

# Documentación del Proyecto: WebInstituto

Moisés José Rico Hernández



# Contenido

- 1. Descripción del Proyecto ..... 3
- 2. Estructura del Proyecto..... 3
- 3. Capas y Responsabilidades ..... 4
- 4. Configuraciones y Dependencias ..... 5
- 5. Puntos de Entrada y Navegación ..... 5
- 6. Migraciones y Base de Datos ..... 5
- 7. Servicios y Seguridad ..... 6
- 8. Notas Adicionales..... 6

## 1. Descripción del Proyecto

WebInstituto es una aplicación web ASP.NET MVC que permite la gestión de un instituto, incluyendo asignaturas, horarios, y personas (Alumnos/Profesores). Está desarrollada con C#, utiliza Entity Framework para acceso a datos y emplea SQLite como base de datos.

## 2. Estructura del Proyecto

WebInstituto/

|

|— BBDD/

|

|— Properties/

|

|— Repositorios/

|

|— Services/

|

|— ViewModels/

|

|— Views/

|

|— wwwroot/

|

|— Configuración y Archivos de Arranque

### 3. Capas y Responsabilidades

#### **Controllers/**

Controladores que manejan las peticiones del usuario:

- AsignaturasController.cs
- HorariosController.cs
- LoginController.cs
- PersonasController.cs

#### **Models/**

Modelos de entidad que representan los datos principales:

- Asignatura.cs, Persona.cs, Horario.cs, etc.
- Relaciones: AsignaturaPersona.cs
- Enumeraciones en subcarpeta Enum/.

#### **Repositorios/**

Capa de acceso a datos:

- RepoAsignaturas.cs
- RepoHorario.cs
- RepoMatricular.cs
- RepoPersonas.cs

#### **Services/**

Lógica de negocio y servicios:

- SeguridadService.cs – seguridad/autenticación.
- SessionService.cs – gestión de sesión del usuario.
- GetPath.cs - obtiene la ruta de la base de datos.

### **ViewModels/**

Modelos de presentación utilizados por las vistas.

### **Views/**

Vistas Razor divididas por entidades:

- Asignaturas/
- Horarios/
- Personas/
- Shared/ – vistas comunes como \_Layout, \_ViewStart.

### **Migrations/**

Archivos generados por Entity Framework para controlar cambios en la base de datos.

## 4. Configuraciones y Dependencias

**Base de Datos:** SQLite (DBSqlite.cs)

## 5. Puntos de Entrada y Navegación

**Program.cs:** Configura y arranca la aplicación.

**DBSqlite.cs:** Inicializa y configura la base de datos SQLite.

## 6. Migraciones y Base de Datos

20250403080454\_InitialCreate.cs que se genera automáticamente con EF

## 7. Servicios y Seguridad

**SeguridadService.cs:** Manejo de autenticación/autorización.

**SessionService.cs:** Gestión de sesión de usuarios logueados.

## 8. Notas Adicionales

### Usuarios de prueba:

Mail	Contraseña	Profesor
carlos@gmail.com	1234	Sí
moi@gmial.com	1234	Sí
juan@gmail.com	1234	Sí
pepe@gmail.com	1234	No
marcos@gmail.com	1234	No

### Código a resaltar:

Services:

**SeguridadService** es una clase estática que proporciona funciones relacionadas con la seguridad de contraseñas. Utiliza el algoritmo BCrypt para el almacenamiento seguro de contraseñas y su posterior verificación.

```

public static class SeguridadService
{
    // Genera un hash de la contraseña usando BCrypt
    1 referencia
    public static string HashPassword(string password)
    {
        return BCrypt.Net.BCrypt.HashPassword(password, workFactor: 12);
    }

    // Verifica si la contraseña ingresada coincide con el hash almacenado
    1 referencia
    public static bool VerifyPassword(string password, string hash)
    {
        return BCrypt.Net.BCrypt.Verify(password, hash);
    }
}

```

**SessionService** es un servicio que encapsula la lógica de manejo de sesión del usuario dentro de una aplicación ASP.NET Core. Proporciona métodos para iniciar sesión, cerrar sesión, verificar el estado de la sesión, y obtener datos del usuario autenticado como su correo electrónico y rol.

Login (Guarda datos relevantes del usuario en sesión)

```

public void Login(Persona persona)
{
    _httpContextAccessor.HttpContext.Session.SetString("email", persona.Mail);
    _httpContextAccessor.HttpContext.Session.SetInt32("isTeacher", persona.Teacher.Value);
}

```

Logout (Limpia completamente la sesión del usuario actual)

```

public void Logout()
{
    _httpContextAccessor.HttpContext.Session.Clear();
}

```

EstaLogeado (Verifica si el usuario ha iniciado sesión)

```

public bool EstaLogeado()
{
    var email = _httpContextAccessor.HttpContext.Session.GetString("email");
    return !string.IsNullOrEmpty(email);
}

```

**GetPath**, para construir dinámicamente la ruta del archivo SQLite, sin tener que escribir rutas absolutas que cambien según la máquina.

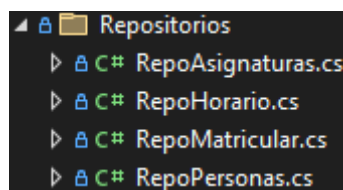
```
public static string GetDatabasePath()
{
    var projectRoot = Directory.GetParent(AppDomain.CurrentDomain.BaseDirectory)
        ?.Parent?.Parent?.Parent?.FullName;

    var path = Path.Combine(projectRoot, "BBDD", "institutoEF.sqlite");

    return path ?? throw new Exception("No se pudo determinar la ruta de la base de datos.");
}
```

## Repositorios:

- **Encapsular el acceso a la base de datos** (evitas escribir consultas directamente en controladores o servicios).
- **Organizar mejor el código** (separan la lógica de datos del resto de la app).



## Mapeo:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    // Asignatura - Horario N:1
    modelBuilder.Entity<Asignatura>().HasMany(a => a.Horarios).WithOne(h => h.Asignatura).HasForeignKey(h=>h.AsignaturaId);

    // Asignatura - Persona (Profesor) :N
    modelBuilder.Entity<Persona>().HasMany(a => a.AsignaturasImpartidas).WithOne(h => h.Profesor).HasForeignKey(h => h.IdProfesor);

    modelBuilder.Entity<AsignaturaPersona>().HasOne(p => p.Alumno).WithMany(a => a.AsignaturasAlumno).HasForeignKey(p=>p.IdAlumno);
    modelBuilder.Entity<AsignaturaPersona>().HasOne(a => a.Asignatura).WithMany(p => p.Alumnos).HasForeignKey(p => p.IdAsignatura);
}
```

## Implementaciones futuras:

- Profesor pueda eliminar alumnos de las asignaturas.
- Gestionar las alertas de error.
- Mejoras de interfaz