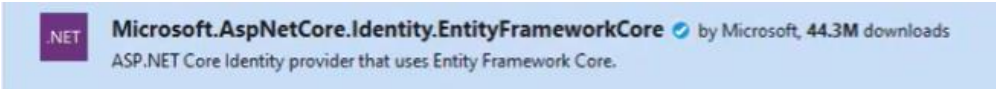
JWT (JSON Web Token) V.126,127,128 y 129

- Compuesto de tres partes: Cabecera (Algoritmo a utilizar), datos(información del usuario) y la firma(Llave secreta).

PARTE BACKEND

1. Instalar Microsoft.AspNetCore.Entity.FrameworkCore



2. Vamos al ApplicationDbContext y cambiamos DbContext por IdentityDbContext en la parte que se hereda la clase. Y con esto ya se configuran solas las tablas de identity

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
public class ApplicationDbContext : IdentityDbContext
```

3. Importante que dentro de la clase anterior tengamos una función OnModelCreating y adentro base. OnModelCreating(modelBuilder);

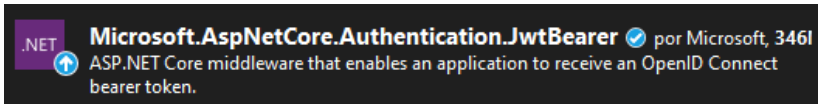
```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{

    base.OnModelCreating(modelBuilder);
}
```

4. Vamos al package manager console y digitar:
   Add-Migration SistemaDeUsuarios -> Crea un clase con toda la configuración de la tabla que se va a crear.
   Update-Database -> Crea la base de datos en Sql con la tabla Estudiantes.

   NOTA: Tener instalado Microsoft.EntityFrameworkCore.Tools
   Hacer esto después de haber creado todas las tablas de la app.

5. Instalar el siguiente paquete.



6. Agregar las siguientes líneas de configuración en el archivo program.cs o startup.cs

```
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();


builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(opciones =>
    opciones.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(builder.Configuration["llavejwt"])),
        ClockSkew = TimeSpan.Zero
    });
```

7. Vamos al archivo appsettings.json y agregamos la variable llavejwt con un string muy largo por temas de seguridad

```
  },
  "llavejwt": "LFSOQMQPASDNIKDSPIQNWAASDKDSASD897234SJDFNAOIUASDKIJASSDFSDFSDFSFDSDFDSFJDJPJAS"
}
```

8. Vamos a Crear CuentasController

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace PeliculasAPI.Controllers
{
    [Route("api/cuentas")]
    [ApiController]
    public class CuentasController: ControllerBase
    {

    }
}
```

9. Creamos los DTO de CredencialesUsuario y RespuestaAutenticacion

```
using System.ComponentModel.DataAnnotations;

namespace Industria.DTOs
{
    3 referencias
    public class CredencialesUsuario
    {
        [Required]
        [EmailAddress]
        5 referencias
        public string Email { get; set; }
        [Required]
        2 referencias
        public string Password { get; set; }
    }
}
```

```
namespace Industria.DTOs
{

    public class RespuestaAutenticacion
    {

        public string Token { get; set; }

        public DateTime Expiracion { get; set;
    }
}
```

10. Agregamos el siguiente código en CuentasController para crear y loguear usuarios, además construir el token.

```
using backEnd.DTOs;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
```

```
namespace backEnd.Controllers
{
    [Route("api/cuentas")]
    [ApiController]
    1 referencia
    public class CuentasController : ControllerBase
    {
        private readonly UserManager<IdentityUser> userManager;
        private readonly IConfiguration configuration;
        private readonly SignInManager<IdentityUser> signInManager;

        0 referencias
        public CuentasController(UserManager<IdentityUser> userManager,
            IConfiguration configuration, SignInManager<IdentityUser> signInManager)
        {
            this.userManager = userManager;
            this.configuration = configuration;
            this.signInManager = signInManager;
        }

        [HttpPost("crear")]

        0 referencias
        public async Task<ActionResult<RespuestaAutenticacion>> Crear([FromBody] CredencialesUsuario credenciales)
        {


            var usuario = new IdentityUser { UserName = credenciales.Email, Email = credenciales.Email };
            var resultado = await userManager.CreateAsync(usuario, credenciales.Password);
            if (resultado.Succeeded)
            {
                return await ConstruirToken(credenciales);
            }
            else
            {
                return BadRequest(resultado.Errors);
            }

        }
```

```
        [HttpPost("login")]
        0 referencias
        public async Task<ActionResult<RespuestaAutenticacion>> Login([FromBody] CredencialesUsuario credenciales)
        {
            var resultado = await signInManager.PasswordSignInAsync(credenciales.Email,credenciales.Password,
                isPersistent: false, lockoutOnFailure: false);
            if(resultado.Succeeded)
            {
                return await ConstruirToken(credenciales);
            }
            else
            {
                return BadRequest("Login incorrecto");
            }

        }

        2 referencias
        private async Task<RespuestaAutenticacion> ConstruirToken(CredencialesUsuario credenciales)
        {
            var claims = new List<Claim>()
            {
                new Claim("email", credenciales.Email)
            };
            var usuario = await userManager.FindByEmailAsync(credenciales.Email);
            var claimsDB = await userManager.GetClaimsAsync(usuario);
            claims.AddRange(claimsDB);
            var llave = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["llavejwt"]));
            var creds = new SigningCredentials(llave, SecurityAlgorithms.HmacSha256);
            var expiracion = DateTime.UtcNow.AddYears(1);

            var token = new JwtSecurityToken(issuer: null, audience:null, claims:null, expires: expiracion,
                signingCredentials: creds);

            return new RespuestaAutenticacion()
            {
                Token = new JwtSecurityTokenHandler().WriteToken(token),
                Expiracion = expiracion
            };
        }
    }
}
```

11. Para Bloquear los endpoint de nuestros controller debemos adicionar:

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Authentication.JwtBearer;

namespace Industria.Controllers
{
    [ApiController]
    [Route("api/empleados")]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

PARTE FRONTEND

1. Creamos un archivo seguridad.ts para crear los DTO para interpretar los llamados Http

```ts
export interface credencialesUsuario{
    email: string;
    password: string;
}

export interface respuestaAutenticacion{
    token: string;
    expiracion: Date;
}
```

2. Creamos el servicio seguridad: ng g s seguridad/seguridad --skip-tests=true
   Con los métodos de registrar, login, etc…

```ts
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from 'src/environments/environments';
import { credencialesUsuario, respuestaAutenticacion } from './registro/seguridad';
import { Observable } from 'rxjs';
```

```ts
@Injectable({
  providedIn: 'root'
})
export class SeguridadService {

  constructor(private httpClient : HttpClient) { }
  apiURL = environment.apiURL + 'cuentas';

  registrar(credenciales: credencialesUsuario) : Observable<respuestaAutenticacion>{
    return this.httpClient.post<respuestaAutenticacion>(this.apiURL+ '/crear', credenciales);
  }

  login(credenciales: credencialesUsuario) : Observable<respuestaAutenticacion>{
    return this.httpClient.post<respuestaAutenticacion>(this.apiURL+ '/login', credenciales);
  }

  private readonly llaveToken ='token';
  private readonly llaveExpiracion ='token-expiracion';

  estaLogueado(): boolean{
    const token = localStorage.getItem(this.llaveToken);
    if(!token)
    return false;
    const expiracion =localStorage.getItem(this.llaveExpiracion);
    const expiracionFecha = new Date(expiracion);
    if(expiracionFecha <= new Date()){
      this.logout();
      return false;
    }
    return true;
  }

  logout(){
    localStorage.removeItem(this.llaveToken);
    localStorage.removeItem(this.llaveExpiracion);
  }

  guardarToken(respuestaAutenticacion: respuestaAutenticacion){
    localStorage.setItem(this.llaveToken, respuestaAutenticacion.token);
    localStorage.setItem(this.llaveExpiracion, respuestaAutenticacion.expiracion.toString());
  }
}
```

3. Creamos el componente registro: ng g c seguridad/registro --skip-tests=true
4. El archivo html del componente registro lo configuramos así:

```
<h2>REGISTRO ESTUDIANTE</h2>
<form [formGroup]="form">

    <mat-form-field appearance="outline">
      <mat-label>Email</mat-label>
      <input formControlName="email" matInput />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Password</mat-label>
      <input formControlName="password" matInput />
    </mat-form-field>
    <div>
      <button mat-flat-button color="primary" (click)="submit()">
        Guardar Cambios
      </button>
      <a mat-stroked-button routerLink="/listado">
        Cancelar
      </a>
    </div>

  </form>
```

5. El archivo .ts del componente registro lo configuramos así:

```
import { Component, OnInit} from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { SeguridadService } from '../seguridad.service';
import { credencialesUsuario } from '../seguridad';
import { Router } from '@angular/router';

@Component({
  selector: 'app-registro',
  templateUrl: './registro.component.html',
  styleUrls: ['./registro.component.css']
})
export class RegistroComponent implements OnInit  {

  constructor(private formBuilder: FormBuilder, private seguridadService: SeguridadService,
    private router: Router){}

  form: FormGroup;

  ngOnInit(): void {
    this.form = this.formBuilder.group({
      email: '',
      password: ''
    })
  }
  submit()
  {
    this.registrar(this.form.value);
  }
  registrar(credenciales: credencialesUsuario){
    this.seguridadService.registrar(credenciales).subscribe(respuesta=>{
      this.seguridadService.guardarToken(respuesta);
      this.router.navigate(['']);
    })
  }
}
```

6. Creamos el componente login: ng g c seguridad/login --skip-tests=true
7. El archivo html del componente login lo configuramos así:

```html
<h2>LOGIN ESTUDIANTE</h2>
<form [formGroup]="form">

    <mat-form-field appearance="outline">
      <mat-label>Email</mat-label>
      <input formControlName="email" matInput />
    </mat-form-field>
    <mat-form-field appearance="outline">
      <mat-label>Password</mat-label>
      <input formControlName="password" matInput />
    </mat-form-field>
    <div>
      <button mat-flat-button color="primary" (click)="submit()">
        Guardar Cambios
      </button>
      <a mat-stroked-button routerLink="/listado">
        Cancelar
      </a>
    </div>

  </form>
```

8. El archivo .ts del componente login lo configuramos así:

```typescript
import { Component ,OnInit} from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { SeguridadService } from '../seguridad.service';
import { credencialesUsuario } from '../registro/seguridad';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit{

  constructor(private formBuilder:FormBuilder, private seguridadService: SeguridadService,
    private router: Router){}

  form : FormGroup;

  ngOnInit(): void {
    this.form = this.formBuilder.group({
      email:'',
      password:''
    });
  }

  submit()
  {
    this.login(this.form.value);
  }

  login(credenciales: credencialesUsuario){
    this.seguridadService.login(credenciales).subscribe(respuesta=>{
      this.seguridadService.guardarToken(respuesta);
      this.router.navigate(['/listado'])
    })
  }
}
```

9. Con esta configuración ya podemos loguearnos y registrarnos.
10. Ahora tenemos que implementar los métodos para poder navegar en el frontEnd cuando se necesita estar logueado. NOTA: Ya se hizo arriba pero se aclara su funcionamiento.

```
private readonly llaveToken ='token';
private readonly llaveExpiracion ='token-expiracion';

estaLogueado(): boolean{
  const token = localStorage.getItem(this.llaveToken);
  if(!token)
  return false;
  const expiracion =localStorage.getItem(this.llaveExpiracion);
  const expiracionFecha = new Date(expiracion);
  if(expiracionFecha <= new Date()){
    this.logout();
    return false;
  }
  return true;
}

logout(){
  localStorage.removeItem(this.llaveToken);
  localStorage.removeItem(this.llaveExpiracion);
}

guardarToken(respuestaAutenticacion: respuestaAutenticacion){
  localStorage.setItem(this.llaveToken, respuestaAutenticacion.token);
  localStorage.setItem(this.llaveExpiracion, respuestaAutenticacion.expiracion.toString());
}
```

11. Más adelante buscar la configuración para usuarios con roles

PROTEGIENDO RUTAS CON CanActivate

1. Vamos al terminal de visual code y digitamos:
   ng g g es-admin --skip-tests=true

```
 <i> to invert selection, and <enter> to proceed)
>(*) CanActivate
 ( ) CanActivateChild
 ( ) CanDeactivate
 ( ) CanLoad
 ( ) CanMatch
```

2. En el archivo es-admin.guard.ts implementamos:

```
import { Injectable } from '@angular/core';
import {
  ActivatedRouteSnapshot,
  ~~CanActivate~~,
  Router,
  RouterStateSnapshot,
  UrlTree,
} from '@angular/router';
import { Observable } from 'rxjs';
import { SeguridadService } from './seguridad/seguridad.service';

@Injectable({
  providedIn: 'root',
})
export class EsAdminGuard implements ~~CanActivate~~ {
  constructor(private seguridadService: SeguridadService, private router : Router) {}

  canActivate(route: ActivatedRouteSnapshot,state: RouterStateSnapshot):
    | Observable<boolean | UrlTree>
    | Promise<boolean | UrlTree>
    | boolean
    | UrlTree {

    if (this.seguridadService.estaLogueado() === true){
      return true;
    }
    this.router.navigate(['/login']);
    return false;
  }
}
```

3. Vamos al archivo app-routing.module.ts y adicionamos canActivate para las rutas que queremos proteger.

```
const routes: Routes = [
  {path:'listado', component: ListadoEstudiantesComponent , canActivate: [EsAdminGuard]},
  {path:'crear', component: CrearEstudianteComponent , canActivate: [EsAdminGuard]},
  {path:'editar/:id', component: EditarEstudianteComponent , canActivate: [EsAdminGuard]},
  {path:'registro', component: RegistroComponent },
  {path:'login', component: LoginComponent }
];
```

CONFIGURACION PARA USUARIOS CON ROLES

PARTE BACKEND

1. Vamos al program.cs y agregamos

```
builder.Services.AddAuthorization(opciones =>
{
    opciones.AddPolicy("EsAdmin", policy => policy.RequireClaim("role", "admin"));
});
```

2.

PARTE FRONTEND

CONFIGURACION TEMPORAL

```
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from 'src/environment/environment.prod';
import { credencialesUsuario, respuestaAutenticacion, usuarioDTO } from './seguridad';
import { Observable } from 'rxjs';
import { Router } from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class SeguridadService {

  constructor(private httpClient : HttpClient, private router: Router) { }

  private apiURL = environment.apiURL + 'cuentas';
  private readonly llaveToken = 'token';
  private readonly llaveExpiracion = 'token-expiracion';
  private readonly campoRol = 'role';

  obtenerUsuarios(pagina:number, recordsPorPagina: number): Observable<any>{
    let params = new HttpParams();
    params = params.append('pagina',pagina.toString());
    params = params.append('recordsPorPagina', recordsPorPagina.toString());
    return this.httpClient.get<usuarioDTO>(`${this.apiURL}/listadousuarios`,
    {observe: 'response', params})
  }

  hacerAdmin (usuarioId: string){
    const headers= new HttpHeaders('Content-type: application/json');
    return this.httpClient.post(`${this.apiURL}/hacerAdmin`, JSON.stringify(usuarioId), {headers});
  }

  removerAdmin (usuarioId: string){
    const headers= new HttpHeaders('Content-type: application/json');
    return this.httpClient.post(`${this.apiURL}/removerAdmin`, JSON.stringify(usuarioId), {headers})
  }
```

```
  estaLogueado(): boolean{

    const token = localStorage.getItem(this.llaveToken);
    if(!token){
      return false;
    }

    const expiracion = localStorage.getItem(this.llaveExpiracion);
    const expiracionFecha = new Date(expiracion);

    if(expiracionFecha <= new Date()){
      this.logout();
      return false;
    }

    return true;
  }

  logout(){
    localStorage.removeItem(this.llaveToken);
    localStorage.removeItem(this.llaveExpiracion);
    this.router.navigate(['/login']);

  }

  obtenerRol(): string{
    return this.obtenerCampoJWT(this.campoRol);
  }
```

```
obtenerCampoJWT(campo: string) : string{
  const token = localStorage.getItem(this.llaveToken);
  if(!token){return '';}
  var dataToken = JSON.parse(atob(token.split('.')[1]));
  return dataToken[campo];
}

registrar(credenciales: credencialesUsuario): Observable<respuestaAutenticacion>
{
    return this.httpClient.post<respuestaAutenticacion>(this.apiURL + '/crear', credenciales);
}

login(credenciales: credencialesUsuario): Observable<respuestaAutenticacion>
{
    return this.httpClient.post<respuestaAutenticacion>(this.apiURL + '/login', credenciales);
}

guardarToken(respuestaAutenticacion: respuestaAutenticacion)
{
    localStorage.setItem(this.llaveToken, respuestaAutenticacion.token);
    localStorage.setItem(this.llaveExpiracion, respuestaAutenticacion.expiracion.toString());
}


obtenerToken(){
  return localStorage.getItem(this.llaveToken);
}

}
```

MENU

```
<mat-toolbar color="primary">
<span>
    <a mat-button routerLink="">
        <mat-icon>local_library</mat-icon> Empleados
    </a>
</span>
<div>
    <a mat-button routerLink="guardar">Guardar</a>
</div>
<div>
    <a mat-button routerLink="editar">Editar</a>
</div>
<div>
    <a mat-button (click)="seguridadService.logout()">Logout</a>
</div>
<div>
    <a mat-button routerLink="login">Login</a>
</div>
<div>
    <a mat-button routerLink="registro">Registro</a>
</div>

</mat-toolbar>
```