuestra aplicación.

**STARTUP.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ApiEmpleadosCoreOAuth.Data;

using ApiEmpleadosCoreOAuth.Repositories;

using ApiEmpleadosCoreOAuth.Token;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using Microsoft.OpenApi.Models;

namespace ApiEmpleadosCoreOAuth

{

    public class Startup

    {

        public Startup(IConfiguration configuration)

        {

            Configuration = configuration;

        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.

        public void ConfigureServices(IServiceCollection services)

        {

            String cadena =

                this.Configuration.GetConnectionString("hospitalazurexamarin");

            services.AddTransient<RepositoryEmpleados>();

            services.AddDbContext<EmpleadosContext>(options

                => options.UseSqlServer(cadena));

            //SWAGGER

            services.AddSwaggerGen(

                c =>

                {

                    //VERSION 2 Y VERSION 1

                    c.SwaggerDoc(

            name: "v1", new OpenApiInfo

            {

                Title = "Api Empleados Seguridad OAuth"

            ,

                Version = "v1",

                Description = "Ejemplo de seguridad OAuth Token"

            });

                });

            HelperToken helper = new HelperToken(this.Configuration);

            //AÑADIMOS AUTENTIFICACION A NUESTRO SERVICIO

            services.AddAuthentication(helper.GetAuthOptions())

                .AddJwtBearer(helper.GetJwtOptions());

            services.AddControllers();

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

        {

            if (env.IsDevelopment())

            {

                app.UseDeveloperExceptionPage();

            }

            app.UseSwagger();

            //UI INDICA DONDE VA A VISUALIZAR EL USUARIO LA DOCUMENTACION

            //GENERADA POR SWAGGER EN NUESTRO SERVIDOR

            app.UseSwaggerUI(

                c =>

                {

                    //DEBEMOS CONFIGURAR LA URL DEL SERVIDOR

                    //PARA LA DOCUMENTACION

                    c.SwaggerEndpoint(

                        url: "/swagger/v1/swagger.json"

                        , name: "Api v1");

                    c.RoutePrefix = "";

                });

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthentication();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>

            {

                endpoints.MapControllers();

            });

        }

    }

}

El siguiente paso será crearnos un controlador para poder generar el Token cuando el usuario nos realice una petición.

Para que el usuario nos pueda enviar la información con formato Json, nos crearemos una clase para mapearla.

Sobre la carpeta **Models**, creamos una nueva clase llamada **LoginModel**.

**LOGINMODEL**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ApiEmpleadosCoreOAuth.Models

{

    public class LoginModel

    {

        public String UserName { get; set; }

        public String Password { get; set; }

    }

}

Sobre la carpeta **Controllers**, creamos un nuevo controlador **Web Api Empty** llamado **AuthController**.

**AUTHCONTROLLER**

using System;

using System.Collections.Generic;

using System.IdentityModel.Tokens.Jwt;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using ApiEmpleadosCoreOAuth.Models;

using ApiEmpleadosCoreOAuth.Repositories;

using ApiEmpleadosCoreOAuth.Token;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Configuration;

using Microsoft.IdentityModel.Tokens;

using Newtonsoft.Json;

namespace ApiEmpleadosCoreOAuth.Controllers

{

    //www.apiempleados/Auth

    [Route("[controller]")]

    [ApiController]

    public class AuthController : ControllerBase

    {

        RepositoryEmpleados repo;

        HelperToken helper;

        public AuthController(RepositoryEmpleados repo

            , IConfiguration configuration)

        {

            this.helper = new HelperToken(configuration);

            this.repo = repo;

        }

        //NECESITAMOS UN PUNTO DE ENTRADA (ENDPOINT) PARA QUE EL

        //USUARIO NOS ENVIE LOS DATOS DE SU VALIDACION

        //LOS ENDPOINT AUTH SON POST

        //LO QUE RECIBIREMOS SERA UserName y Password

        //QUE NOSOTROS LO HEMOS INCLUIDO CON LoginModel

        [HttpPost]

        [Route("[action]")]

        public IActionResult Login(LoginModel model)

        {

            Empleado empleado =

                this.repo.ExisteEmpleado(model.UserName

                , int.Parse(model.Password));

            if (empleado != null)

            {

                //NECESITAMOS CREARNOS UN TOKEN

                //EL TOKEN LLEVARA INFORMACION DE TIPO ISSUER

                //, TIEMPO DE DURACION

                //, CREDENCIALES DEL USUARIO

                //, PODEMOS ALMACENAR INFO EXTRA DENTRO DEL TOKEN.

                //VAMOS A ALMACENAR A NUESTRO EMPLEADO

                Claim[] claims = new[]

                {

                    new Claim("UserData",

                    JsonConvert.SerializeObject(empleado))

                };

                JwtSecurityToken token = new JwtSecurityToken

                    (

                     issuer: helper.Issuer

                     , audience: helper.Audience

                     , claims: claims

                     , expires: DateTime.UtcNow.AddMinutes(10)

                     , notBefore: DateTime.UtcNow

                     , signingCredentials:

new SigningCredentials(this.helper.GetKeyToken(), SecurityAlgorithms.HmacSha256)

                    );

                //DEVOLVEMOS UNA RESPUESTA AFIRMATIVA

                //CON SU TOKEN

                return Ok(

                    new

                    {

                        response =

                        new JwtSecurityTokenHandler().WriteToken(token)

                    });

            }

            else

            {

                return Unauthorized();

            }

        }

    }

}

Ahora es el momento de incluir nuestra API de consumo para que los usuarios puedan recuperar datos.

Creamos un controlador con los métodos típicos de Get y Get(id)

Sobre **Controllers**, creamos un nuevo controlador Web API vacío llamado **EmpleadosController**.

Primero haremos dos métodos simples para comprobar que nos devuelve los datos y, posteriormente, incluiremos la seguridad y funcionalidad con el usuario y token.

**EMPLEADOSCONTROLLER**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using ApiEmpleadosCoreOAuth.Models;

using ApiEmpleadosCoreOAuth.Repositories;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

namespace ApiEmpleadosCoreOAuth.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class EmpleadosController : ControllerBase

    {

        RepositoryEmpleados repo;

        public EmpleadosController(RepositoryEmpleados repo)

        {

            this.repo = repo;

        }

        [HttpGet]

        public ActionResult<List<Empleado>> GetEmpleados()

        {

            return this.repo.GetEmpleados();

        }

        [HttpGet("{id}")]

        public ActionResult<Empleado> BuscarEmpleado(int id)

        {

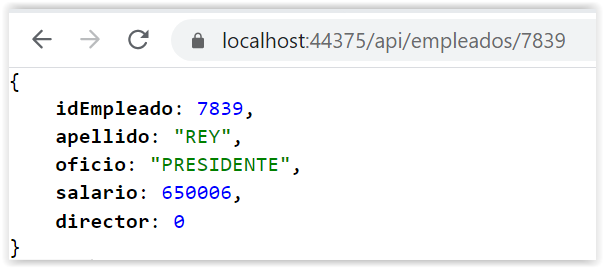
            return this.repo.BuscarEmpleado(id);

        }

    }

}

Ya podemos ejecutar la aplicación y comprobar que los métodos del controlador Empleados son accesibles desde cualquier programa Rest Client, como Postman o Insomnia Rest.



Es el momento de incluir seguridad a nuestro método GET de empleados.

Para ello, le incluiremos la cabecera [Authorize] en los métodos que requieran Autentificación.

[Authorize]

[HttpGet]

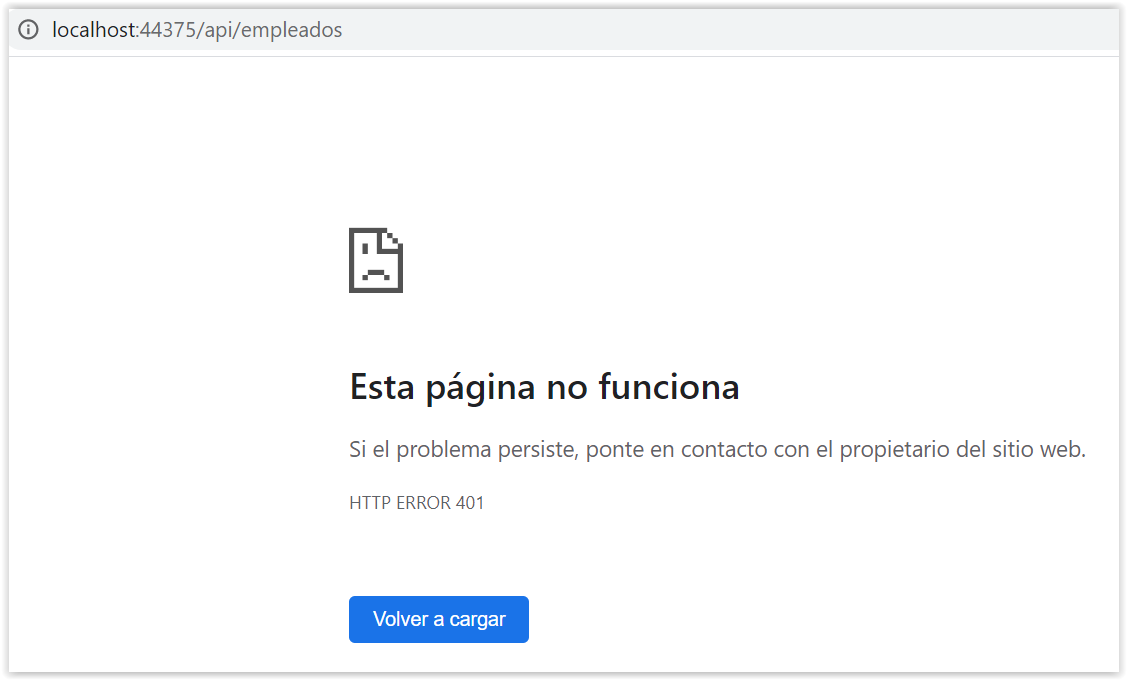
public ActionResult<List<Empleado>> GetEmpleados()

{

    return this.repo.GetEmpleados();

}

Al acceder al método, visualizaremos el error **401** que indica que no estamos autorizados para el acceso al método.



Comprobaremos que al método GET no nos permite acceso, pero al método Get(id) sí que tenemos acceso.

Ahora es cuando tendríamos que crearnos un cliente que consumiera primero el token de autorización y lo enviase en la petición al método get de empleados.

Para probar que la seguridad está habilitada, vamos a utilizar un cliente llamado POSTMAN o INSOMNIA que nos permite realizar todo tipo de peticiones al servidor.

Podemos instalar postman con las aplicaciones del explorador Chrome.

Abrimos Insomnia y vamos a realizar una petición POST a la url que hemos indicado en nuestro controlador **AuthController**.

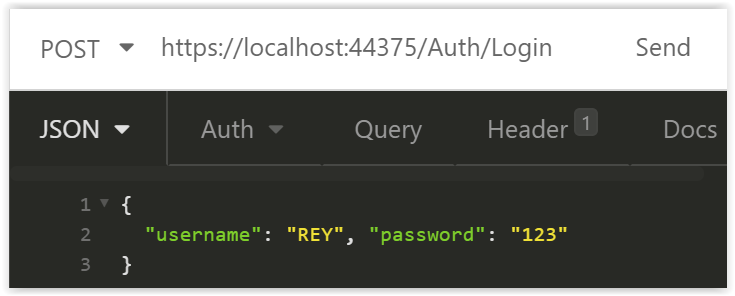
La url será:

<https://localhost:44375/Auth/Login>

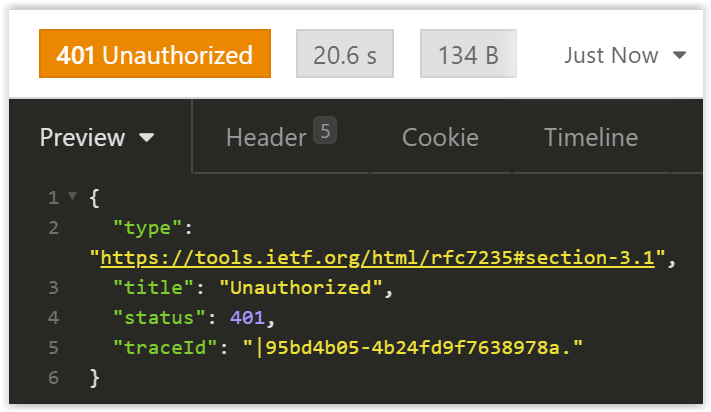
Los datos que debemos indicar en insomnia es una JSON con UserName y Password.

Debemos indicar que el método será **POST** y enviaremos un objeto JSON, tal y como podemos comprobar en la imagen.

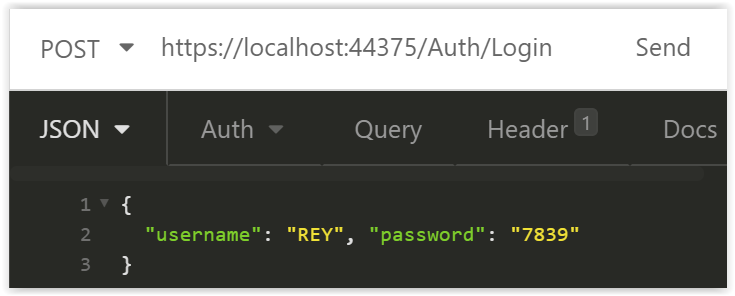
Vamos a comenzar incluyendo datos que NO son correctos.

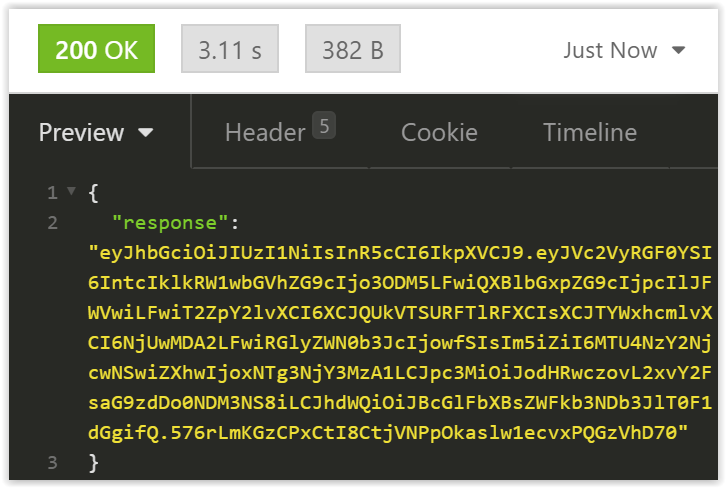


Si ponemos datos que no son correctos nos dará un error de autorización:



Ahora vamos a incluir los datos correctos y veremos que nos devuelve el token de acceso que hemos configurado para las peticiones que necesiten autorización en nuestra API.

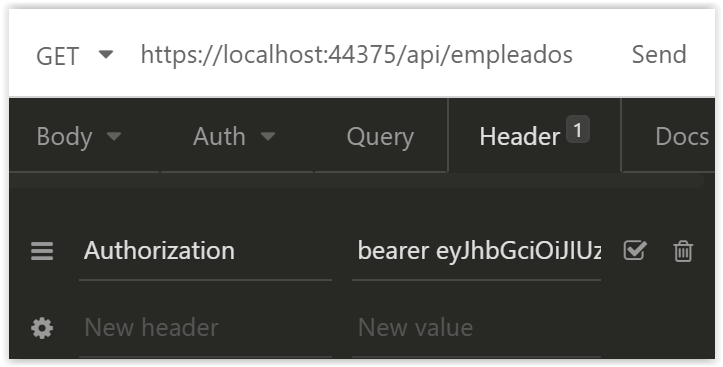




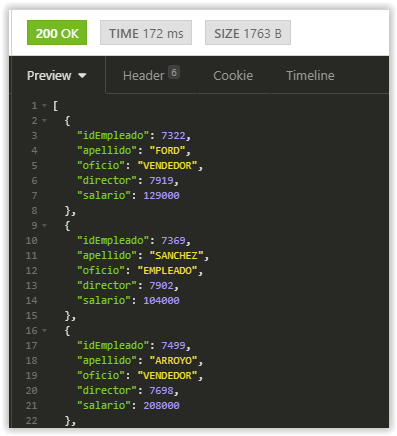
Con dicho token de salida ya estableceríamos la comunicación y autorización al método GET de empleados.

Para poder realizar la petición correcta, habría que incluir en la cabecera de la llamada al método Get la siguiente clave:

Authorization bearer TOKEN



Para comprobarlo, haremos una petición a api/Empleados incluyendo nuestro token.



Por último, vamos a configurar nuestro controlador **EmpleadosController** para recuperar la información del Empleado que se ha dado de alta en nuestro Api.

**EMPLEADOSCONTROLLER**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using ApiEmpleadosCoreOAuth.Models;

using ApiEmpleadosCoreOAuth.Repositories;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

namespace ApiEmpleadosCoreOAuth.Controllers

{

    [Route("api/[controller]")]

    [ApiController]

    public class EmpleadosController : ControllerBase

    {

        RepositoryEmpleados repo;

        public EmpleadosController(RepositoryEmpleados repo)

        {

            this.repo = repo;

        }

        [Authorize]

        [HttpGet]

        public ActionResult<List<Empleado>> GetEmpleados()

        {

            return this.repo.GetEmpleados();

        }

        [HttpGet("{id}")]

        public ActionResult<Empleado> BuscarEmpleado(int id)

        {

            return this.repo.BuscarEmpleado(id);

        }

        [Authorize]

        [HttpGet]

        [Route("[action]")]

        //PONEMOS ACTIONRESULT POR SI DESEAMOS ENVIAR RESPUESTAS

        //PERSONALIZADAS, ADEMAS DE LA INFORMACION

        //SI SOLO ENVIAMOS LA INFORMACION, BASTARIA CON EL OBJETO

        //A DEVOLVER

        public ActionResult<Empleado> PerfilEmpleado()

        {

            //EN ESTE METODO HEMOS RECIBIDO LOS CLAIMS DEL TOKEN

            //DICHOS CLAIMS SE ALMACENAN EN EL SERVICIO DENTRO

            //DEL USUARIO

            List<Claim> claims =

                HttpContext.User.Claims.ToList();

            //DENTRO DE ESTOS CLAIMS TENEMOS UserData O LO QUE HAYAMOS

            //ALMACENADO EN EL TOKEN

            //BUSCAMOS EL JSON DEL EMPLEADO EN UserData

            String json =

                claims.SingleOrDefault(x => x.Type == "UserData").Value;

            //CONVERTIMOS EL JSON A OBJETO

            Empleado emp =

                JsonConvert.DeserializeObject<Empleado>(json);

            return emp;

        }

        [Authorize]

        [HttpGet]

        [Route("[action]")]

        //PONEMOS ACTIONRESULT POR SI DESEAMOS ENVIAR RESPUESTAS

        //PERSONALIZADAS, ADEMAS DE LA INFORMACION

        //SI SOLO ENVIAMOS LA INFORMACION, BASTARIA CON EL OBJETO

        //A DEVOLVER

        public ActionResult<List<Empleado>> Subordinados()

        {

            //EN ESTE METODO HEMOS RECIBIDO LOS CLAIMS DEL TOKEN

            //DICHOS CLAIMS SE ALMACENAN EN EL SERVICIO DENTRO

            //DEL USUARIO

            List<Claim> claims =

                HttpContext.User.Claims.ToList();

            //DENTRO DE ESTOS CLAIMS TENEMOS UserData O LO QUE HAYAMOS

            //ALMACENADO EN EL TOKEN

            //BUSCAMOS EL JSON DEL EMPLEADO EN UserData

            String json =

                claims.SingleOrDefault(x => x.Type == "UserData").Value;

            //CONVERTIMOS EL JSON A OBJETO

            Empleado emp =

                JsonConvert.DeserializeObject<Empleado>(json);

            List<Empleado> empleados = this.repo.GetEmpleadosSubordinados(emp.IdEmpleado);

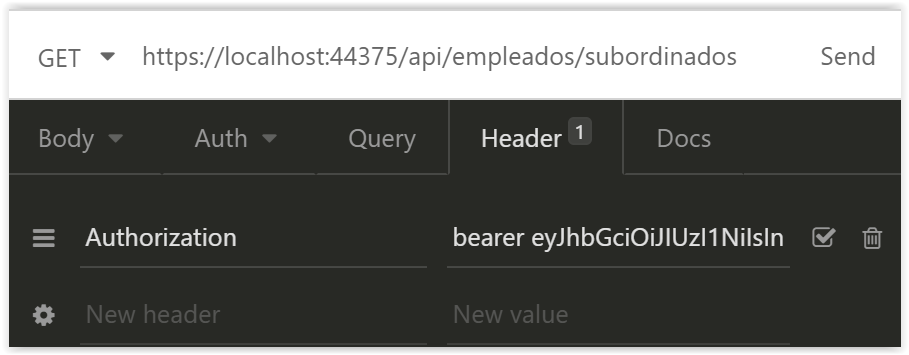
            return empleados;

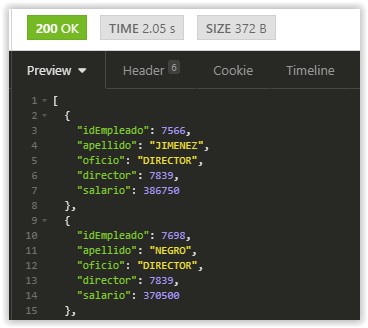
        }

    }

}

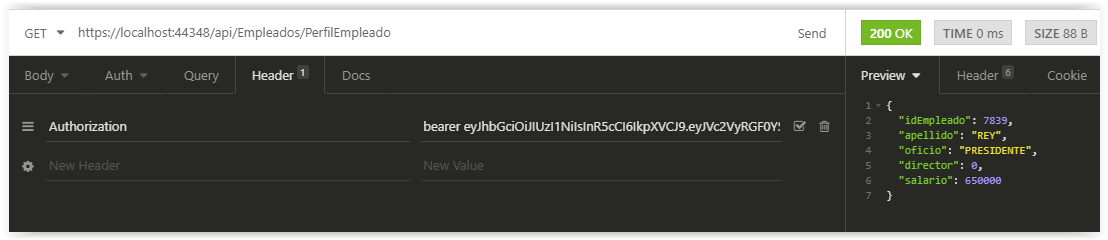
Si ejecutamos Insomnia llamando a alguno de los métodos personalizados con validación, veremos que ya nos devuelve el perfil o los subordinados.





También podremos visualizar nuestro Perfil con el Token.

<https://localhost:44348/api/Empleados/PerfilEmpleado>



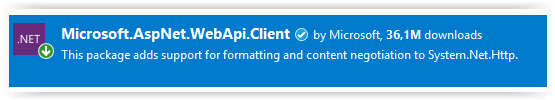
**Alojamos nuestro proyecto en Azure para trabajar.**

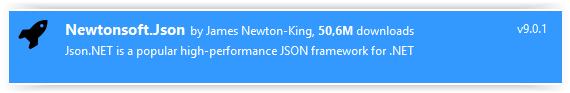
[**https://apiempleadoscoreoauth.azurewebsites.net**](https://apiempleadoscoreoauth.azurewebsites.net)

Ahora vamos a visualizar un ejemplo rápido para comprobar el consumo de la Api desde un cliente. Creamos una nueva aplicación de consola **Net Framework**.



Agregamos dos **NuGet**.





Creamos una clase llamada **Login**

**LOGIN**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

public class Login

{

public String UserName { get; set; }

public String Password { get; set; }

}

}

Sobre la clase **Program.cs** escribimos el siguiente código:

**PROGRAM**

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

const string userName = "REY";

const string password = "7839";

const string apiBaseUri = "https://localhost:44348/";

private static async Task<string> GetAPIToken(string userName, string password, string apiBaseUri)

{

using (HttpClient client = new HttpClient())

{

//setup client

client.BaseAddress = new Uri(apiBaseUri);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

Login login = new Login();

login.UserName = userName;

login.Password = password;

String json = JsonConvert.SerializeObject(login);

StringContent content =

new StringContent(json

, System.Text.Encoding.UTF8, "application/json");

String peticion = "Auth/Login";

HttpResponseMessage response =

await client.PostAsync(peticion, content);

String contenido =

await response.Content.ReadAsStringAsync();

var jObject = JObject.Parse(contenido);

return jObject.GetValue("response").ToString();

}

}

static async Task<string> GetRequest(string token, string requestPath)

{

using (HttpClient cliente = new HttpClient())

{

cliente.BaseAddress = new Uri(apiBaseUri);

cliente.DefaultRequestHeaders.Accept.Clear();

cliente.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

cliente.DefaultRequestHeaders.Add("Authorization", "bearer " + token);

HttpResponseMessage response = await cliente.GetAsync(requestPath);

var responseString = await response.Content.ReadAsStringAsync();

return responseString;

}

}

static void Main(string[] args)

{

//Get the token

var token = GetAPIToken(userName, password, apiBaseUri).Result;

Console.WriteLine("Token: {0}", token);

//Make the call

var response = GetRequest(token, "api/Empleados").Result;

Console.WriteLine("response: {0}", response);

//wait for key press to exit

Console.ReadKey();

}

}

}



**CLIENTE MVC CORE**

Una vez comprobado cómo funciona por código rápido, haremos una aplicación más compleja para validar los empleados.

Vamos a visualizar la funcionalidad con un proyecto MVC Core.

Montaremos la seguridad con un Login y con zonas protegidas.

Todos los recursos los utilizaremos de la Api.

Nada más comenzar, veremos una página estándar y podremos acceder a nuestro perfil o a nuestros subordinados utilizando la validación.

* Pediremos validación de usuarios en el MVC utilizando el acceso al token del API.

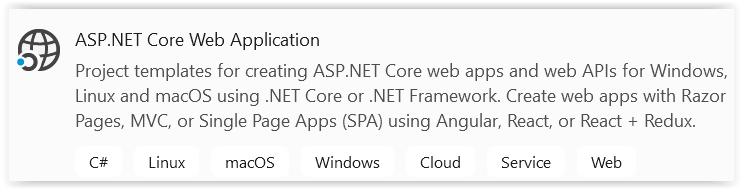


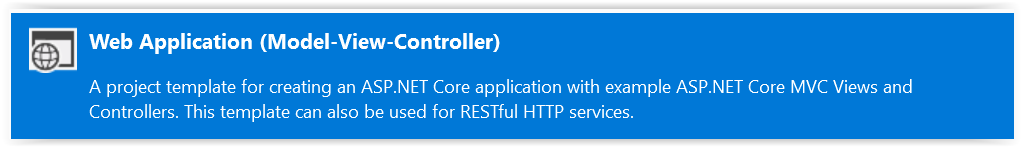
Cuando nos hayamos validado, podremos ver nuestros empleados subordinados o nuestro perfil.



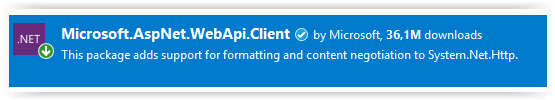


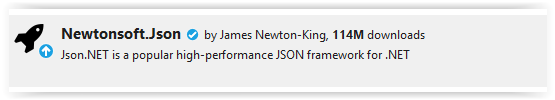
Comenzamos creando un nuevo proyecto de tipo **Mvc Core** con la plantilla MVC llamado **ClienteOAuthEmpleados**.

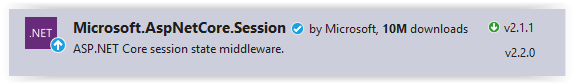




Agregamos los siguientes Nuget:







Sobre la carpeta **Controllers**, creamos un nuevo controlador llamado **ManageController**.

Tendremos una única vista llamada **Login** que contendrá un método Get y otro Post, recibiendo **usuario** y **password**.

**MANAGECONTROLLER**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

namespace ClienteApiSeguridadCore.Controllers

{

public class ManageController : Controller

{

public IActionResult Login()

{

return View();

}

[HttpPost]

public IActionResult Login(String usuario, String password)

{

return View();

}

}

}

Creamos la vista **Login.cshtml** con una caja de texto usuario y otra para password.

**LOGIN.CSHTML**

@{

    ViewData["Title"] = "Login";

}

<h1>Login Empleados</h1>

<form method="post">

    <div>

        <label>Apellido: </label>

        <input type="text" name="username" class="form-control"/>

    </div>

    <div>

        <label>Id Empleado: </label>

        <input type="text" name="password" class="form-control"/>

    </div>

    <button type="submit" class="btn btn-success">

        LogIn

    </button>

</form>

<h1 style="color:red">@ViewData["MENSAJE"]</h1>

El siguiente paso será crear el controlador para la zona que deseamos proteger con acceso de usuarios.

Sobre **Controllers**, creamos un nuevo controlador llamado **EmpleadosController**.

Y nos creamos su vista **Index.cshtml**, en la que implementaremos seguridad.

**INDEX.CSHTML**

@{

ViewData["Title"] = "Index";

}

<h2>Perfil del empleado</h2>

Por último, creamos un Link en la página **\_Layout** para acceder al perfil del usuario.

**\_LAYOUT.CSHTML**

<li class="nav-item">

<**a** **asp-action**="Index" class="nav-link"

**asp-controller**="Empleados">Zona empleados</**a**>

</li>

Comprobamos la navegación en nuestra Web y veremos que todo está correcto.

Ahora es el momento de montar la seguridad de nuestra aplicación.

Comenzamos creando una clase Atributo de autorización personalizada, para interceptar peticiones y mostrar la vista de Login.

Dicha clase debe heredar de **AuthorizeAttribute** y de **IAuthorizacionFilter**.

Sobre el proyecto, creamos una carpeta llamada **Filters** y agregamos una clase llamada **EmpleadosAuthorizeAttribute**.

**EMPLEADOSAUTHORIZEATTRIBUTE**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using Microsoft.AspNetCore.Routing;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ClienteOAuthEmpleados.Filters

{

    public class EmpleadosAuthorizeAttribute : AuthorizeAttribute

        , IAuthorizationFilter

    {

        public void OnAuthorization(AuthorizationFilterContext context)

        {

            //SOLO QUEREMOS SABER SI EL EMPLEADO SE HA VALIDADO O NO

            //CUANDO NO ESTE VALIDADO, LO ENVIAMOS A LOGIN

            var user =

                context.HttpContext.User;

            if (user.Identity.IsAuthenticated == false)

            {

                RouteValueDictionary ruta =

                    new RouteValueDictionary(

                        new

                        {

                            controller = "Manage"

                            , action = "Login"

                        });

                RedirectToRouteResult result =

                    new RedirectToRouteResult(ruta);

                context.Result = result;

            }

        }

    }

}

Ahora es el momento de montar nuestro acceso al Api.

Sobre **Models**, creamos una nueva clase llamada **Empleado**.

**EMPLEADO**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ClienteOAuthEmpleados.Models

{

public class Empleado

{

public int IdEmpleado { get; set; }

public String Apellido { get; set; }

public String Oficio { get; set; }

public int Director { get; set; }

public int Salario { get; set; }

}

}

También creamos un modelo para enviar nuestro username y password al Api mediante Json.

Sobre **Models**, creamos una nueva clase llamada **LoginModel**.

**LOGINMODEL**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ClienteOAuthEmpleados.Models

{

    public class LoginModel

    {

        public String UserName { get; set; }

        public String Password { get; set; }

    }

}

A continuación, crearemos nuestro repositorio para las consultas a la API de Azure.

Lo que haremos será implementar todos los métodos de la Api para poder utilizarlos dentro de nuestra aplicación.

**Nota:** Los métodos que en el API tengan la decoración [Authorize] tendrán que recibir el Token.

Creamos una nueva carpeta llamada **Repositories** y una clase llamada **RepositoryEmpleados**.

**REPOSITORYEMPLEADOS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Net;

using System.Net.Http;

using System.Net.Http.Headers;

using ClienteOAuthEmpleados.Models;

using Newtonsoft.Json;

using System.Text;

using Newtonsoft.Json.Linq;

using ClasesSeguridadNuGet;

namespace ClienteOAuthEmpleados.Repositories

{

    public class RepositoryEmpleados

    {

        private String url;

        private MediaTypeWithQualityHeaderValue header;

        public RepositoryEmpleados()

        {

            this.url = "https://apiempleadoscoreoauth.azurewebsites.net/";

            this.header =

                new MediaTypeWithQualityHeaderValue("application/json");

        }

        //METODO PARA VALIDAR USUARIOS NO DEVUELVE UN EMPLEADO

        //DEVOLVERA UN TOKEN...

        //EL SERVICIO NOS DIRA SI EXISTE O NO EXISTE

        public async Task<String> GetToken(String username, String password)

        {

            using (HttpClient client = new HttpClient())

            {

                client.BaseAddress = new Uri(this.url);

                client.DefaultRequestHeaders.Accept.Clear();

                client.DefaultRequestHeaders.Accept.Add(header);

                //CREAMOS EL MODELO LOGIN PARA EL API

                LoginModel login = new LoginModel();

                login.UserName = username;

                login.Password = password;

                //CONVERTIMOS EL LOGIN A JSON PARA EL SERVICIO API

                String json = JsonConvert.SerializeObject(login);

                StringContent content =

                    new StringContent(json, Encoding.UTF8, "application/json");

                String request = "Auth/Login";

                HttpResponseMessage response = await

                    client.PostAsync(request, content);

                if (response.IsSuccessStatusCode)

                {

                    String data = await

                        response.Content.ReadAsStringAsync();

                    JObject jobject = JObject.Parse(data);

                    String token = jobject.GetValue("response").ToString();

                    return token;

                }

                else

                {

                    return null;

                }

            }

        }

        //METODO PARA RESOLVER LAS PETICIONES API <T>

        //SIN SEGURIDAD (TOKEN)

        private async Task<T> CallApi<T>(String request)

        {

            using (HttpClient client = new HttpClient())

            {

                client.BaseAddress = new Uri(this.url);

                client.DefaultRequestHeaders.Accept.Clear();

                client.DefaultRequestHeaders.Accept.Add(header);

                HttpResponseMessage response =

                    await client.GetAsync(request);

                if (response.IsSuccessStatusCode)

                {

                    T data = await

                        response.Content.ReadAsAsync<T>();

                    return (T)Convert.ChangeType(data, typeof(T));

                }

                else

                {

                    return default(T);

                }

            }

        }

        //METODO PARA RESOLVER LAS PETICIONES API <T>

        //CON SEGURIDAD (TOKEN)

        private async Task<T> CallApi<T>(String request, String token)

        {

            using (HttpClient client = new HttpClient())

            {

                client.BaseAddress = new Uri(this.url);

                client.DefaultRequestHeaders.Accept.Clear();

                client.DefaultRequestHeaders.Accept.Add(header);

                client.DefaultRequestHeaders.Add("Authorization"

                    , "bearer " + token);

                HttpResponseMessage response =

                    await client.GetAsync(request);

                if (response.IsSuccessStatusCode)

                {

                    T data = await

                        response.Content.ReadAsAsync<T>();

                    return (T)Convert.ChangeType(data, typeof(T));

                }

                else

                {

                    return default(T);

                }

            }

        }

        //METODOS PARA LAS PETICIONES API Y NUESTRA APP CLIENTE MVC

        //SIN SEGURIDAD

        public async Task<Empleado> BuscarEmpleado(int empno)

        {

            Empleado emp = await

                this.CallApi<Empleado>("api/empleados/" + empno);

            return emp;

        }

        //CON SEGURIDAD

        public async Task<Empleado> PerfilEmpleado(String token)

        {

            Empleado emp = await

                this.CallApi<Empleado>("api/empleados/perfilempleado", token);

            return emp;

        }

        public async Task<List<Empleado>> GetSubordinados(String token)

        {

            List<Empleado> empleados = await

                this.CallApi<List<Empleado>>("api/empleados/subordinados", token);

            return empleados;

        }

    }

}

El siguiente paso es ir a la clase **Startup.cs** y montar la inyección de dependencias en el método **ConfigureServices()**

También vamos a habilitar las opciones para utilizar la sesión.

**STARTUP.CS**

public void ConfigureServices(IServiceCollection services)

{

services.AddTransient<IRepositoryApiEmpleados

, RepositoryApiEmpleados>();

services.AddControllersWithViews();

}

Debemos inyectar **RepositoryEmpleados** en el constructor de **ManageController** y en el constructor de **EmpleadosController**

RepositoryEmpleados repo;

public **ManageController**(RepositoryEmpleados repo)

{

this.repo = repo;

}

IRepositoryApiEmpleados repo;

public **EmpleadosController**(RepositoryEmpleados repo)

{

this.repo = repo;

}

A continuación, debemos implementar el método POST de Login en **ManageController**.

Dicho método comprobará nuestro usuario y password y creará un usuario **ClaimsPrincipal** y nos validará con una **Cookie** de autorización.

Para validarnos con la Cookie se realiza con un método asíncrono, por lo que debemos utilizar las palabras **async** y **await**. El método devolverá un **Task** con el **IActionResult**.

**MANAGECONTROLLER**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using ClasesSeguridadNuGet;

using ClienteOAuthEmpleados.Repositories;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Session;

namespace ClienteOAuthEmpleados.Controllers

{

    public class ManageController : Controller

    {

        RepositoryEmpleados repo;

        public ManageController(RepositoryEmpleados repo)

        {

            this.repo = repo;

        }

        public IActionResult Login()

        {

            return View();

        }

        [HttpPost]

        public async Task<IActionResult> Login(String username, String password)

        {

            //PARA VALIDAR USUARIOS, UTILIZAMOS EL TOKEN

            //SI EL TOKEN ES NULL, CREDENCIALES INCORRECTAS

            String token = await this.repo.GetToken(username, password);

            if (token == null)

            {

                ViewData["MENSAJE"] = "Usuario/Password incorrectos";

                return View();

            }

            else

            {

                //RECUPERAR AL USUARIO QUE SE HA VALIDADO

                Empleado empleado = await this.repo.PerfilEmpleado(token);

                //HABILITAMOS LA SEGURIDAD DE MVC CORE CON CLAIMS

                ClaimsIdentity identity =

                    new ClaimsIdentity(CookieAuthenticationDefaults

                    .AuthenticationScheme, ClaimTypes.Name

                    , ClaimTypes.Role);

                //ALMACENAMOS EL NUMERO DE EMPLEADO

                identity.AddClaim(new Claim(ClaimTypes.NameIdentifier

                    , empleado.IdEmpleado.ToString()));

                identity.AddClaim(new Claim(ClaimTypes.Name

                    , empleado.Apellido));

                identity.AddClaim(new Claim(ClaimTypes.Role

                    , empleado.Oficio));

                ClaimsPrincipal principal = new ClaimsPrincipal(identity);

                await HttpContext.SignInAsync(

                    CookieAuthenticationDefaults.AuthenticationScheme

                    , principal, new AuthenticationProperties

                    {

                        IsPersistent = true, ExpiresUtc =

                        DateTime.Now.AddMinutes(60)

                    });

                //DEBEMOS ALMACENAR EL TOKEN UNA VEZ QUE EL USUARIO YA EXISTE

                HttpContext.Session.SetString("TOKEN", token);

                return RedirectToAction("Index", "Empleados");

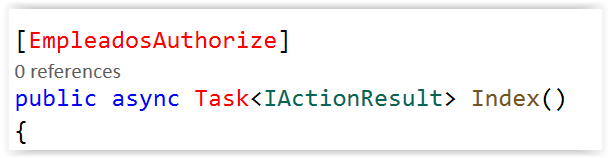
            }

        }

    }

}

Por último, debemos decorar nuestro método **Index** del controlador **EmpleadosController** para aplicar los filtros de seguridad.



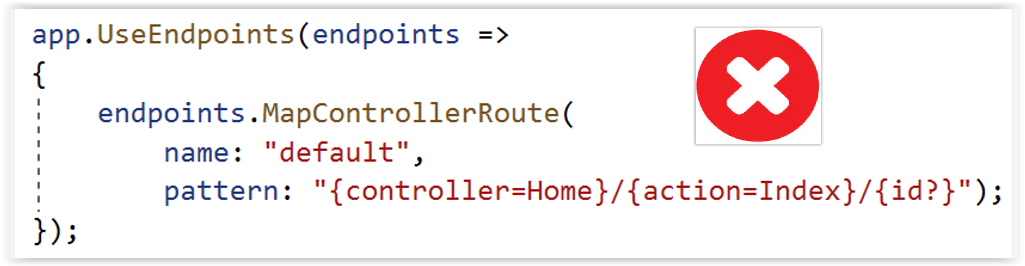
Ya tenemos montada toda la seguridad. Es el momento de configurar el servidor con la capa de Middleware para habilitar la seguridad con Cookies y Session.

Abrimos la clase **Startup.cs**

Lo primero es indicar en el método **Configure()** que nuestra aplicación utilizará autorizaciones mediante **app.UseAuthentication()**, también indicaremos que utilizaremos Session mediante **app.UseSession()**



También debemos cambiar **app.EndPoint** por **app.UseMvc** para poder aplicar la seguridad.





**STARTUP.CS METODO CONFIGURE()**

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

    if (env.IsDevelopment())

    {

        app.UseDeveloperExceptionPage();

    }

    else

    {

        app.UseExceptionHandler("/Home/Error");

        // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

        app.UseHsts();

    }

    app.UseHttpsRedirection();

    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseAuthentication();

    app.UseSession();

    app.UseMvc(routes =>

    {

        routes.MapRoute(

            name: "default",

            template: "{controller=Home}/{action=Index}/{id?}");

    });

}

El siguiente paso es configurar los servicios para configurar el uso de **Cookies** de seguridad y **Session** en el método **ConfigureServices()**, además que debemos indicar que deshabilitamos el método EndPoint.

**STARTUP.CS**

        public void ConfigureServices(IServiceCollection services)

    {

        services.AddTransient<RepositoryEmpleados>();

        services.AddDistributedMemoryCache();

        services.AddSession(options =>

        {

            options.IdleTimeout = TimeSpan.FromMinutes(60);

            options.Cookie.HttpOnly = true;

            options.Cookie.IsEssential = true;

        });

        services.AddAuthentication(options => {

            options.DefaultSignInScheme =

            CookieAuthenticationDefaults.AuthenticationScheme;

            options.DefaultChallengeScheme =

            CookieAuthenticationDefaults.AuthenticationScheme;

            options.DefaultAuthenticateScheme =

            CookieAuthenticationDefaults.AuthenticationScheme;

        }).AddCookie();

        services.AddMvc(options => options.EnableEndpointRouting = false).AddSessionStateTempDataProvider();

        services.AddSingleton<ITempDataProvider

, CookieTempDataProvider>();

        services.AddMvc().AddSessionStateTempDataProvider();

        services.AddSession();

        services.AddControllersWithViews();

    }

El siguiente paso será incluir la funcionalidad para poder Cerrar la sesión de un usuario.

Dentro de **Manage** creamos un action para que el usuario pueda cerrar la sesión.

**MANAGECONTROLLER**

public async Task<IActionResult> CerrarSesion()

{

    await HttpContext.SignOutAsync(

        CookieAuthenticationDefaults.AuthenticationScheme);

    if (HttpContext.Session.GetString("TOKEN") != null)

    {

        //HttpContext.Session.SetString("TOKEN", String.Empty);

        HttpContext.Session.Remove("TOKEN");

    }

    return RedirectToAction("Index", "Home");

}

Creamos una vista parcial para que nos muestre los datos del usuario y un acceso para cerrar sesión.

Sobre **Shared**, creamos una vista llamada **\_MenuEmpleado.cshtml**

**\_MENUEMPLEADO.CSHTML**

@{

ViewData["Title"] = "\_MenuEmpleado";

}

@if (Context.User.Identity.IsAuthenticated == false)

{

<li class="nav-item">

<**a** **asp-action**="Login" class="nav-link" **asp-controller**="Manage">

Log In

</**a**>

</li>

}

else

{

<li class="nav-item">

<**a** **asp-action**="CerrarSesion" class="nav-link" **asp-controller**="Manage">

Cerrar sesión <span style="color:white">@Context.User.Identity.Name</span>

</**a**>

</li>

}

Y en la página **\_Layout** incluimos el partial view del menú del usuario

**\_LAYOUT.CSHTML**

<**partial** **name**="\_MenuEmpleado"/>

Por último, damos funcionalidad a la zona de empleados.

En la página **Index**, el empleado visualizará su Perfil.

Tendremos otra página dónde el empleado podrá visualizar a sus subordinados.

**Nota:** Dichos métodos estarán protegidos con **[EmpleadosAuthorize]**

**EMPLEADOSCONTROLLER**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ClasesSeguridadNuGet;

using ClienteOAuthEmpleados.Filters;

using ClienteOAuthEmpleados.Repositories;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace ClienteOAuthEmpleados.Controllers

{

    public class EmpleadosController : Controller

    {

        RepositoryEmpleados repo;

        public EmpleadosController(RepositoryEmpleados repo)

        {

            this.repo = repo;

        }

        [EmpleadosAuthorize]

        public async Task<IActionResult> Index()

        {

            String token = HttpContext.Session.GetString("TOKEN");

            Empleado empleado = await

                this.repo.PerfilEmpleado(token);

            return View(empleado);

        }

        [EmpleadosAuthorize]

        public async Task<IActionResult> Subordinados()

        {

            String token = HttpContext.Session.GetString("TOKEN");

            List<Empleado> empleados = await

                this.repo.GetSubordinados(token);

            return View(empleados);

        }

        public async Task<IActionResult> DetallesEmpleado(int empno)

        {

            Empleado emp = await

                this.repo.BuscarEmpleado(empno);

            return View(emp);

        }

    }

}

Y creamos las vistas para su visualización.

**INDEX.CSHTML**

@model Empleado

@{

ViewData["Title"] = "Index";

}

<**a** **asp-action**="Subordinados" **asp-controller**="Empleados">

Mis empleados

</**a**>

<h1 style="color:blue">Perfil de @Model.Apellido</h1>

<dl class="row">

<dt class="col-sm-3">Oficio</dt>

<dd class="col-sm-9">@Model.Oficio</dd>

<dt class="col-sm-3">Salario</dt>

<dd class="col-sm-9">@Model.Salario</dd>

<dt class="col-sm-3">Director</dt>

<dd class="col-sm-9">@Model.Director</dd>

</dl>

**SUBORDINADOS.CSHTML**

@model List<Empleado>

@{

    ViewData["Title"] = "Subordinados";

}

<h1>Subordinados de @Context.User.Identity.Name</h1>

<a **asp-controller**="Empleados" **asp-action**="Index">

    Volver al perfil

</a>

<table class="table table-light">

    <tr>

        <th>APELLIDO</th>

        <th>OFICIO</th>

        <th></th>

    </tr>

    @foreach (Empleado emp in Model)

    {

        <tr>

            <td>@emp.Apellido</td>

            <td>@emp.Oficio</td>

            <td>

                <a **asp-controller**="Empleados"

**asp-action**="DetallesEmpleado"

**asp-route-empno**="@emp.IdEmpleado">

                    Detalles

                </a>

            </td>

        </tr>

    }

</table>

**DETALLESEMPLEADO.CSHTML**

@model Empleado

@{

    ViewData["Title"] = "DetallesEmpleado";

}

<h1>Detalles Empleado</h1>

<a **asp-controller**="Empleados" **asp-action**="Subordinados">

    Volver a subordinados

</a>

<dl>

    <dt>Apellido: @Model.Apellido</dt>

    <dd>Oficio: @Model.Oficio</dd>

    <dd>Salario: @Model.Salario</dd>

    <dd>Director: @Model.Director</dd>

</dl>