

## Prácticas

### Tema 7 – Desarrollo de aplicaciones Web MVC (II)

<b>Objetivos</b>	<p>En esta práctica, se aborda la realización de tareas relacionadas con:</p> <ul style="list-style-type: none"><li>• Crear aplicaciones Web basadas en ASP.NET Core MVC empleando el enfoque <i>Code First</i> para el enlace a datos.</li><li>• Crear controladores y vistas que implementen procesamientos de tipo CRUD (<i>Create, Read, Update and Delete</i>),</li><li>• Crear controladores y vistas que implementen procesamientos específicos que cumplan con requisitos de desarrollo concretos.</li><li>• Utilizar <i>Entity Framework Core (EF Core)</i> como una herramienta ORM que facilita el acceso los datos que maneja la aplicación Web a través de un modelo conceptual expresado mediante un conjunto de objetos.</li><li>• Utilizar las funciones de la API de EF Core y las consultas <i>LINQ To Entities</i> para acceder a los datos almacenados.</li><li>• Conocer y utilizar el sistema predeterminado para la gestión de acceso de usuarios a las aplicaciones Web denominado <i>ASP.NET Core Identity</i>.</li></ul>
<b>Conocimientos previos</b>	<p>Para realizar esta práctica, es necesario tener conocimientos sobre:</p> <ul style="list-style-type: none"><li>• Características de las aplicaciones Web Interactivas.</li><li>• Modelos de arquitectura de software. Características del modelo de arquitectura de software Modelo-Vista-Controlador (MVC).</li><li>• Lenguaje de programación Microsoft Visual C#.</li><li>• Bases de datos relacionales.</li><li>• Crear modelos partiendo del código (<i>Code First</i>).</li></ul>
<b>Escenario</b>	<p>En la práctica del tema anterior se realizaron diversos ejercicios prácticos cuyo objetivo principal es comprender los aspectos más básicos del desarrollo de aplicaciones Web basadas en la arquitectura del software Modelo-Vista-Controlador (MVC), utilizando para ello la tecnología de desarrollo ASP.NET Core MVC. En este tema se avanza y profundiza el uso del código que se emplea junto con la tecnología de desarrollo ASP.NET Core MVC para poder incorporar a las aplicaciones Web de procesamientos con funcionalidades específicas y adaptadas a las necesidades específicas de los usuarios. Se presentan diversos ejercicios que van dirigidos a profundizar, comprender, experimentar e indagar en algunos de los aspectos prácticos más específicos del uso del código para abordar el desarrollo de aplicaciones Web basadas en ASP.NET Core MVC.</p>

## Ejercicio 1

# Creación de una aplicación Web basada en ASP.NET Core MVC empleando el enfoque *Code First* para el enlace a datos

En este ejercicio se creará una nueva aplicación Web basada en ASP.NET Core MVC, denominada *MvcSoporte*, que facilitará el servicio de soporte técnico informático de una empresa u organización. Esta aplicación Web incluirá funcionalidades para gestionar la información de los avisos de reparación, configuración e instalación de los equipos informáticos de una empresa u organización que realizan sus empleados. En este ejercicio se aborda la creación de aplicaciones Web basadas en ASP.NET Core MVC empleando un procedimiento de trabajo basado en el enfoque *Code First* para el enlace a datos.

### Crear una nueva Aplicación Web de ASP.NET Core MVC

En primer lugar, crear una nueva aplicación Web basada en ASP.NET Core MVC utilizando la plantilla de proyecto correspondiente de Visual Studio, tal como se realizó en la práctica anterior. Tanto la Solución de Visual Studio como el Proyecto de ASP.NET Core MVC se denominarán *MvcSoporte*. Es importante tener en cuenta que durante la creación de la aplicación Web, se debe seleccionar la opción **Cuentas individuales** para establecer el tipo de **Autenticación** de la aplicación Web.

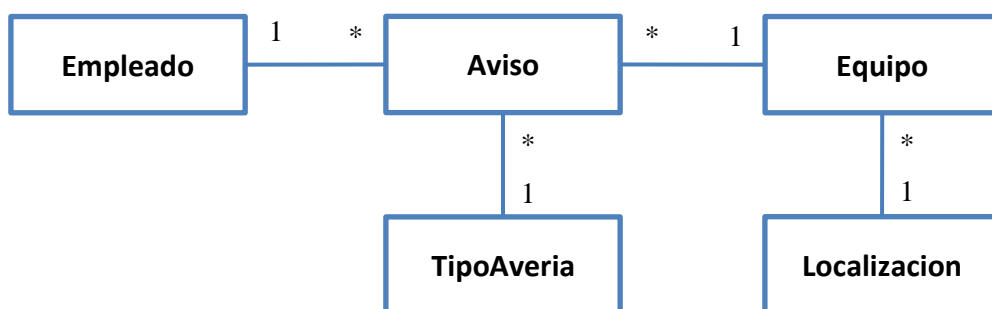
### Ejecutar y adaptar el Proyecto de ASP.NET Core MVC creado

Una vez creado el Proyecto de ASP.NET Core MVC ya puede solicitarse su ejecución, para ello basta con Iniciar la depuración. Si se desea, aunque puede realizarse más tarde, se puede adaptar la plantilla de diseño de vistas *\_Layout.cshtml* ubicada en la carpeta */Views/Shared*, y la hoja de estilo externa *site.css* ubicada en la carpeta */wwwroot/css* para adecuar la apariencia visual de la aplicación Web.

### Crear el modelo

Como paso previo a la creación del modelo, comprobar que están instalados los paquetes de *Entity Framework Core* en la Solución. Para ello, utilizar la herramienta **Administrador de paquetes NuGet** que, además, permite instalar los paquetes que sean necesarios para la solución de forma desatendida.

A continuación, se va a crear el modelo de la aplicación Web *MvcSoporte*, que está formado por cinco entidades de datos relacionadas entre sí, tal como se muestra en la siguiente ilustración.



Para crear el modelo de la aplicación Web basada en ASP.NET Core MVC que se está desarrollando mediante un enfoque de acceso a datos *Code First*, realizar las siguientes acciones:

1. Se utilizarán cinco clases para poder representar cada una de las **entidades de datos** del modelo. Para crear estas clases de datos, realizar las siguientes acciones:
  - a. En la carpeta */Models*, crear un archivo de clase denominado *Empleado.cs*. Para ello, desde el **Explorador de soluciones**, hacer clic en el botón derecho sobre la carpeta */Models*, desplegar la opción **Agregar** y seleccionar la opción **Clase...** A continuación, introducir el nombre y añadir el siguiente código al archivo de clase creado:

```
public class Empleado
{
    public int Id { get; set; }
    [Required(ErrorMessage = "El nombre del empleado es un campo requerido.")]
    public string? Nombre { get; set; }
    [Display(Name = "Correo electrónico")]
    [EmailAddress(ErrorMessage = "Dirección de correo electrónico invalida")]
    public string? Email { get; set; }
    [Display(Name = "Teléfono")]
    public string? Telefono { get; set; }
    [Display(Name = "Fecha de nacimiento")]
    [DataType(DataType.Date)]
    public DateTime? FechaNacimiento { get; set; }

    public ICollection<Aviso>? Avisos { get; set; }
}
```

La propiedad *Id* de la clase de datos *Empleado* se convertirá en la columna clave principal de la tabla correspondiente. Por convención, *Entity Framework Core* interpreta una propiedad denominada **ID** o **Id** como clave primaria. La propiedad *Avisos* es una propiedad de navegación que permite expresar una relación con otra entidad de datos. En este caso, la propiedad *Avisos* mantendrá una lista o colección de entidades que contiene el conjunto de entidades de tipo *Aviso* con las cuales una entidad *Empleado* se relaciona. De manera que, si un empleado crea o realiza varios avisos, entonces la propiedad *Avisos* mantendrá la colección de entidades de tipo *Aviso* correspondientes a ese *Empleado*.

- b. En la carpeta */Models*, crear un archivo de clase denominado *TipoAveria.cs* y añadir el siguiente código:

```
public class TipoAveria
{
    public int Id { get; set; }
    [Display(Name = "Descripción")]
    [Required(ErrorMessage = "La descripción del tipo de avería es un campo requerido.")]
    public string? Descripcion { get; set; }

    public ICollection<Aviso>? Avisos { get; set; }
}
```

- c. En la carpeta */Models*, crear un archivo de clase denominado *Localizacion.cs* y añadir el siguiente código:

```
public class Localizacion
{
    public int Id { get; set; }
    [Display(Name = "Descripción")]
    [Required(ErrorMessage = "La descripción de la localización es un campo requerido.")]
    public string? Descripcion { get; set; }

    public ICollection<Equipo>? Equipos { get; set; }
}
```

- d. En la carpeta */Models*, crear un archivo de clase denominado *Equipo.cs* y añadir el siguiente código:

```
public class Equipo
{
    public int Id { get; set; }
    [Display(Name = "Código equipo")]
    [Required(ErrorMessage = "El código del equipo es un campo requerido.")]
    public string? CodigoEquipo { get; set; }
    [Required(ErrorMessage = "La marca es un campo requerido.")]
    public string? Marca { get; set; }
    [Required(ErrorMessage = "El modelo es un campo requerido.")]
    public string? Modelo { get; set; }
    [Display(Name = "Número de serie")]
    [Required(ErrorMessage = "El número de serie es un campo requerido.")]
    public string? NumeroSerie { get; set; }
    [Display(Name = "Características técnicas")]
    public string? Caracteristicas { get; set; }
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Precio { get; set; }
    [Display(Name = "Precio")]
    [RegularExpression(@"^[0-9]{1,15}([0-9]{1,2}(\.[0-9]{1,2})?)?$",
        ErrorMessage = "El valor introducido debe ser de tipo monetario.")]
    [Required(ErrorMessage = "El precio es un campo requerido")]
    public string PrecioCadena
    {
        get {
            return Convert.ToString(Precio).Replace('.', ',');
        }
        set {
            Precio = Convert.ToDecimal(value.Replace(',', '.'));
        }
    }
    [Display(Name = "Fecha de compra")]
    [Required(ErrorMessage = "La fecha de compra es un campo requerido.")]
    [DataType(DataType.Date)]
    public DateTime FechaCompra { get; set; }
    [Display(Name = "Fecha de baja")]
    [DataType(DataType.Date)]
    public DateTime? FechaBaja { get; set; }
    [Display(Name = "Localización")]
    public int LocalizacionId { get; set; }

    public ICollection<Aviso>? Avisos { get; set; }

    public Localizacion? Localizacion { get; set; }
}
```

En el código anterior, la propiedad *LocalizacionId* será una clave externa. La propiedad de navegación correspondiente es *Localizacion*. De este modo, una entidad de tipo *Equipo* se relaciona con una entidad de tipo *Localizacion*. *Entity Framework Core* interpreta una propiedad de clave externa si se denomina según el siguiente formato: *<nombre de la entidad relacionada><nombre de la propiedad de clave principal>*. Así, por ejemplo, la propiedad de clave externa *LocalizacionId* en la clase *Equipo* se ha formado uniendo los nombres de la entidad relacionada, que es *Localizacion*, y de la propiedad clave principal en la entidad *Localizacion*, que es *Id*. En el código anterior también puede apreciarse que se especifican las propiedades *Precio* y *PrecioCadena* de tipo *decimal* y de tipo *string*, respectivamente. El uso combinado de estas dos propiedades permite obtener, al mismo tiempo, una presentación visual adecuada y un tratamiento correcto de los valores monetarios, cuando la configuración cultural de la aplicación Web es distinta de la inglesa (en-gb) o americana (en-us). Este recurso suele emplearse cuando se desea utilizar una configuración cultural en español (es-es), en la cual los valores monetarios se presentan con la coma como punto decimal y el punto se utiliza como separador de miles.

- e. En la carpeta */Models*, crear un archivo de clase denominado *Aviso.cs* y añadir el siguiente código:

```
public class Aviso
{
    public int Id { get; set; }
    [Display(Name = "Descripción del problema")]
    [Required(ErrorMessage = "La descripción del aviso es un campo requerido.")]
    public string? Descripcion { get; set; }
    [Display(Name = "Fecha de aviso")]
    [Required(ErrorMessage = "La fecha en que se realiza el aviso es un campo requerido.")]
    [DataType(DataType.Date)]
    public DateTime FechaAviso { get; set; }
    [Display(Name = "Fecha de cierre")]
    [DataType(DataType.Date)]
    public DateTime? FechaCierre { get; set; }
    public string? Observaciones { get; set; }
    [Display(Name = "Empleado")]
    public int EmpleadoId { get; set; }
    [Display(Name = "Tipo de avería")]
    public int TipoAveriaId { get; set; }
    [Display(Name = "Equipo")]
    public int EquipoId { get; set; }

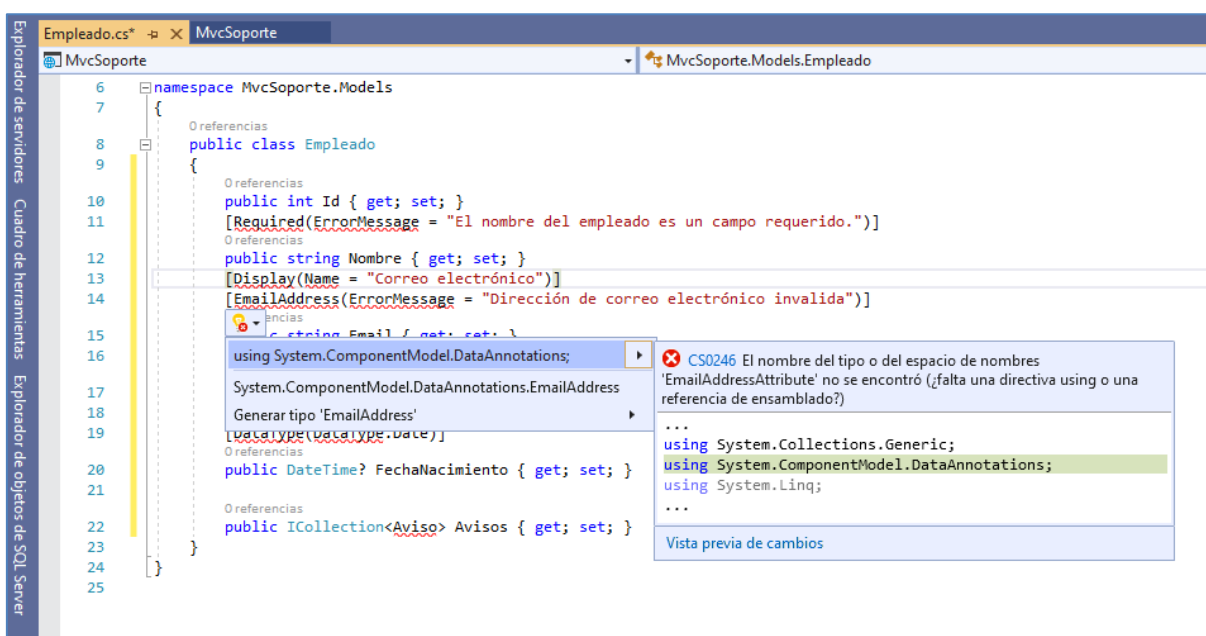
    public Empleado? Empleado { get; set; }
    public TipoAveria? TipoAveria { get; set; }
    public Equipo? Equipo { get; set; }
}
```

Como puede apreciarse en el código de las clases de entidades de datos anteriores, se han añadido anotaciones de validación o *DataAnnotations* para definir características específicas a algunas propiedades. Al añadir el código de las clases podría ocurrir que se subrayarían en rojo las anotaciones de validación, indicando un error. Este error se puede producir porque faltaría añadir una directiva *using* que referencie el espacio de nombres *DataAnnotations* correspondiente. Para solucionarlo, en cada archivo de clase, situar el puntero del ratón sobre

una de las anotaciones de validación subrayada en rojo, desplegar las opciones del icono en forma de bombilla y seleccionar la primera opción propuesta para resolver el error:

```
using System.ComponentModel.DataAnnotations;
```

Evidentemente, en caso de ser necesario, también podría solucionarse el error escribiendo directamente la línea de código anterior al principio del código en cada archivo de clase.



Si fuera necesario, sobre las anotaciones de validación del código que se subrayan en rojo, resolver los errores añadiendo las cláusulas *using* que correspondan en cada caso, tal como se ha realizado anteriormente.

2. A continuación, se va a crear la clase del **contexto de datos**. Mediante el código de la clase del contexto de datos se especifican las entidades de datos incluidas en el modelo y permite personalizar el comportamiento de *Entity Framework Core* en el proyecto de ASP.NET Core MVC. La clase del contexto de datos es una **clase derivada de la clase *DbContext*** del espacio de nombres *Microsoft.EntityFrameworkCore* y especifica los conjuntos de entidades de datos que maneja la aplicación Web que se está desarrollando. El código siguiente crea una propiedad *DbSet* para cada clase de datos del modelo. Por convención, en la terminología del *framework* ORM de *Entity Framework Core*, los nombres de las propiedades *DbSet* se usan como nombres de **tabla** de la base de datos, una entidad se corresponde con una **fila** de una tabla y una propiedad se corresponde con una **columna** de una fila de la tabla. Es importante tener en cuenta que en las aplicaciones Web basadas en ASP.NET Core MVC, la clase del contexto de datos se ubica en la carpeta */Data* del Proyecto. Para crear la clase del contexto de datos, basta con crear un archivo de clase en la carpeta */Data* y añadir en código correspondiente. Por lo tanto, en este punto del desarrollo de la aplicación Web *MvcSoporte*, crear un archivo de clase denominado *MvcSoporteContexto.cs* en la carpeta */Data* y, a continuación, añadir el código que se muestra a continuación.

```
public class MvcSoporteContexto : DbContext
{
    public MvcSoporteContexto(DbContextOptions<MvcSoporteContexto> options)
        : base(options)
    {
    }

    public DbSet<Aviso>? Avisos { get; set; }
    public DbSet<Empleado>? Empleados { get; set; }
    public DbSet<Equipo>? Equipos { get; set; }
    public DbSet<Localizacion>? Localizaciones { get; set; }
    public DbSet<TipoAveria>? TipoAverias { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Poner el nombre de las tablas en singular
        modelBuilder.Entity<Aviso>().ToTable("Aviso");
        modelBuilder.Entity<Empleado>().ToTable("Empleado");
        modelBuilder.Entity<Equipo>().ToTable("Equipo");
        modelBuilder.Entity<Localizacion>().ToTable("Localizacion");
        modelBuilder.Entity<TipoAveria>().ToTable("TipoAveria");

        // Deshabilitar la eliminación en cascada en todas las relaciones
        base.OnModelCreating(modelBuilder);
        foreach (var relationship in
            modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys()))
        {
            relationship.DeleteBehavior = DeleteBehavior.Restrict;
        }
    }
}
```

Si fuera necesario, tal como se ha realizado anteriormente, sobre los elementos del código que se subrayan en rojo, resolver los errores añadiendo las cláusulas *using* que correspondan en cada caso. En el código anterior puede observarse que se ha incluido el método *OnModelCreating()*. En primer lugar, se emplea el método *ToTable()* para establecer una definición explícita de los nombres de las tablas a los elementos *DbSet*, de modo que los nombres de las tablas se especifican en singular. En segundo lugar, se evita la eliminación en cascada en tablas relacionadas. De este modo, se modifica el comportamiento predeterminado de *Entity Framework Core* y se emplea, en su lugar, una configuración de la restricción de integridad referencial más habitual.

3. Compilar el Proyecto de ASP.NET Core MVC, seleccionando la opción **Generar MvcSoporte** del menú **Compilar**. De este modo, las clases de datos y la clase del contexto de datos se ponen a disposición de la aplicación Web y se comprueba que no existen errores. Es importante comprobar que el resultado de la compilación no devuelve errores.

### Registrar el contexto de base de datos que almacena la información que maneja la aplicación Web

Para registrar el contexto de la base de datos, realizar las siguientes acciones:

1. Abrir el archivo *Program.cs*, que se encuentra ubicado en la carpeta raíz del proyecto.
2. En el archivo de inicio *Program.cs*, añadir el código que aparece resaltado a continuación.



```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using MvcSoporte.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ??
    throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));

// Registrar el contexto de la base de datos
builder.Services.AddDbContext<MvcSoporteContexto>(options =>
    options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options =>
    options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>();
. . .
```

En el registro del contexto de datos, se especifica el nombre de la cadena de conexión que va a utilizarse para conectar el contexto de datos *MvcSoporteContexto* con la base de datos correspondiente. Como puede observarse, se especifica la cadena de conexión denominada *DefaultConnection* que se ha creado de manera predeterminada en la aplicación Web.

### Especificar las características de la cadena de conexión de la base de datos

En el archivo *appsettings.json*, que está ubicado en la carpeta raíz del proyecto, se especifican las características de la cadena de conexión *DefaultConnection* que se utiliza para enlazar el contexto de datos *MvcSoporteContexto* de la aplicación Web con la base de datos correspondiente. A continuación, abrir el archivo *appsettings.json* y modificar el código que se muestra resaltado a continuación:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\MSSQL15;Database=aspnet-MvcSoporte-FFA2C5DE-1E14-42EE-9F65-9B8A19098DAD;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

En el código anterior puede observarse que se utilizará la instancia de base de datos denominada *(localdb)\\MSSQL15* y, además, que se empleará el archivo de base de datos de SQL Server denominado *aspnet-MvcSoporte-FFA2C5DE-1E14-42EE-9F65-9B8A19098DAD.mdf*, o nombre similar en cada caso. Este archivo se alojará en la carpeta raíz del usuario: *c:\\Usuarios\\[nombre\_usuario]*.



## Migración inicial para crear la base de datos

En este momento del desarrollo de la aplicación Web interactiva *MvcSoporte*, que está basada en la tecnología de desarrollo ASP.NET Core MVC, se va a crear la base de datos, para lo cual debe realizarse una **Operación de Migración**. Las operaciones de migración permiten crear y actualizar el esquema de una base de datos para que corresponda con la definición de las clases del modelo. Para crear la base de datos mediante una operación de migración inicial, realizar las siguientes acciones:

1. Como paso previo, se debe compilar el proyecto, mediante la opción **Generar MvcSoporte** del menú **Compilar**. Es importante comprobar que no se producen errores de compilación.
2. En el menú **Herramientas**, desplegar la opción **Administrador de paquetes NuGet** y seleccionar la opción **Consola del Administrador de paquetes**.
3. En la **Consola del Administrador de paquetes** ejecutar el siguiente comando:

```
PM> Add-Migration Inicial -context MvcSoporteContexto
```

El comando **Add-Migration** genera un archivo de migración en la carpeta */Migrations* del proyecto con el nombre de la migración especificado, que en este caso es *Inicial*.

4. En la **Consola del Administrador de paquetes** ejecutar el siguiente comando:

```
PM> Update-Database -context MvcSoporteContexto
```

El comando **Update-Database** actualiza el esquema de la base de datos a la migración más reciente. En este caso, ejecuta el método *Up()* del archivo de migración *Inicial.cs* creado en la carpeta */Migrations* mediante la instrucción *Add-Migration* anterior. Y, como consecuencia, de la ejecución del comando *Update-Database* se creará la base de datos.

Si en un futuro, se desea incorporar algún cambio en el código de las clases del modelo especificado en el contexto de datos *MvcSoporteContexto*, entonces deberá realizarse una nueva migración para poder actualizar el esquema de la base de datos a partir del código del modelo (*Code First*).

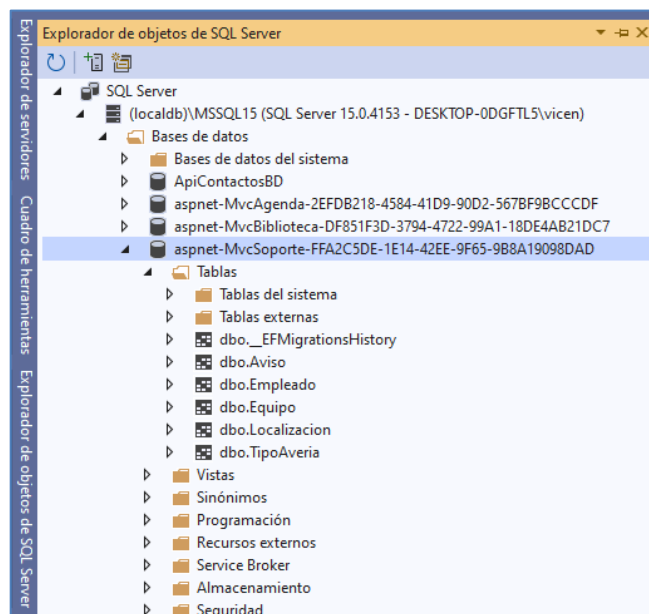
## Comprobar la creación de la base de datos y poder trabajar con ella

Para comprobar que la base de datos se ha creado correctamente, realizar las siguientes acciones:

1. En el menú **Ver**, abrir **Explorador de objetos de SQL Server**. Si fuera necesario, sobre la ventana correspondiente, puede seleccionarse la opción **Ocultar automáticamente** para hacer que quede siempre disponible en la barra vertical izquierda de Visual Studio.
2. Desplegar la opción **SQL Server**. A continuación, desplegar la opción **(localdb)\MSSQL15** y desplegar la opción **Bases de datos**.
3. Desplegar la opción correspondiente al nombre de la base de datos, que coincide con el valor del atributo *Database* del archivo *app-settings.json* y es similar a **aspnet-MvcSoporte-XXXXXX**. Este nombre será diferente en cada caso, debido al sufijo aleatorio que se añade al nombre.
4. Al desplegar la opción **Tablas**, puede observarse que se muestran las tablas: **Aviso**, **Empleado**, **Equipo**, **Localizacion** y **TipoAveria**. Puede comprobarse que estas tablas estarán vacías.

5. Comprobar que se han creado los archivos de base de datos de SQL Server correspondientes, cuya extensión de nombre es .mdf y .ldf, en la carpeta `c:\Usuarios\[nombre_usuario]`.

9



**Crear los controladores, y sus vistas asociadas, para generar un procesamiento de tipo CRUD para cada una de las tablas de la base de datos que maneja la aplicación Web**

En este apartado se desarrollarán los procesamientos de tipo CRUD (*Create Read Update and Delete*) de cada una de las clases de datos del modelo. Para ello, realizar las siguientes acciones:

1. Siguiendo el procedimiento empleado en la práctica anterior mediante el uso del asistente de *Scaffold*, crear los procesamientos de tipo CRUD correspondientes a las tablas de la base de datos que maneja la aplicación Web. Los nombres de los controladores con vistas a crear son los siguientes: *EmpleadosController.cs*, *TipoAveriasController.cs*, *LocalizacionesController.cs*, *EquiposController.cs* y *AvisosController.cs*.
2. Una vez creados los controladores y las vistas correspondientes, se puede probar el funcionamiento de los procesamientos tipo CRUD creados, llamando a la ejecución de la acción *Index()* de cada controlador directamente desde la barra de direcciones del navegador. Así, por ejemplo, puede comprobarse que una vez que se haya iniciado la depuración, se puede introducir el texto <http://localhost:xxxxx/Empleados/Index> en la barra de direcciones del navegador para solicitar la acción *Index()* del controlador *EmpleadosController.cs*.

En las vistas del procesamiento de tipo CRUD de la clase de datos *Equipo*, a continuación, va a modificarse el código para conseguir una presentación visual y una introducción o edición del precio del equipo que sea adecuada como valor monetario. Esta acción es necesario realizarla cuando la configuración cultural de la aplicación Web es distinta de la inglesa (en-gb) o americana (en-us). Muy posiblemente, la configuración cultural que se estará utilizando será la española (es-es), por lo que deberán realizarse las siguientes acciones para conseguir que los valores monetarios se presenten adecuadamente, mediante la coma como punto decimal y el punto como separador de miles.

Para conseguir que la presentación visual y la introducción o edición del valor del precio del equipo se realicen de manera adecuada con un formato monetario en español, se han especificado dos propiedades diferentes que representan ambas el valor del precio en la clase de datos *Equipo*. Una de ellas, la propiedad *PrecioCadena* es un campo calculado de tipo *string* que se utilizará para introducir los valores de los precios en los cuadros de texto de las vistas. Y la otra, la propiedad *Precio*, que es de tipo *decimal*, se utilizará para presentar visualmente los valores en formato monetario sobre las páginas Web. Para adecuar el código de las vistas del CRUD correspondiente a la entidad *Equipo*, de manera que se utilice el valor del precio con un formato monetario adecuado que esté basado en la configuración cultural en español, realizar las siguientes acciones:

1. En las vistas *Index.cshtml*, *Details.cshtml* y *Delete.cshtml*, que están alojadas en la carpeta */Views/Equipos*, es necesario comentar el código correspondiente a la presentación visual de la propiedad *PrecioCadena*. Para ello, hacer:
  - a. Abrir el archivo de vista *Index.cshtml* que está alojado en la carpeta */Views/Equipos* y realizar las modificaciones en el código que aparecen resaltadas a continuación:

```
@model IEnumerable<MvcSoporte.Models.Equipo>

@{
    ViewData["Title"] = "Index";
}

<h1>Equipos</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.CodigoEquipo)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Marca)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Modelo)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.NumeroSerie)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Caracteristicas)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Precio)
            </th>
            @*
            <th>
                @Html.DisplayNameFor(model => model.PrecioCadena)
            </th>
            *@
            <th>
                @Html.DisplayNameFor(model => model.FechaCompra)
            </th>
        </tr>
    </thead>
</table>
```

```
</th>
<th>
    @Html.DisplayNameFor(model => model.FechaBaja)
</th>
<th>
    @Html.DisplayNameFor(model => model.Localizacion)
</th>
<th></th>
</tr>
</thead>
<tbody>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.CodigoEquipo)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Marca)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Modelo)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.NumeroSerie)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Caracteristicas)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Precio)
        </td>
        <td>
            @*
            @Html.DisplayFor(modelItem => item.PrecioCadena)
            *@
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FechaCompra)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FechaBaja)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Localizacion.Descripcion)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
            <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
        </td>
    </tr>
}
</tbody>
</table>
```

Al realizar estas modificaciones, se obtendrá el siguiente resultado al iniciar la depuración y solicitar la dirección: <https://localhost:xxxxx/Equipos/Index>.

MvcSoporte

Home

Privacy

Register

Login

Equipos

Create New

Código equipo	Marca	Modelo	Número de serie	Características técnicas	Precio	Fecha de compra	Fecha de baja	Localizacion	
PC20167	HP	Pavilion 1023	ZXD2374SL	Intel i3. 4 GB. HD 512 GB. Nvidia 450	452,30 €	12/08/2013		Sala de reuniones	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
PC34778	Dell	A320-ES	XDF7865FC	Intel i5. 8 GB. HD 1TB. Nvidia 68500	545,80 €	03/01/2016		Dirección	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
PR23451	HP	Deskjet 6830	FDT3245VX	Color. Chorro de tinta	152,25 €	14/04/2015		Dirección	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
PR28762	HP	Laserjet 5690	KYJ5296TP	Laser. B/N. Wi-Fi	75,00 €	23/03/2016		Departamento de Administración	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
PC23917	Asus	T34-SP	SPT3851TL	Intel i3. 4 GB. HD 512 GB. Nvidia 350	359,35 €	12/04/2014		Despacho 101	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
PC95781	HP	Pavilion 5300	LGP4901SP	Intel i7. 8 GB. HD 1TB. Intel Graphics	839,00 €	02/05/2016		Despacho 202	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

- b. Proceder de manera similar en las vistas *Details.cshtml* y *Delete.cshtml* de la carpeta */Views/Equipos*, para comentar la presentación visual de la propiedad *PrecioCadena*.
2. En las vistas *Create.cshtml* y *Edit.cshtml*, que están alojadas en la carpeta */Views/Equipos*, es necesario comentar el código correspondiente a la introducción del valor de la propiedad *Precio*. Para ello, hacer:
  - a. Abrir el archivo de vista *Create.cshtml* que está alojado en la carpeta */Views/Equipos* y realizar la modificación en el código que aparece resaltada a continuación:

```

...
<div class="form-group">
  <label asp-for="Caracteristicas" class="control-label"></label>
  <input asp-for="Caracteristicas" class="form-control" />
  <span asp-validation-for="Caracteristicas" class="text-danger"></span>
</div>
@*
<div class="form-group">
  <label asp-for="Precio" class="control-label"></label>
  <input asp-for="Precio" class="form-control" />
  <span asp-validation-for="Precio" class="text-danger"></span>
</div>
*@
<div class="form-group">
  <label asp-for="PrecioCadena" class="control-label"></label>
  <input asp-for="PrecioCadena" class="form-control" />
  <span asp-validation-for="PrecioCadena" class="text-danger"></span>

```

```
</div>
<div class="form-group">
  <label asp-for="FechaCompra" class="control-label"></label>
  <input asp-for="FechaCompra" class="form-control" />
  <span asp-validation-for="FechaCompra" class="text-danger"></span>
</div>
```

...

Y ahora, al realizar esta modificación en el código, se obtendrá el siguiente resultado al iniciar la depuración y solicitar la URL: <https://localhost:xxxxx/Equipos/Create>.

MvcSoporte Home Privacy

Create

Equipo

Código equipo

Marca

Modelo

Número de serie

Características técnicas

Precio

Fecha de compra

mm/dd/aaaa

Fecha de baja

mm/dd/aaaa

Localización

Despacho 101

Create

Back to List

- b. Proceder de manera similar en la vista *Edit.cshtml* de la carpeta */Views/Equipos*, para comentar el código que facilita la introducción del valor de la propiedad *Precio*.

- Finalmente realizar las pruebas necesarias para comprobar que la presentación visual y la introducción del valor del precio del equipo son adecuadas, cuando se utiliza el formato monetario de la configuración cultural en español (es-es). De este modo, el valor del precio del equipo se presenta visualmente con el carácter coma como punto decimal y el carácter punto se utiliza como separador de miles. Comprobar, además, que se admite el carácter punto o el carácter coma para representar el punto decimal, cuando se introduce el valor del precio desde el teclado, tanto al crear como al editar la información del precio de un equipo.

### Crear las opciones de menú para los controladores creados, comprobar el funcionamiento de los procesamientos de tipo CRUD y añadir información en cada una de las tablas de la base de datos

Para finalizar la parte de desarrollo que permite obtener el prototipo más básico de la aplicación Web *MvcSoporte*, realizar los cambios necesarios en el archivo `_Layout.cshtml` para añadir las opciones del menú que faciliten el acceso a la gestión de empleados, tipos de averías, Localizaciones, equipos y avisos. Estas acciones se realizarán del mismo modo que se hizo en un ejercicio de la práctica anterior.

Se recomienda que al mismo tiempo que se va probando el funcionamiento de los procesamientos de tipo CRUD creados, **se vaya introduciendo información** en cada una de las tablas. De este modo, se dispondrá de información almacenada suficiente para poder realizar las pruebas de funcionamiento más adecuadas en los siguientes ejercicios. Se recomienda introducir, como mínimo, unas 10 filas en una cada una de las tablas.

MvcSoporte	<a href="#">Inicio</a>	<a href="#">Privacidad</a>	<a href="#">Avisos</a>	<a href="#">Equipos</a>	<a href="#">Localizaciones</a>	<a href="#">Averías</a>	<a href="#">Empleados</a>	<a href="#">Register</a>	<a href="#">Login</a>
<h2>Empleados</h2> <a href="#">Create New</a>									
Nombre	Correo electrónico	Teléfono	Fecha de nacimiento						
Julio García Sogorb	<a href="mailto:jgarcia@empresa.com">jgarcia@empresa.com</a>	659223344	21/12/1991	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
Eva Juarez Lujan	<a href="mailto:ejarez@empresa.com">ejarez@empresa.com</a>	623998877	02/05/1982	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
Javier Sogorb Espinosa	<a href="mailto:jsogorb@empresa.com">jsogorb@empresa.com</a>	636445577	01/01/1984	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
María García Pérez	<a href="mailto:mgarcia@empresa.com">mgarcia@empresa.com</a>	676552299	02/10/1992	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
Alberto Fenoll Llópis	<a href="mailto:afenoll@empresa.com">afenoll@empresa.com</a>	646112277	02/11/1984	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
Josep Joan Torró i Falcó	<a href="mailto:jtorro@empresa.com">jtorro@empresa.com</a>	616553377	15/08/1987	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
María de los Ángeles Pérez Casado	<a href="mailto:mperez@empresa.com">mperez@empresa.com</a>	656229911	12/10/1993	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
Marta Betoret i Masià	<a href="mailto:mbetoret@empresa.com">mbetoret@empresa.com</a>	626449911	22/10/1983	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
Joan Beltrá i Lloret	<a href="mailto:jbeltra@empresa.com">jbeltra@empresa.com</a>	686554422	27/03/1975	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
Ángeles Martínez Molina	<a href="mailto:amartinez@empresa.com">amartinez@empresa.com</a>	676228877	26/10/1976	<a href="#">Edit</a>	<a href="#">Details</a>	<a href="#">Delete</a>			
© 2022 - MvcSoporte - IES Mare Nostrum. DAW-DWS - <a href="#">Política de privacidad</a>									



## Ejercicio 2

### Paginación en los procesamientos de tipo CRUD

En el ejercicio anterior se ha creado la aplicación Web *MvcSoporte* y, además, se han implementado los procesamientos de tipo CRUD de las tablas: *Empleado*, *TipoAveria*, *Localizacion*, *Equipo* y *Aviso*. Con ello, la aplicación Web *MvcSoporte* ha alcanzado un estado básico de desarrollo. En este y en los siguientes ejercicios de la práctica, se van a ir añadiendo nuevas funcionalidades a la aplicación Web, de manera que irá evolucionando su desarrollo informático.

En este ejercicio se va a incorporar la funcionalidad de paginación en el procesamiento de tipo CRUD correspondiente a la tabla *Empleado* en la aplicación Web *MvcSoporte*. Para ello, se modificará el código de la acción *Index()* del controlador *EmpleadosController.cs* y el de la vista correspondiente.

#### Paginación en la lista de datos de los empleados

Para implementar la característica de la paginación es necesario, como paso previo, crear una clase genérica en el proyecto que permita seleccionar el número de filas que forma cada página en una lista de datos. Una vez creada esta clase genérica, se realizarán cambios en el código del método *Index()* del controlador *EmpleadosController.cs*. Y, finalmente, se modificará el código de la vista *Index.cshtml* correspondiente, para agregar los botones de paginación que permitirán acceder a la página anterior y la página siguiente de la lista de datos. Para agregar paginación en la lista de empleados, hacer:

1. Como paso previo, en primer lugar, se creará una clase genérica denominada *PaginatedList.cs* en la carpeta raíz del Proyecto de ASP.NET Core MVC. Para ello, hacer:
  - a. En el **Explorador de soluciones**, hacer clic en botón derecho sobre el nombre del proyecto *MvcSoporte*, desplegar la opción **Agregar** y seleccionar la opción **Clase...**
  - b. Introducir *PaginatedList.cs* como **Nombre** de la clase y hacer clic en **Agregar**.
  - c. En el archivo de clase *PaginatedList.cs*, agregar el siguiente código:

```
public class PaginatedList<T> : List<T>
{
    public int PageIndex { get; private set; }
    public int TotalPages { get; private set; }

    public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)
    {
        PageIndex = pageIndex;
        TotalPages = (int)Math.Ceiling(count / (double)pageSize);

        this.AddRange(items);
    }

    public bool HasPreviousPage
    {
        get
        {
            return (PageIndex > 1);
        }
    }

    public bool HasNextPage
```

```
{
    get
    {
        return (PageIndex < TotalPages);
    }
}

public static async Task<PaginatedList<T>> CreateAsync(IQueryable<T> source,
    int pageIndex, int pageSize)
{
    var count = await source.CountAsync();
    var items = await source.Skip((pageIndex - 1) * pageSize)
        .Take(pageSize)
        .ToListAsync();
    return new PaginatedList<T>(items, count, pageIndex, pageSize);
}
```

Si fuera necesario, agregar el espacio de nombres *Microsoft.EntityFrameworkCore* al código de la clase mediante la cláusula *using* correspondiente. En el código anterior, puede apreciarse que la clase *PaginatedList<T>* deriva de la clase base *List<T>*. También puede apreciarse que el método *CreateAsync()* toma como parámetro un tipo de datos *IQueryable*, el número de la página y el tamaño de la página. Y, a continuación, aplica los métodos *Skip()* y *Take()* al tipo de datos *IQueryable* que guarda el conjunto de entidades recuperado del modelo. Estos dos métodos permiten establecer un filtro sobre las filas recuperadas cuando se llama al método *ToListAsync()*, de modo que solo se devolverán las filas correspondientes a la página solicitada, en lugar de recuperar todas las filas. Las propiedades *HasPreviousPage* y *HasNextPage* se pueden usar para habilitar o deshabilitar los botones de la página **Siguiente** y **Anterior**. Para crear el objeto *PaginatedList<T>* se utilizará la llamada al método *CreateAsync()*, en lugar de usar un constructor, porque los constructores no pueden ejecutar código asíncrono.

2. Abrir el controlador *EmpleadosController.cs* para comentar y modificar el código existente en el método de acción *Index()*, tal como se muestra resaltado a continuación.

```
// GET: Empleados
public async Task<IActionResult> Index(int? pageNumber)
{
    // Cargar datos de Empleados
    var empleados = from s in _context.Empleados
        select s;

    int pageSize = 3;
    return View(await PaginatedList<Empleado>.CreateAsync(empleados.AsNoTracking(),
        pageNumber ?? 1, pageSize));

    // return View(await _context.Empleados.ToListAsync());
}
```

En el código anterior puede apreciarse que se ha agregado un parámetro, denominado *pageNumber*, a la acción *Index()* del controlador *Empleados*. Este parámetro representa el número de página actual de la lista de empleados. También puede apreciarse el uso que se hace del método *CreateAsync()* de la clase *PaginatedList<T>* para convertir la consulta de todos los empleados en una consulta de una sola página que pertenece a un tipo de colección que admite la paginación. Como puede apreciarse, esta consulta de una sola página es la que se

pasa a la vista. En el código anterior, la expresión `pageNumber ?? 1` devuelve el valor de `pageNumber` si tiene algún valor o devuelve 1 si `pageNumber` es `null`. Puede apreciarse la utilización del método `AsNoTracking()` de *Entity Framework Core* que devuelve una consulta nueva y las entidades devueltas no serán almacenadas en caché por el contexto, de manera que no se realiza un seguimiento de los cambios realizados. El uso del método `AsNoTracking()` mejora el rendimiento, en caso de que no se vayan a actualizar las entidades devueltas por la consulta durante el contexto actual.

3. Abrir la vista `Index.cshtml` alojada en la subcarpeta de vistas `/Views/Empleados`, modificar el código que aparece resaltado a continuación.

```
@* @model IEnumerable<MvcSoporte.Models.Empleado> *@
@model PagedList<MvcSoporte.Models.Empleado>

@{
    ViewData["Title"] = "Index";
}

<h1>Empleados</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                Nombre
                @* @Html.DisplayNameFor(model => model.Nombre) *@
            </th>
            <th>
                Correo electrónico
                @* @Html.DisplayNameFor(model => model.Email) *@
            </th>
            <th>
                Teléfono
                @* @Html.DisplayNameFor(model => model.Telefono) *@
            </th>
            <th>
                Fecha de nacimiento
                @* @Html.DisplayNameFor(model => model.FechaNacimiento) *@
            </th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Nombre)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Email)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Telefono)
                </td>
```

```

        <td>
            @Html.DisplayFor(modelItem => item.FechaNacimiento)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
            <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
        </td>
    </tr>
}
</tbody>
</table>

@* Poner botones de página Siguiente y Anterior *@
@{
    var prevDisabled = !Model.HasPreviousPage ? "disabled" : "";
    var nextDisabled = !Model.HasNextPage ? "disabled" : "";
}
<a asp-action="Index" asp-route-pageNumber="@((Model.PageIndex - 1))"
    class="btn btn-outline-secondary @prevDisabled">
    Anterior
</a>
<a asp-action="Index" asp-route-pageNumber="@((Model.PageIndex + 1))"
    class="btn btn-outline-secondary @nextDisabled">
    Siguiente
</a>
  
```

En el código puede apreciarse como la directiva `@model` especifica que la vista obtiene un objeto `PagedList<T>`. Este objeto no admite el uso del helper `@Html.DisplayNameFor()` para obtener los nombres de las propiedades de la clase `Empleado` del modelo, por lo que se especifica el texto correspondiente. También se aprecia el código añadido para especificar los elementos de navegación que permiten avanzar y retroceder las páginas de la lista de datos.

4. Iniciar la depuración para comprobar los resultados obtenidos.

MvcSoporte	Home	Privacy	Avisos	Equipos	Localizaciones	Averias	Empleados	Register	Login
------------	------	---------	--------	---------	----------------	---------	-----------	----------	-------

<h2>Empleados</h2> <a href="#">Create New</a>				
Nombre	Email	Teléfono	Fecha de nacimiento	
Julio García Sogorb	jgarcia@empresa.com	659223344	21/12/1991	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Eva Juarez Lujan	ejuarez@empresa.com	623998877	02/05/1982	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Javier Sogorb Espinosa	jsogorb@empresa.com	636445577	01/01/1984	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

Anterior	Siguiente
----------	-----------

## Ejercicio 3

### Ordenación, búsqueda y filtrado en los procesamientos de tipo CRUD

En este ejercicio incorporarán las funcionalidades de ordenación, búsqueda y filtrado al procesamiento de tipo CRUD la tabla *Aviso* en la aplicación Web *MvcSoporte*. Para ello, se modificará el código de la acción *Index()* del controlador *AvisosController.cs* y también el código de la vista correspondiente.

#### Ordenación de la lista de avisos

La información sobre los avisos de soporte relativos a las incidencias informáticas almacenadas por los empleados se presenta desordenada. Es interesante que esta información se muestre ordenada por el campo *FechaAviso* de manera descendente. De esta forma, la información sobre los avisos más recientes se mostrará al principio de la lista. Para ello, realizar las siguientes acciones:

1. Abrir el archivo *AvisosController.cs* situado en la carpeta */Controllers*.
2. En el método de acción *Index()*, agregar y modificar el código resaltado que se muestra a continuación.

```
// GET: Avisos
public async Task<IActionResult> Index()
{
    // Cargar datos de los avisos
    var avisos = _context.Avisos.AsQueryable();

    // Ordenar los avisos de forma descendente por FechaAviso
    avisos = avisos.OrderByDescending(s => s.FechaAviso);

    avisos = avisos.Include(a => a.Empleado)
                   .Include(a => a.Equipo)
                   .Include(a => a.TipoAveria);

    return View(await avisos.AsNoTracking().ToListAsync());

    // var mvcSoporteContexto = _context.Avisos.Include(a => a.Empleado).Include(a =>
    // a.Equipo).Include(a => a.TipoAveria);
    // return View(await mvcSoporteContexto.ToListAsync());
}
```

En el código anterior puede apreciarse el uso que se hace del método *AsQueryable()* de *LINQ to Entities* para obtener la colección de entidades de tipo *Aviso* que forman la tabla o *DbSet Avisos*. Dicho de otro modo, el método *AsQueryable()* devuelve una consulta que contiene la lista de todos los avisos almacenados. También puede observarse el uso del método *OrderByDescending()* para ordenar la lista de avisos, según el valor de la propiedad *FechaAviso* y en orden descendente. La sintaxis de algunas consultas basadas en métodos de *LINQ to Entities* suelen utilizar expresiones lambda como parámetro. En este caso, se usa una expresión lambda para especificar la propiedad por la que se ordena. Finalmente, se utiliza el método *Include()* para incluir las entidades relacionadas en la consulta.

3. Iniciar la depuración para comprobar los resultados obtenidos.

MvcSoporte	Home	Privacy	Avisos	Equipos	Localizaciones	Averias	Empleados	Register	Login
------------	------	---------	--------	---------	----------------	---------	-----------	----------	-------

## Avisos

[Create New](#)

Descripción del problema	Fecha de aviso	Fecha de cierre	Observaciones	Empleado	TipoAveria	Equipo	
No se pone en marcha	14/05/2019			Javier Sogorb Espinosa	No arranca el sistema	PC20167	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
No responde el teclado	02/05/2019	02/05/2019	Sustitución teclado	María García Pérez	No funciona el teclado	PC34778	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
No se pone en marcha el ordenador	01/05/2019	04/05/2019	Sustitución de fuente de alimentación	Josep Joan Torró i Falcò	No arranca el sistema	PC23917	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
No imprime bien la impresora	01/05/2019	01/05/2019	Sustitucion de toner	Joan Beltrá i Lloret	La impresora no tiene tinta	PR28762	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Aparece error grave del sistema	12/04/2019	14/04/2019	Reinstalación de Windows	Alberto Fenoll Llópis	Problema indeterminado	PC12391	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
No aparece nada en la pantalla	05/04/2018	12/04/2018	Se restaura la imagen anterior	Ángeles Martínez Molina	No arranca el sistema	PC95781	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

### Búsqueda de filas en la lista de valores

A continuación, se añadirá la funcionalidad que permite buscar en la lista de valores de la tabla *Aviso* por el nombre del empleado. De esta manera, se podrá obtener la información de los avisos a través de una búsqueda realizada a partir del nombre del empleado que realizó el aviso. En primer lugar, se modificará el código del método de acción *Index()* del controlador *AvisosController.cs* para poder obtener la información según el valor de búsqueda. Y, en segundo lugar, se modificará el código del archivo de la vista correspondiente para agregar un formulario a la vista. Este formulario incluirá un cuadro de texto, que permitirá introducir el nombre del empleado que actuará como valor de búsqueda, y un botón de envío. Para ello, realizar las siguientes acciones:

1. En la acción *Index()* del archivo *AvisosController.cs*, añadir el código resaltado a continuación.

```

public async Task<IActionResult> Index(string strCadenaBusqueda)
{
    ViewData["BusquedaActual"] = strCadenaBusqueda;

    // Cargar datos de los avisos
    var avisos = _context.Avisos.AsQueryable();

    // Ordenar descendente por FechaAviso
    avisos = avisos.OrderByDescending(s => s.FechaAviso);

    // Para buscar avisos por nombre de empleado en la lista de valores
  
```

```
if (!String.IsNullOrEmpty(strCadenaBusqueda))
{
    avisos = avisos.Where(s => s.Empleado.Nombre.Contains(strCadenaBusqueda));
}

avisos = avisos.Include(a => a.Empleado)
                .Include(a => a.Equipo)
                .Include(a => a.TipoAveria);

return View(await avisos.AsNoTracking().ToListAsync());

// var mvcSoporteContexto = _context.Avisos.Include(a => a.Empleado).Include(a =>
a.Equipo).Include(a => a.TipoAveria);
// return View(await mvcSoporteContexto.ToListAsync());
}
```

En el código anterior, puede apreciarse que se ha añadido el parámetro *strCadenaBusqueda* al método de acción *Index()* del controlador *AvisosController.cs* y cuyo valor se pasará desde la vista *Index.cshtml* de la subcarpeta de vistas */Views/Avisos* a través de un formulario. También puede apreciarse el uso del objeto *ViewData* que permite pasar datos auxiliares desde una acción de un controlador hacia la vista correspondiente. Finalmente, puede apreciarse el uso que se hace de los métodos de operación *Where()* y *Contains()* de *LINQ to Entities*.

2. Abrir el archivo de vista *Index.cshtml* situado en la subcarpeta de vistas */Views/Avisos* y, a continuación, modificar o agregar las líneas del código que aparecen resaltadas a continuación:

```
@model IEnumerable<MvcSoporte.Models.Aviso>

@{
    ViewData["Title"] = "Index";
}

<h1>Avisos</h1>

<p>
    <a asp-action="Create">Nuevo aviso</a>
</p>

@* Agregar un cuadro de búsqueda en la vista Index de Avisos *@
<form asp-action="Index">
    <div class="container">
        <div class="row align-items-start">
            <div class="col-md">
                <label class="control-label">Nombre empleado:</label>
                <input type="text" name="strCadenaBusqueda"
                    value="@ViewData["BusquedaActual"]" class="form-control" />
            </div>
            <div class="col-md align-self-end">
                <input type="submit" value="Buscar" class="btn btn-default" /> |
                <a asp-action="Index" class="btn btn-default">Lista completa</a>
            </div>
        </div>
    </div>
</form>

<table class="table">
    <thead>
        <tr>
```



```
<th>
    @Html.DisplayNameFor(model => model.Descripcion)
</th>
<th>
    @Html.DisplayNameFor(model => model.FechaAviso)
</th>
<th>
    @Html.DisplayNameFor(model => model.FechaCierre)
</th>
<th>
    @Html.DisplayNameFor(model => model.Observaciones)
</th>
<th>
    @Html.DisplayNameFor(model => model.Empleado)
</th>
<th>
    Tipo de avería
    @* @Html.DisplayNameFor(model => model.TipoAveria) *@
</th>
<th>
    @Html.DisplayNameFor(model => model.Equipo)
</th>
<th></th>
</tr>
</thead>
<tbody>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Descripcion)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.FechaAviso)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.FechaCierre)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Observaciones)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Empleado.Nombre)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.TipoAveria.Descripcion)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Equipo.CodigoEquipo)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.Id">Editar</a> |
                <a asp-action="Details" asp-route-id="@item.Id">Detalles</a> |
                <a asp-action="Delete" asp-route-id="@item.Id">Eliminar</a>
            </td>
        </tr>
    }
</tbody>
</table>
```

En el código anterior puede apreciarse que el nombre del cuadro de texto donde se introducirá el texto a buscar, que es *strCadenaBusqueda*, coincide con el nombre del parámetro definido en el método *Index()* del controlador *AvisosController.cs* y, por lo tanto, este parámetro tomará el valor introducido en el cuadro de texto. Puede observarse que se usa el atributo *asp-action* en la etiqueta *form* para especificar la acción de destino del formulario, que en este caso es la acción *Index()* de controlador actual. El método de envío predeterminado del formulario es POST, sin embargo no se ha especificado la sobrecarga *HttpPost* de la acción *Index()*. Como puede comprobarse esto no supone ningún problema, porque el envío es atendido por la misma acción *Index()*, al no haber sido definida una sobrecarga del método para los envíos POST. Además, se ha modificado el código para mejorar la presentación. Concretamente, se han puesto en castellano los textos de los enlaces hacia las acciones de manipulación de datos y se han mejorado algunos textos de la cabecera de la lista de valores. También puede apreciarse que, como es habitual, se han comentado las líneas de código sobrantes en lugar de eliminarlas, utilizando los símbolos *@\** y *\*@* de Razor para abrir y cerrar los comentarios.

3. Iniciar la depuración para comprobar los resultados obtenidos.

MvcSoporte
Inicio
Privacidad
Avisos
Equipos
Localizaciones
Averías
Empleados
Register
Login

## Avisos

[Nuevo aviso](#)

Nombre empleado:  [Buscar](#) | [Lista completa](#)

Descripción del problema	Fecha de aviso	Fecha de cierre	Observaciones	Empleado	Tipo de avería	Equipo
No arranca	21/06/2019			Joan Beltrá i Lloret	No arranca el sistema	PC20167 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>
No se pone en marcha el ordenador	01/05/2019	04/05/2019	Sustitución de fuente de alimentación	Josep Joan Torró i Falcó	No arranca el sistema	PC23917 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>
No imprime bien la impresora	01/05/2019	01/05/2019	Sustitucion de toner	Joan Beltrá i Lloret	La impresora no tiene tinta	PR28762 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>
No va el ratón	03/05/2017	04/02/2019	Sustitución	Josep Joan Torró i Falcó	No funciona el ratón	PC23917 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>

© 2022 - MvcSoporte - IES Mare Nostrum. DAW-DWS - [Política de privacidad](#)

### Filtrado de filas en la lista de valores

A continuación, se añadirá un filtrado por tipo de avería en la lista de valores del procesamiento CRUD de la tabla *Aviso*. En primer lugar, se modificará el código en el controlador para poder presentar la información de los avisos según el valor del tipo de avería seleccionado en la vista. Y, en segundo lugar, se agregará un cuadro combinado a la vista, que permitirá seleccionar el tipo de avería por el que se desea filtrar la información. Para ello, realizar las siguientes acciones:

1. En la acción *Index()* del archivo *AvisosController.cs*, que está situado en la carpeta */Controllers*, añadir el código que aparece resaltado a continuación.

```
public async Task<IActionResult> Index(string strCadenaBusqueda,
                                     int? intTipoAveriaId)
{
    ViewData["BusquedaActual"] = strCadenaBusqueda;

    // Cargar datos de los avisos
    var avisos = _context.Avisos.AsQueryable();

    // Ordenar descendente por FechaAviso
    avisos = avisos.OrderByDescending(s => s.FechaAviso);

    // Para buscar avisos por nombre de empleado en la lista de valores
    if (!String.IsNullOrEmpty(strCadenaBusqueda))
    {
        avisos = avisos.Where(s => s.Empleado.Nombre.Contains(strCadenaBusqueda));
    }

    // Para filtrar avisos por tipo de avería
    if (intTipoAveriaId == null)
    {
        ViewData["TipoAveriaId"] = new SelectList(_context.TipoAverias, "Id",
                                                "Descripcion");
    }
    else
    {
        ViewData["TipoAveriaId"] = new SelectList(_context.TipoAverias, "Id",
                                                "Descripcion", intTipoAveriaId);
        avisos = avisos.Where(x => x.TipoAveriaId == intTipoAveriaId);
    }

    avisos = avisos.Include(a => a.Empleado)
                    .Include(a => a.Equipo)
                    .Include(a => a.TipoAveria);

    return View(await avisos.AsNoTracking().ToListAsync());

    // var mvcSoporteContexto = _context.Avisos.Include(a => a.Empleado).Include(a =>
    // a.Equipo).Include(a => a.TipoAveria);
    // return View(await mvcSoporteContexto.ToListAsync());
}
```

En el código anterior puede apreciarse que se utiliza el objeto *ViewData* para pasar la lista de los tipos de avería a la vista. En efecto, la instrucción *ViewData["TipoAveriaId"]* pasa a la vista una lista de los tipos de avería existentes, que estará formada por las propiedades *Id* y *Descripción*. Esta lista permitirá componer un cuadro combinado para seleccionar en la vista el tipo de avería por el que se desean filtrar los avisos. También puede apreciarse que se ha añadido el parámetro *intTipoAveriaId* a la acción *Index()*. Este parámetro acogerá el valor del tipo de avería que será seleccionado en la vista y enviado a través del formulario. Como se puede comprobar, el nombre de este parámetro coincide con el nombre del cuadro combinado que permite seleccionar el tipo de avería en la vista para el filtrado de los avisos.

2. En el archivo de vista *Index.cshtml*, que está situado en la carpeta */Views/Avisos*, agregar el código que aparece resaltado a continuación.

```
@model IEnumerable<MvcSoporte.Models.Aviso>

@{
    ViewData["Title"] = "Index";
}

<h1>Avisos</h1>

<p>
    <a asp-action="Create">Nuevo aviso</a>
</p>

@* Agregar un cuadro de búsqueda en la vista Index de Avisos *@
<form asp-action="Index">
    <div class="container">
        <div class="row align-items-start">
            <div class="col-md">
                <label class="control-label">Nombre empleado:</label>
                <input type="text" name="strCadenaBusqueda"
                    value="@ViewData["BusquedaActual"]" class="form-control" />
            </div>
            <div class="col-md">
                <label class="control-label">Tipo de avería:</label>
                <select name="intTipoAveriaId" asp-items="ViewBag.TipoAveriaId"
                    class="form-control">
                    <option value="">Todas</option>
                </select>
            </div>
            <div class="col-md align-self-end">
                <input type="submit" value="Buscar" class="btn btn-default" /> |
                <a asp-action="Index" class="btn btn-default">Lista completa</a>
            </div>
        </div>
    </div>
</form>

<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Descripcion)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.FechaAviso)
            </th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>
                @Html.DisplayFor(modelItem => model.Descripcion)
            </td>
            <td>
                @Html.DisplayFor(modelItem => model.FechaAviso)
            </td>
        </tr>
    </tbody>
</table>
```

En el código anterior, el asistente de etiqueta <select> generará un cuadro combinado en la página de respuesta que contendrá los tipos de averías. Este cuadro combinado, denominado *intTipoAveriaId*, permitirá seleccionar el tipo de avería para filtrar la información. Este valor se pasará a la acción *Index()* a través del formulario. En el código puede apreciarse que el nombre del cuadro combinado coincide con el nombre del parámetro correspondiente en la acción *Index()*, lo que, precisamente, hace que el valor seleccionado en el cuadro combinado se pase desde la vista hacia la acción del controlador a través del formulario y pueda ser recuperado.

3. Iniciar la depuración para comprobar los resultados obtenidos.

MvcSoporte Inicio Privacidad Avisos Equipos Localizaciones Averías Empleados Register Login

## Avisos

[Nuevo aviso](#)

Nombre empleado:  Tipo de avería:  [Buscar](#) | [Lista completa](#)

Descripción del problema	Fecha de aviso	Fecha de cierre	Observaciones	Empleado	Tipo de avería	Equipo
No arranca	21/06/2019			Joan Beltrá i Lloret	No arranca el sistema	PC20167 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>
No se pone en marcha el ordenador	01/05/2019	04/05/2019	Sustitución de fuente de alimentación	Josep Joan Torró i Falcó	No arranca el sistema	PC23917 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>

### Paginación de las filas en la lista de avisos

Finalmente, se va a agregar paginación a la lista de avisos en el procesamiento de tipo CRUD de la tabla *Aviso*. Se realizarán cambios en el código de la acción *Index()* del controlador *AvisosController.cs* y de la vista *Index.cshtml* correspondiente que está situada en la subcarpeta de vistas */Views/Avisos*. Para agregar la funcionalidad de paginación en la lista de avisos, realizar las siguientes acciones:

1. En el método de acción *Index()* del archivo *AvisosController.cs*, agregar el código que aparece resaltado a continuación.

```
// GET: Avisos
public async Task<IActionResult> Index(string strCadenaBusqueda,
    string busquedaActual, int? intTipoAveriaId, int? tipoAveriaIdActual,
    int? pageNumber)
{
    if (strCadenaBusqueda != null)
    {
        pageNumber = 1;
    }
    else
    {
        strCadenaBusqueda = busquedaActual;
    }
    ViewData["BusquedaActual"] = strCadenaBusqueda;

    if (intTipoAveriaId != null)
    {
        pageNumber = 1;
    }
    else
    {
        intTipoAveriaId = tipoAveriaIdActual;
    }
    ViewData["TipoAveriaIdActual"] = intTipoAveriaId;
}
```

```
// Cargar datos de los avisos
var avisos = _context.Avisos.AsQueryable();

// Ordenar descendente por FechaAviso
avisos = avisos.OrderByDescending(s => s.FechaAviso);

// Para buscar avisos por nombre de empleado en la lista de valores
if (!String.IsNullOrEmpty(strCadenaBusqueda))
{
    avisos = avisos.Where(s => s.Empleado.Nombre.Contains(strCadenaBusqueda));
}

// Para filtrar avisos por tipo de avería
if (intTipoAveriaId == null)
{
    ViewData["TipoAveriaId"] = new SelectList(_context.TipoAverias, "Id",
        "Descripcion");
}
else
{
    ViewData["TipoAveriaId"] = new SelectList(_context.TipoAverias, "Id",
        "Descripcion", intTipoAveriaId);
    avisos = avisos.Where(x => x.TipoAveriaId == intTipoAveriaId);
}

avisos = avisos.Include(a => a.Empleado)
                .Include(a => a.Equipo)
                .Include(a => a.TipoAveria);

int pageSize = 3;
return View(await PaginatedList<Aviso>.CreateAsync(avisos.AsNoTracking(),
    pageNumber ?? 1, pageSize));
// return View(await avisos.AsNoTracking().ToListAsync());

// var mvcSoporteContexto = _context.Avisos.Include(a => a.Empleado).Include(a =>
a.Equipo).Include(a => a.TipoAveria);
// return View(await mvcSoporteContexto.ToListAsync());
}
```

En el código anterior pueden observarse los cambios realizados en el código para incorporar la paginación a la lista de avisos. Estos cambios en el código son similares a los que se realizaron en otro ejercicio realizado anteriormente. También pueden apreciarse los cambios realizados y los parámetros añadidos a la acción *Index()* para poder disponer de los valores de búsqueda y/o de filtrado actual cuando se cambia de página en la lista de avisos, de manera que pueda realizarse la consulta correspondiente para obtener los resultados solicitados.

2. En la vista *Index.cshtml* de la subcarpeta de vistas */Views/Avisos*, modificar o añadir el código que aparece resaltado a continuación.

```
@* @model IEnumerable<MvcSoporte.Models.Aviso> *@
@model PaginatedList<MvcSoporte.Models.Aviso>

@{
    ViewData["Title"] = "Index";
}

<h1>Avisos</h1>
```

```

<p>
  <a asp-action="Create">Nuevo aviso</a>
</p>

@* Agregar un cuadro de búsqueda en la vista Index de Avisos *@
<form asp-action="Index">
  <div class="container">
    <div class="row align-items-start">
      <div class="col-md">
        <label class="control-label">Nombre empleado:</label>
        <input type="text" name="strCadenaBusqueda"
              value="@ViewData["BusquedaActual"]" class="form-control" />
      </div>
      <div class="col-md">
        <label class="control-label">Tipo de avería:</label>
        <select name="intTipoAveriaId" asp-items="ViewBag.TipoAveriaId"
              class="form-control">
          <option value="">Todas</option>
        </select>
      </div>
      <div class="col-md align-self-end">
        <input type="submit" value="Buscar" class="btn btn-default" /> |
        <a asp-action="Index" class="btn btn-default">Lista completa</a>
      </div>
    </div>
  </div>
</form>

<table class="table">
  <thead>
    <tr>
      <th>
        Descripción
        @* @Html.DisplayNameFor(model => model.Descripcion) *@
      </th>
      <th>
        Fecha de aviso
        @* @Html.DisplayNameFor(model => model.FechaAviso) *@
      </th>
      <th>
        Fecha de cierre
        @* @Html.DisplayNameFor(model => model.FechaCierre) *@
      </th>
      <th>
        @*
        @Html.DisplayNameFor(model => model.Observaciones)
        *@
      </th>
      <th>
        Empleado
        @* @Html.DisplayNameFor(model => model.Empleado) *@
      </th>
      <th>
        Tipo de avería
        @* @Html.DisplayNameFor(model => model.TipoAveria) *@
      </th>
      <th>
        Equipo
        @* @Html.DisplayNameFor(model => model.Equipo) *@
      </th>
    </tr>
  </thead>

```



```
</th></th>
</tr>
</thead>
<tbody>
  @foreach (var item in Model)
  {
    <tr>
      <td>
        @Html.DisplayFor(modelItem => item.Descripcion)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.FechaAviso)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.FechaCierre)
      </td>
      @*
      <td>
        @Html.DisplayFor(modelItem => item.Observaciones)
      </td>
      *@
      <td>
        @Html.DisplayFor(modelItem => item.Empleado.Nombre)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.TipoAveria.Descripcion)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.Equipo.CodigoEquipo)
      </td>
      <td>
        <a asp-action="Edit" asp-route-id="@item.Id">Editar</a> |
        <a asp-action="Details" asp-route-id="@item.Id">Detalles</a> |
        <a asp-action="Delete" asp-route-id="@item.Id">Eliminar</a>
      </td>
    </tr>
  }
</tbody>
</table>

@* Poner botones de página Siguiente y Anterior *@
@{
  var prevDisabled = !Model.HasPreviousPage ? "disabled" : "";
  var nextDisabled = !Model.HasNextPage ? "disabled" : "";
}
<a asp-action="Index" asp-route-pageNumber="@((Model.PageIndex - 1))"
  asp-route-busquedaActual="@ViewData["BusquedaActual"]"
  asp-route-tipoAveriaIdActual="@ViewData["TipoAveriaIdActual"]"
  class="btn btn-outline-secondary @prevDisabled">
  Anterior
</a>
<a asp-action="Index" asp-route-pageNumber="@((Model.PageIndex + 1))"
  asp-route-busquedaActual="@ViewData["BusquedaActual"]"
  asp-route-tipoAveriaIdActual="@ViewData["TipoAveriaIdActual"]"
  class="btn btn-outline-secondary @nextDisabled">
  Siguiente
</a>
```

En el código anterior, puede apreciarse que la directiva `@model` especifica que la vista obtiene un objeto `PaginatedList<T>` en lugar de un objeto `IEnumerable<T>`. También se observan los cambios realizados en el código debido a que el objeto `PaginatedList<T>` no admite el uso del HTML Helper `@Html.DisplayNameFor()`. Finalmente, puede apreciarse la especificación de los elementos de navegación para avanzar y retroceder de página en la lista de avisos.

3. Iniciar la depuración para comprobar los resultados obtenidos.

MvcSoporte Inicio Privacidad Avisos Equipos Localizaciones Averías Empleados Register Login

## Avisos

[Nuevo aviso](#)

Nombre empleado:

Tipo de avería:

Buscar | [Lista completa](#)

Descripción	Fecha de aviso	Fecha de cierre	Empleado	Tipo de avería	Equipo
No arranca	21/06/2019		Joan Beltrá i Lloret	No arranca el sistema	PC20167 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>
No se pone en marcha el ordenador	01/05/2019	04/05/2019	Josep Joan Torró i Falcó	No arranca el sistema	PC23917 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>
No imprime bien la impresora	01/05/2019	01/05/2019	Joan Beltrá i Lloret	La impresora no tiene tinta	PR28762 <a href="#">Editar</a>   <a href="#">Detalles</a>   <a href="#">Eliminar</a>

Anterior

Siguiente

© 2022 - MvcSoporte - IES Mare Nostrum. DAW-DWS - [Política de privacidad](#)

## Ejercicio 4

### Procesamiento de tipo Maestro-Detalle

En este ejercicio se aborda el desarrollo de un procesamiento típico denominado Maestro-Detalle o Cabecera-Líneas. Este procesamiento suele formar parte de algunos procesamientos de tipo CRUD en las aplicaciones informáticas. A continuación, se va a desarrollar un procesamiento Maestro-Detalle para presentar, en una misma página, los avisos recibidos de un determinado equipo. Este procesamiento se va a implementar en la opción *Details* del procesamiento CRUD de la tabla *Equipo*. Además, en este ejercicio se incluirá una opción para poder imprimir el resultado del procesamiento.

#### Desarrollar un procesamiento de tipo Maestro-Detalle

En un procesamiento de tipo Maestro-Detalle existe una tabla maestra y una tabla de detalle. Para poder implementar este tipo de procesamiento debe existir una relación 1:N entre la tabla maestra y la tabla detalle. En este ejercicio se va a desarrollar un procesamiento Maestro-Detalle para presentar los avisos relacionados con un determinado equipo. En este caso, la tabla maestra es *Equipo* y la tabla de detalle es *Aviso*. Se modificará el código de la acción *Details()* del controlador *EquiposController.cs*, así como el código de la vista correspondiente. Para ello, realizar las siguientes acciones:

1. Abrir el archivo *EquiposController.cs* situado en la carpeta */Controllers*. Y, a continuación, en la acción *Details()*, agregar el código que aparece resaltado a continuación.

```
// GET: Equipos/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null || _context.Equipos == null)
    {
        return NotFound();
    }

    var equipo = await _context.Equipos
        .Include(e => e.Localizacion)
        .Include(e => e.Avisos)
        .ThenInclude(a => a.Empleado)
        .Include(e => e.Avisos)
        .ThenInclude(b => b.TipoAveria)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (equipo == null)
    {
        return NotFound();
    }

    return View(equipo);
}
```

En el código anterior puede apreciarse el uso del método *Include()* para incluir los datos relacionados directamente con *Equipos*, así como el uso del método *ThenInclude()* para incluir los datos relacionados con la entidad de datos incluida previamente mediante *Include()*.

2. Abrir la vista *Details.cshtml* de la subcarpeta de vistas */Views/Equipos*. Y, a continuación, modificar o añadir el código que aparece resaltado a continuación.

```
@model MvcSoporte.Models.Equipo

@{
    ViewData["Title"] = "Details";
}

<h1>Detalles</h1>

<div>
    <h4>Equipo</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.CodigoEquipo)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.CodigoEquipo)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Marca)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Marca)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Modelo)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Modelo)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.NumeroSerie)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.NumeroSerie)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Caracteristicas)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Caracteristicas)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Precio)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Precio)
        </dd>
        @*
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.PrecioCadena)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.PrecioCadena)
        </dd>
        *@
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.FechaCompra)
        </dt>
        <dd class="col-sm-10">
```

```
@Html.DisplayNameFor(model => model.FechaCompra)
</dd>
<dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.FechaBaja)
</dt>
<dd class="col-sm-10">
    @Html.DisplayNameFor(model => model.FechaBaja)
</dd>
<dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.Localizacion)
</dt>
<dd class="col-sm-10">
    @Html.DisplayNameFor(model => model.Localizacion.Descripcion)
</dd>
</dl>
</div>

@* Se añade código para presentar los avisos del equipo seleccionado *@
@{ var contador = 0; }
<div>
    <h4>Avisos recibidos del equipo</h4>
    <table class="table border-bottom">
        <tr>
            <th>Descripción</th>
            <th>Fecha Aviso</th>
            <th>Fecha Cierre</th>
            <th>Empleado</th>
            <th>Tipo de avería</th>
            <th></th>
        </tr>
        @foreach (var item in Model.Avisos)
        {
            <tr>
                <td>
                    @Html.DisplayNameFor(modelItem => item.Descripcion)
                </td>
                <td>
                    @Html.DisplayNameFor(modelItem => item.FechaAviso)
                </td>
                <td>
                    @Html.DisplayNameFor(modelItem => item.FechaCierre)
                </td>
                <td>
                    @Html.DisplayNameFor(modelItem => item.Empleado.Nombre)
                </td>
                <td>
                    @Html.DisplayNameFor(modelItem => item.TipoAveria.Descripcion)
                </td>
                <td>
                    <div class="d-print-none">
                        <a asp-action="Edit" asp-controller="Avisos"
                          asp-route-id="@item.Id">Editar</a> |
                        <a asp-action="Delete" asp-controller="Avisos"
                          asp-route-id="@item.Id">Eliminar</a>
                    </div>
                </td>
            </tr>
            contador = contador + 1;
        }
    </table>
```

```

</div>

<p class="text-center">Número de avisos: @contador </p>

<div class="d-print-none text-center">
  <a class="btn btn-primary" href="javascript:window.print()">Imprimir</a>
</div>

<div class="d-print-none">
  @* <a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a> |
  <a asp-action="Index">Back to List</a> *@
  <a asp-action="Index">Volver</a>
</div>
  
```

En el código anterior puede apreciarse que se ha añadido una estructura de repetición de tipo *foreach* para recorrer y presentar los datos de los avisos del equipo seleccionado en formato tabular. Se recorre la lista de avisos del equipo seleccionado, a cuyos datos se puede acceder directamente a través del modelo que se pasa a la vista y que es una entidad de tipo *Equipo*. Y también puede apreciarse el uso de la variable *contador* para contar el número de avisos.

3. Iniciar la depuración para comprobar los resultados obtenidos seleccionando la opción *Details* de un equipo en el procesamiento de tipo CRUD de Equipos.

MvcSoporte
Inicio
Privacidad
Avisos
Equipos
Localizaciones
Averías
Empleados
Register
Login

## Detalles

### Equipo

<b>Código equipo</b>	PC20167
<b>Marca</b>	HP
<b>Modelo</b>	Pavilion 1023
<b>Número de serie</b>	ZXD2374SL
<b>Características técnicas</b>	Intel i3. 4 GB. HD 512 GB. Nvidia 450
<b>Precio</b>	452,30 €
<b>Fecha de compra</b>	12/08/2013
<b>Fecha de baja</b>	
<b>Localización</b>	Sala de reuniones

### Avisos recibidos del equipo

Descripción	Fecha Aviso	Fecha Cierre	Empleado	Tipo de avería	
No se pone en marcha	14/05/2019		Javier Sogorb Espinosa	No arranca el sistema	<a href="#">Editar</a>   <a href="#">Eliminar</a>
No tiene acceso a Internet	11/02/2017	13/02/2017	Marta Betoret i Masià	No funciona la red o Internet	<a href="#">Editar</a>   <a href="#">Eliminar</a>
No arranca	21/06/2019		Joan Beltrá i Lloret	No arranca el sistema	<a href="#">Editar</a>   <a href="#">Eliminar</a>

Número de avisos: 3

[Volver](#)

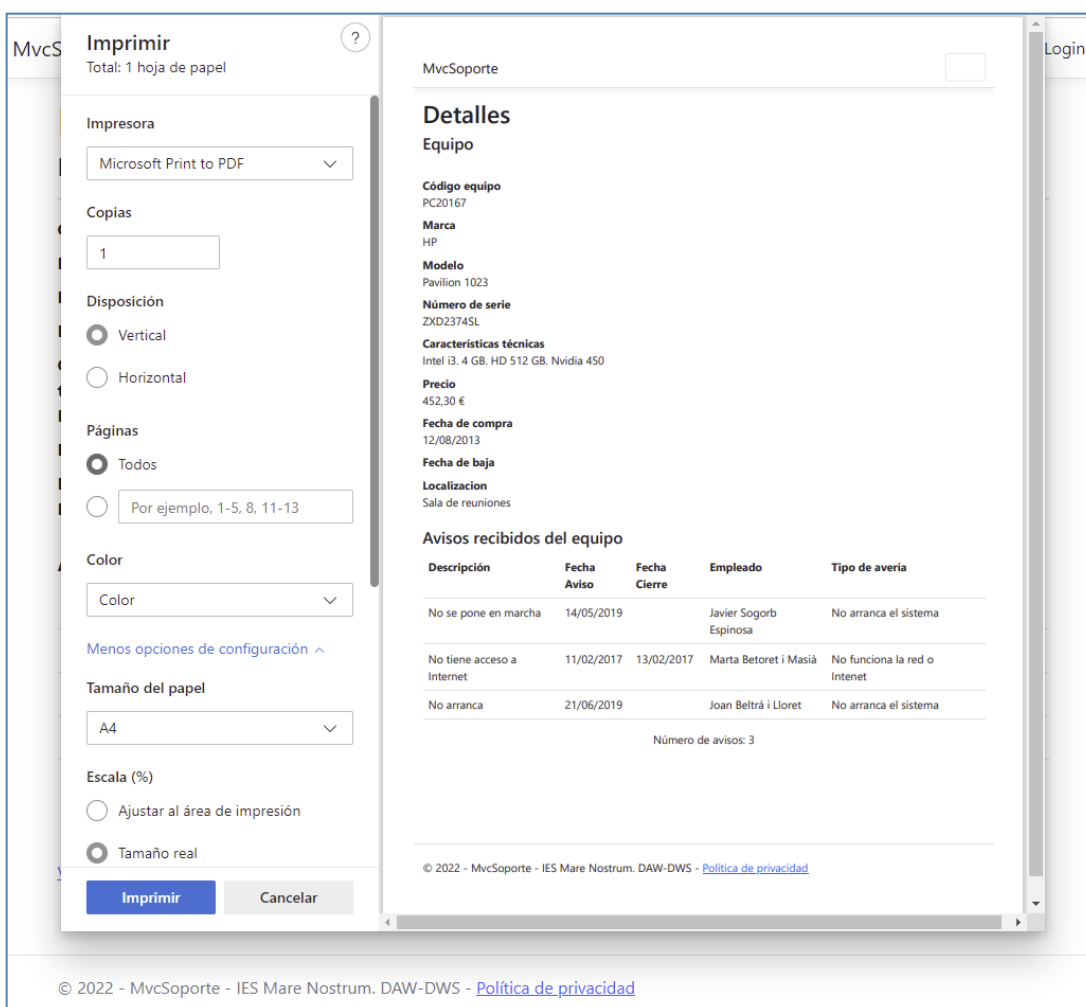
[Imprimir](#)

© 2022 - MvcSoporte - IES Mare Nostrum. DAW-DWS - [Política de privacidad](#)

El procesamiento de tipo Maestro-Detalle desarrollado en este ejercicio puede servir de referencia para poder realizar otros procesamientos similares. Se recomienda analizar y estudiar el código con detenimiento para llegar a comprender todos los aspectos del desarrollo que incluye. Es importante observar y analizar con detenimiento el código para ir comprendiendo su funcionalidad y utilidad.

### Imprimir el resultado de un procesamiento de datos

En el código anterior correspondiente a la vista *Details.cshtml* de la subcarpeta */Views/Equipos*, puede apreciarse que se incorpora un enlace con forma de botón para poder imprimir la página resultante. Para ello, se realiza la llamada al método *print()* del objeto *window* del modelo de objetos del documento o DOM (*Document Model Objects*), mediante lenguaje javascript. Igualmente, puede apreciarse que se agrega la clase de estilo *d-print-none* de *Bootstrap* para evitar que se impriman determinados elementos de la página.



The screenshot shows a web browser window with a print dialog open on the left and a details page on the right. The print dialog is titled 'Imprimir' and shows 'Total: 1 hoja de papel'. It includes options for 'Impresora' (Microsoft Print to PDF), 'Copias' (1), 'Disposición' (Vertical), 'Páginas' (Todos), 'Color' (Color), 'Tamaño del papel' (A4), and 'Escala (%)' (Tamaño real). The details page is titled 'Detalles Equipo' and shows information for a computer (PC20167, HP Pavilion 1023). It includes technical specifications, price (452,30 €), purchase date (12/08/2013), and a table of received reports.

Descripción	Fecha Aviso	Fecha Cierre	Empleado	Tipo de avería
No se pone en marcha	14/05/2019		Javier Sogorb Espinosa	No arranca el sistema
No tiene acceso a Internet	11/02/2017	13/02/2017	Marta Betoret i Masià	No funciona la red o Internet
No arranca	21/06/2019		Joan Beltrà i Lloret	No arranca el sistema

Número de avisos: 3

© 2022 - MvcSoporte - IES Mare Nostrum. DAW-DWS - [Política de privacidad](#)

El procesamiento de impresión ilustrado en este ejercicio puede servir de referencia para poder realizar otros procesamientos similares, a la hora de imprimir los resultados visuales obtenidos en una página, como resultado de la ejecución de una vista.



## Ejercicio 5

# Gestión de la autenticación y autorización de los usuarios a una Aplicación Web basada en ASP.NET Core MVC

En este ejercicio se aborda el desarrollo de los procesamientos que van a permitir gestionar la autenticación y autorización de usuarios en una aplicación Web basada en ASP.NET Core MVC. Para ello, se utilizará la aplicación Web *MvcSoporte* en la que se definirán dos roles de usuario: Usuario y Administrador. Cada uno de estos roles de usuario definen el caso o tipo de uso y, por tanto, establecen los procesamientos que podrá ejecutar un usuario autenticado cuando acceda a la aplicación Web, dependiendo del rol al cual pertenezca. La realización de este ejercicio se basa en la utilización del sistema de autenticación y autorización predeterminado denominado *ASP.NET Core Identity*.

### Sistema de autenticación y autorización de usuarios de ASP.NET Core (ASP.NET Core Identity)

De manera predeterminada, los proyectos de ASP.NET Core MVC que han sido creados utilizando el tipo de autenticación de **Cuentas individuales** incorporan un contexto de datos específico para almacenar y administrar la información relativa a los usuarios registrados en la aplicación Web. Además, al seleccionar esta opción de autenticación, la aplicación Web también incorpora, de manera predeterminada, la implementación básica del procesamiento del registro y del acceso de los usuarios.

El sistema de registro, autenticación y autorización de usuarios predeterminado para las aplicaciones Web de ASP.NET Core utiliza la biblioteca de clases *ASP.NET Core Identity*. Esta biblioteca de clases agrega las funcionalidades de registro e inicio de sesión de los usuarios a la interfaz de usuario de las aplicaciones Web basadas en ASP.NET Core. Algunas de sus principales características son:

- Puede utilizarse tanto con ASP.NET Razor Pages como con ASP.NET Core MVC.
- Forma parte integrada de las plantillas de proyecto disponibles de Visual Studio para las aplicaciones basadas en la tecnología ASP.NET Core.
- Facilita la autenticación y autorización de usuarios.
- Permite personalizar las aplicaciones Web mediante el uso de roles o perfiles de usuario.
- Incorpora un conjunto de procesamientos predeterminados para: gestionar el inicio y cierre de la sesión, registrar un usuario, cambiar la contraseña, iniciar sesión con OAuth utilizando credenciales de proveedores de autenticación externos (Facebook, Twitter, Google, etc.), verificar la cuenta de usuario en dos pasos mediante correo electrónico o SMS, etc.
- Utiliza un contexto de datos específico para representar la información de los usuarios y los roles de la aplicación Web. Se utiliza una cadena de conexión para especificar el archivo de base de datos que almacenará la información de usuarios y roles de usuario de la aplicación.
- Proporciona un sistema de acceso de usuarios seguro, que incluye funcionalidades avanzadas de registro, autenticación y autorización de usuarios.

La utilización de *ASP.NET Core Identity* permite implementar funcionalidades tanto de autenticación como de autorización de usuarios en una aplicación Web. La autenticación es el proceso que permite determinar la identidad de un usuario para averiguar si el usuario es quién dice ser. La autorización consiste en determinar si un usuario puede tener acceso a un determinado procesamiento o recurso.

La biblioteca de clases *ASP.NET Core Identity* da soporte a la implementación de la interfaz de usuario que permite gestionar el acceso de usuarios a una aplicación Web basada en ASP.NET Core mediante sesiones de usuario. Y, además, proporciona una API (*Application program Interface*) que permite registrar nuevos usuarios y administrar usuarios, contraseñas, datos de perfil, roles, notificaciones, confirmación de correo electrónico, etc.

Los usuarios pueden crearse una cuenta con la información de inicio de sesión y *ASP.NET Core Identity* se encarga de gestionar el almacenamiento en su propio sistema, o bien, puede utilizar un proveedor de inicio de sesión externo. Entre los proveedores de inicio de sesión externos admitidos se incluyen Facebook, Google, Microsoft y Twitter. El almacenamiento de la información relativa al inicio de sesión de los usuarios administrados propiamente por *ASP.NET Core Identity*, se configura mediante una base de datos SQL Server que almacena los nombres de usuario, las contraseñas, los roles, etc. O como alternativa, también se puede utilizar un almacén persistente, por ejemplo, *Azure Table Storage*.

En este ejercicio, se utilizará *ASP.NET Core Identity* para facilitar a los usuarios que puedan registrarse en la aplicación *MvcSoporte*. Además, los usuarios ya registrados podrán autenticarse, de manera que puedan iniciar y cerrar sesión en la aplicación Web. Y, además, se dispondrá de un sistema de autorización seguro que permitirá la ejecución o no de los procesamientos de la aplicación Web, considerando el rol al cual pertenezca el usuario que los solicita.

Cuando se utiliza *ASP.NET Core Identity* para gestionar el registro, la autenticación y la autorización de usuarios, se suelen utilizar **dos contextos de datos diferentes en la misma aplicación Web**. Por un lado, el modelo especificado en el contexto de datos que representa la información que maneja la aplicación Web. Y, por otro, el contexto que representa la información relativa a las cuentas de los usuarios registrados y sus roles.

De manera predeterminada, el modelo definido en el contexto de datos que representa la información de los usuarios y sus roles se denomina ***ApplicationDbContext***. En la aplicación Web *MvcSoporte* ya se dispone, además, del modelo especificado en el contexto ***MvcSoporteContexto*** que representa a los datos que maneja la aplicación Web relativos a los avisos de soporte informático que realizan los empleados. De modo que, en la aplicación Web *MvcSoporte* se utilizarán los modelos especificados en los contextos de datos *MvcSoporteContexto* y *ApplicationDbContext*. Sin embargo, la información que representan ambos modelos se almacenará en la misma base de datos, porque ambos contextos utilizan la cadena de conexión ***DefaultConnection*** para enlazar la aplicación Web con la base de datos, tal como puede observarse en los registros de los dos contextos de datos, que están definidos en el archivo de inicio *Program.cs*, así como en la configuración definida en el archivo *appsettings.json*.

### **Migración inicial del modelo especificado en el contexto de datos *ApplicationDbContext***

Para poder implementar un sistema de registro, autenticación y autorización de usuarios basado Roles empleando la biblioteca de clases *ASP.NET Core Identity* es necesario:

- Haber creado un proyecto de aplicación Web de ASP.NET Core MVC con tipo de autenticación basada en **Cuentas individuales**.
- Haber realizado una **migración inicial del contexto *ApplicationDbContext***, que representa la información sobre usuarios y roles, para inicializar la base de datos.

La aplicación Web *MvcSoporte* incluye las características y funcionalidades de *ASP.NET Core Identity*, porque se seleccionó la opción de autenticación **Cuentas individuales** al crear la aplicación Web. Sin embargo, no se ha realizado la migración inicial del contexto *ApplicationDbContext*, que representa la información relativa a usuarios y roles, para crear las tablas correspondientes de *ASP.NET Core Identity* y, de este modo, inicializar la base de datos completamente.

A continuación, es necesario realizar una migración inicial del contexto de datos *ApplicationDbContext* para inicializar las tablas correspondientes a la gestión de usuarios y roles en la base de datos de la aplicación Web. Para ello, ejecutar el siguiente comando en la **Consola del Administrador de paquetes**.

```
PM> Update-database -context ApplicationDbContext
```

En el caso de la migración inicial del contexto *ApplicationDbContext*, no se necesita ejecutar el comando *Add-Migration* previamente, porque ya se generó el archivo de migración correspondiente en la carpeta */Data*, cuando se creó la aplicación Web basada en autenticación de Cuentas individuales.

### Probar el registro y el inicio de una sesión de usuario

Para comprobar el funcionamiento del sistema predeterminado de acceso de usuarios de *ASP.NET Core Identity* registrar un nuevo usuario en la aplicación Web *MvcSoporte* del siguiente modo:

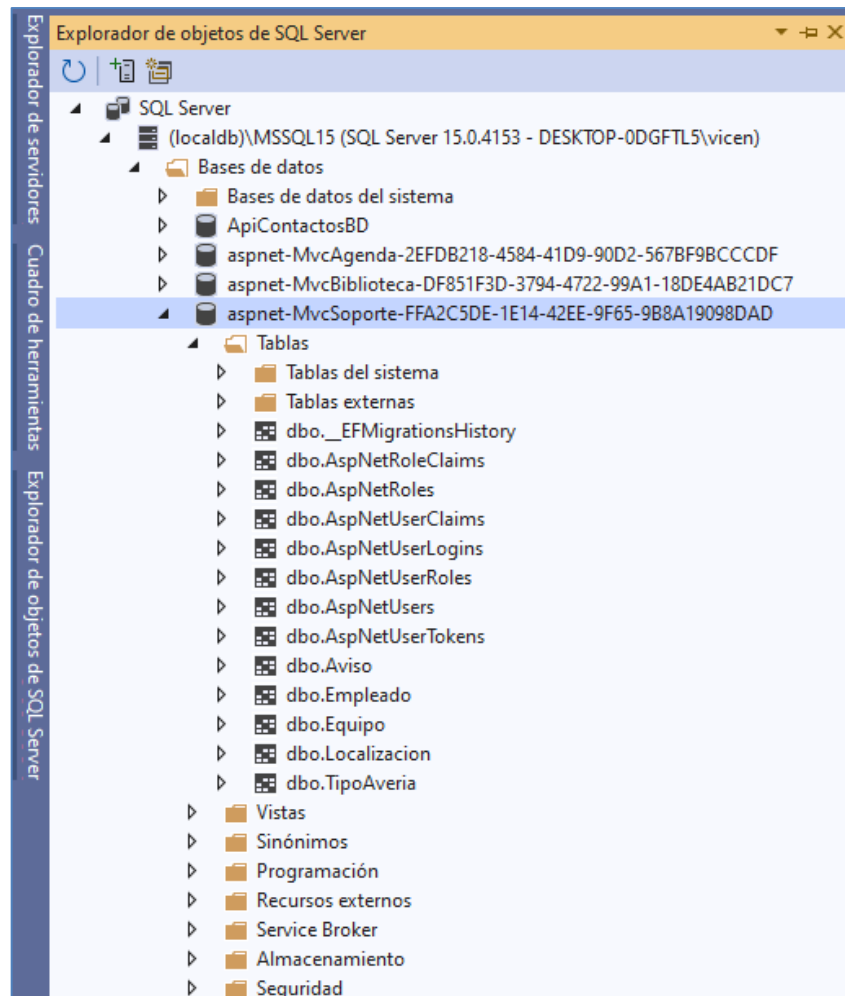
1. Iniciar la depuración de la aplicación Web.
2. Para registrar un usuario, seleccionar la opción **Register** situada en la barra de navegación situada en la parte superior de la ventana de visualización.
3. Introducir el correo electrónico del usuario y una contraseña adecuada dos veces. Por ejemplo, registrar el usuario "prueba@empresa.com" con la contraseña: "Prueba-123". A continuación, se debe confirmar el registro de la cuenta, haciendo clic en el enlace correspondiente, cuyo texto original se muestra en inglés: "*Click here to confirm your account*". La confirmación de la cuenta de usuario tiene por objeto comprobar la identidad del usuario durante el proceso de registro. Esta confirmación se realiza mediante el método denominado en dos pasos que utiliza el envío de un correo o un SMS de confirmación. Sin embargo, en tiempo de desarrollo puede realizarse de forma simplificada, haciendo clic en el enlace indicado anteriormente.
4. Acceder a la cuenta del usuario registrado. Para ello, seleccionar la opción **Login** situada en la barra de navegación y, a continuación, introducir su correo electrónico y contraseña.
5. Comprobar que el correo del usuario "prueba@empresa.com" se presenta visualmente en la barra de navegación superior, lo que indica que la sesión del usuario está activa.
6. Cerrar la sesión de este usuario, haciendo clic en la opción **Logout**. Y finalizar la depuración.

### Comprobar la base de datos y las tablas del sistema de acceso de usuarios

A continuación, se va a comprobar que la migración inicial realizada anteriormente ha creado las tablas definidas en el contexto de datos *ApplicationDbContext* en la base de datos. Estas tablas almacenan la información de los usuarios y roles de la aplicación Web. Para ello, realizar las siguientes acciones:

1. En el **Explorador de objetos de SQL Server** de Visual Studio, desplegar la opción **SQL Server**. A continuación, desplegar la opción **(localdb)\MSSQL15** y la opción **Bases de datos**.

2. Para acceder a los objetos de la base de datos de la aplicación Web, desplegar la opción que correspondiente al nombre de la base de datos **aspnet-MvcSoporte-XXXXXX** o similar, que está definido en el archivo *appsettings.json* mediante la propiedad *Database* de la conexión.
3. Desplegar la opción **Tablas** para ver todas las tablas de la base de datos. Puede observarse que los nombres de las tablas del contexto *ApplicatioDbContext* comienzan por el prefijo *AspNet*.



4. Para visualizar la información de los usuarios registrados, seleccionar la tabla *AspNetUsers*, hacer clic con el botón derecho y seleccionar la opción **Ver datos**. Puede observarse que los datos de los campos relativos a la identificación y contraseña de usuario aparecen encriptados.

Para resolver los procesamiento de autenticación y autorización de usuarios basada en roles de una aplicación Web, bastará con utilizar las tres tablas siguientes:

- **AspNetUsers**. Almacena la información de los usuarios de la aplicación Web. Contiene los campos: *Id*, *Email*, *UserName*, *PasswordHash*, *PhoneNumber*, etc. Por razones de seguridad, los valores de los campos que almacenan contraseñas e identificadores están encriptados.

- **AspNetRoles.** Almacena información de los roles. Contiene las columnas *Id* y *Name*.
- **AspNetUserRoles.** Almacena información que relaciona a un usuario con su rol. Contiene las columnas *UserId* y *RoleId* que actúan como claves ajenas de las tablas anteriores. La existencia de esta tabla posibilita que un usuario pueda pertenecer a varios roles que corresponde al caso más general. Sin embargo, este ejercicio se resolverá considerando que un usuario solo podrá pertenecer solo a un rol, lo cual puede realizarse utilizando esta misma estructura de tablas.

También pueden visualizarse el contenido almacenado en las tablas *AspNetRoles* y *AspNetUsersRoles*, aunque están vacías en este momento, porque aún no se ha introducido información en ellas.

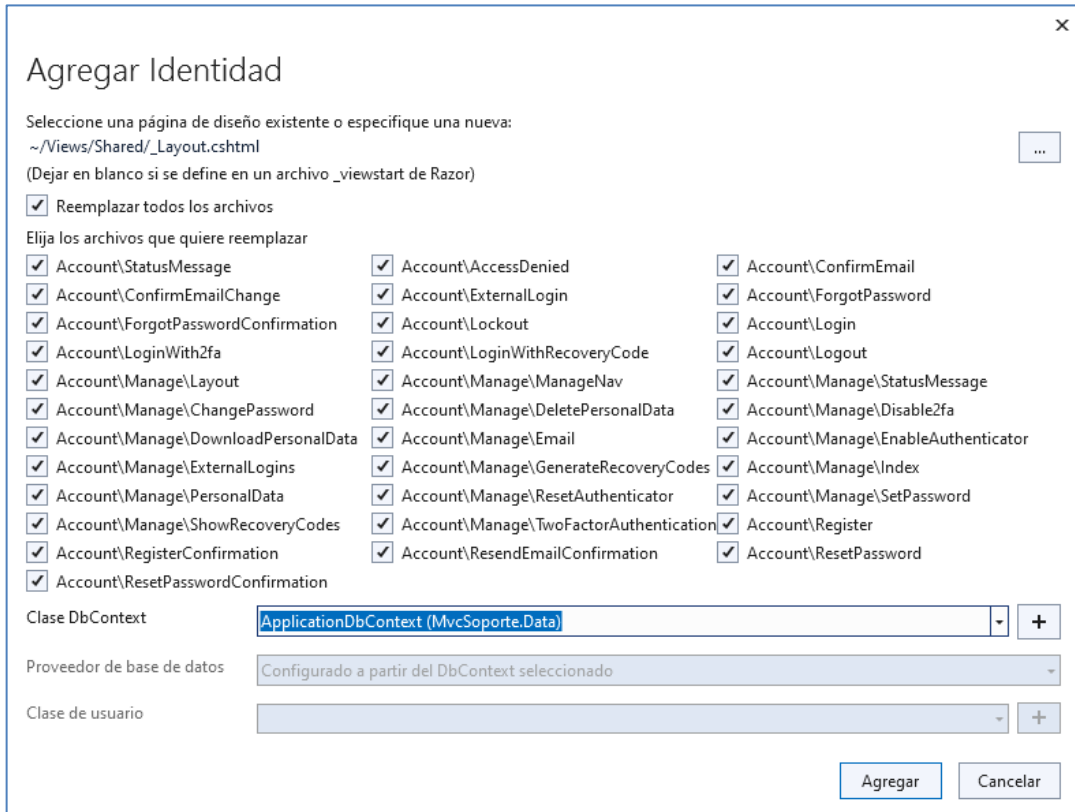
Una vez comprobado el acceso a las tablas de la base de datos que se encargan de almacenar la información relativa a usuarios y roles y también que el sistema predeterminado de autenticación de usuarios de *ASP.NET Core Identity* funciona correctamente, a continuación, en los siguientes apartados del ejercicio, se va a implementar un sistema personalizado de autenticación y autorización de usuarios basado en roles empleando la biblioteca de clases *ASP.NET Core Identity*.

### **Agregar el código de ASP.NET Core Identity a la aplicación Web**

La tecnología ASP.NET Core proporciona el código de *ASP.NET Core Identity* como una biblioteca de clases de Razor Pages. El modelo de desarrollo *Razor Pages* o Páginas de Razor permite crear aplicaciones Web de tipo Modelo-Vista-Vista-Modelo (MVVM). En las aplicaciones Web basadas en ASP.NET Core que vayan a incluir las funcionalidades de *ASP.NET Core Identity*, se puede agregar de forma selectiva el código fuente de la biblioteca de clases de *ASP.NET Core Identity*. De este modo, es posible modificar el código para adaptar el comportamiento predeterminado y obtener un comportamiento personalizado sobre la autenticación y autorización de usuarios, en cada caso concreto. En estos casos, se puede seleccionar el código de *ASP.NET Core Identity* que va a ser modificado para personalizar el comportamiento predeterminado. El código modificado tendrá siempre prioridad sobre el mismo código que forma parte del comportamiento predeterminado.

Para agregar el código fuente de la biblioteca de clases *ASP.NET Core Identity*, de manera que pueda personalizarse su comportamiento en la aplicación Web *MvcSoporte*, realizar las siguientes acciones:

1. En el **Explorador de soluciones**, hacer clic en el botón derecho sobre el proyecto *MvcSoporte*, desplegar la opción **Agregar** y seleccionar la opción **Nuevo elemento con scaffold...**
2. En el panel izquierdo de la ventana **Agregar nuevo elemento con scaffold**, seleccionar la opción **Identidad**, en el panel central seleccionar la opción **Identidad** y hacer clic en **Agregar**.
3. En la ventana **Agregar identidad**, realizar las siguientes acciones:
  - a. Modificar la página de diseño propuesta, para ello hacer clic sobre el botón en el que aparecen tres puntos que está situado a la derecha. A continuación, en la ventana **Seleccionar página de diseño**, elegir la página de diseño existente *\_Layout.cshtml* que está ubicada en la carpeta */Views/Shared* y hacer clic en **Aceptar**.
  - b. De nuevo en la ventana **Agregar identidad**, marcar la casilla **Reemplazar todos los archivos**, de este modo que se tendrá un control total sobre el código.
  - c. En el cuadro combinado **Clase de DbContext**, seleccionar la clase del contexto de datos denominada *ApplicationDbContext (MvcSoporte.Data)*.
  - d. Hacer clic en el botón **Agregar**.



- Tras finalizar el trabajo del asistente de generación de código, podrá comprobarse que se habrá agregado la subcarpeta **/Areas/Identity** dentro de la carpeta **/Areas** del proyecto. Esta carpeta incluye las Páginas de Razor que implementan la interfaz Web y el comportamiento predeterminado de los procesamientos de registro, autenticación y autorización de los usuarios y de los roles de usuario en una aplicación Web de ASP.NET Core que utiliza *ASP.NET Core Identity*. Una vez que están disponibles estas páginas en la aplicación Web, se podrá modificar el código como mejor convenga para personalizar la interfaz de usuario y administrar el comportamiento de la gestión de los usuarios y de los roles de usuario.

#### Configurar el uso de Roles en *ASP.NET Core Identity*. Crear los roles predeterminados y el usuario administrador predeterminado

A continuación, se va a configurar el servicio que posibilita el uso de los usuarios y de los roles o perfiles de usuario en una aplicación Web basada en ASP.NET Core MVC que utiliza *ASP.NET Core Identity*. Y, además, también se va a añadir el código que permita crear los roles predeterminados de la aplicación Web, “Usuario” y “Administrador”, y el administrador predeterminado, [admin@empresa.com](mailto:admin@empresa.com). Para todo ello, se utiliza el archivo de inicio *Program.cs* que está ubicado en la carpeta raíz del proyecto. El archivo de inicio *Program.cs* tiene la particularidad de que su código se ejecuta cada vez que se inicia la ejecución de la aplicación Web, por lo que este archivo se suele utilizar para especificar los servicios y la configuración que se debe aplicar cada vez que se inicie la aplicación Web. Además, como paso previo, se va a crear una clase genérica en el proyecto para que incluya el código que permite crear los roles y el administrador predeterminados de la aplicación Web en el caso de que no existan, lo que facilita la separación funcional del código y aporta claridad al desarrollo.



Para configurar el uso de los roles en la aplicación Web, así como para crear los roles de usuario y el administrador predeterminados, si es que no existen, realizar las siguientes acciones:

1. Como paso previo, en primer lugar, se creará una clase genérica denominada *SeedData.cs* en la carpeta raíz del Proyecto de ASP.NET Core MVC. Para ello, hacer:
  - a. En el **Explorador de soluciones**, hacer clic en botón derecho sobre el nombre del proyecto *MvcSoporte*, desplegar la opción **Agregar** y seleccionar la opción **Clase...**
  - b. Introducir *SeedData.cs* como **Nombre** de la clase y hacer clic en **Agregar**.
  - c. En el archivo de clase *SeedData.cs*, agregar el siguiente código:

```
public class SeedData
{
    public static async Task InitializeAsync(IServiceProvider services)
    {
        // Comprobar y crear los roles predeterminados
        var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
        await CrearRolesAsync(roleManager);

        // Comprobar y crear el administrador predeterminado
        var userManager = services.GetRequiredService<UserManager<IdentityUser>>();
        await CrearAdminAsync(userManager);
    }

    private static async Task CrearRolesAsync(RoleManager<IdentityRole> roleManager)
    {
        // Si no existe, se crea el rol predeterminado "Administrador"
        string nombreRol = "Administrador";
        var yaExiste = await roleManager.RoleExistsAsync(nombreRol);
        if (!yaExiste)
            await roleManager.CreateAsync(new IdentityRole(nombreRol));

        // Si no existe, se crea el rol predeterminado "Usuario"
        nombreRol = "Usuario";
        yaExiste = await roleManager.RoleExistsAsync(nombreRol);
        if (!yaExiste)
            await roleManager.CreateAsync(new IdentityRole(nombreRol));
    }

    private static async Task CrearAdminAsync(UserManager<IdentityUser> userManager)
    {
        // Comprobar si existe el administrador predeterminado
        var testAdmin = userManager.Users
            .Where(x => x.UserName == "admin@empresa.com")
            .SingleOrDefault();

        if (testAdmin != null) return;

        testAdmin = new IdentityUser {
            UserName = "admin@empresa.com",
            Email = "admin@empresa.com"
        };
        string admPasswd = "Admin-123";

        // Si no existe, se crea el administrador predeterminado "admin@empresa.com"
        IdentityResult userResult;
        userResult = await userManager.CreateAsync(testAdmin, admPasswd);
    }
}
```

```
// Se agrega el rol "Administrador" al administrador predeterminado
if (userResult.Succeeded)
{
    await userManager.AddToRoleAsync(testAdmin, "Administrador");
}
}
```

En el código anterior, puede apreciarse que se han especificado dos métodos. El primero de ellos, *CrearRolesAsync()*, permite crear los roles predeterminados en la aplicación Web que se denominarán “Administrador” y “Usuario”, si es que no existen. Para ello, se emplea el método *CreateAsync()* del objeto *roleManager*. Este objeto permite administrar los roles de usuario de la aplicación Web a través del código. El segundo método, *CrearAdminAsync()*, permite crear el administrador predeterminado cuyo nombre de usuario es `admin@empresa.com` y su contraseña es “Admin-123”, en el caso de que no exista. Para ello, emplea el método *CreateAsync()* del objeto *userManager*. Este objeto permite administrar los usuarios de la aplicación Web a través del código. En el código de la clase anterior, debe agregarse el espacio de nombres *Microsoft.AspNetCore.Identity*, mediante la cláusula *using* correspondiente.

2. Abrir el archivo *Program.cs* que está ubicado de la carpeta raíz del Proyecto.
3. En el archivo *Program.cs*, modificar y añadir el código necesario para obtener la especificación de código que se presenta a continuación.

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using MvcSoporte;
using MvcSoporte.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));

// Registrar el contexto de la base de datos
builder.Services.AddDbContext<MvcSoporteContexto>(options =>
    options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

// Deshabilitar confirmación de usuario. Configurar Identity para utilizar roles
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
    options.SignIn.RequireConfirmedAccount = false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseMigrationsEndPoint();
}
```



```
}  
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.UseAuthentication();  
app.UseAuthorization();  
  
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");  
app.MapRazorPages();  
  
// Crear los roles y el administrador predeterminados  
using (var scope = app.Services.CreateScope())  
{  
    var services = scope.ServiceProvider;  
  
    SeedData.InitializeAsync(services).Wait();  
}  
  
app.Run();
```

En el código anterior, puede apreciarse que se configura el servicio de *ASP.NET Core Identity* para poder usar los roles de usuario en la aplicación Web, a través de la llamada al método *AddRoles()*. También puede apreciarse en el código, que se modifica a *false* el valor de la opción *RequireConfirmedAccount*, de manera que ya no será necesario realizar la confirmación de las cuentas de usuario cuando se crean. Finalmente, puede apreciarse en el código anterior, la llamada al método *InitializeAsync()* que utiliza el servicio con ámbito *services* como parámetro. Tal como se ha descrito anteriormente, el método *InitializeAsync()* se ha especificado en el archivo de clase *SeedData.cs* por razones de separación y claridad del código. En el método *InitializeAsync()* se crean, si no existen, los roles de usuario y el administrador predeterminados. El método *Wait()* garantiza que el procesamiento de la aplicación Web esperará hasta que finalice la ejecución del método *InitializeAsync()*.

### Hacer que los nuevos usuarios que se registren en la aplicación Web pertenezcan al rol de Usuario

Además de crear los roles predeterminados de la aplicación Web, se debe hacer que todos los nuevos usuarios que se registren pertenezcan al rol "Usuario". Para ello, se modificará el código del método *OnPostAsync()* de la página *Register.cshtml* del código de *ASP.NET Core Identity*, de la siguiente forma:

1. En el **Explorador de soluciones** desplegar la carpeta */Areas* y, a continuación, desplegar sucesivamente las carpetas: *Identity*, *Pages* y *Account*.

2. Desplegar el archivo *Register.cshtml*, haciendo clic en la flecha existente junto al nombre del archivo y abrir el archivo de la clase *Register.cshtml.cs*.
3. En el método *OnPostAsync()*, añadir las líneas de código resaltadas a continuación.

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");
    ExternalLogins = (await
_signinManager.GetExternalAuthenticationSchemesAsync()).ToList();
    if (ModelState.IsValid)
    {
        var user = CreateUser();

        await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
        await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
        var result = await _userManager.CreateAsync(user, Input.Password);

        if (result.Succeeded)
        {
            // Asignar el rol "Usuario" al registrarse un nuevo usuario
            await _userManager.AddToRoleAsync(user, "Usuario");

            _logger.LogInformation("User created a new account with password.");

            var userId = await _userManager.GetUserIdAsync(user);
            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
        }
    }
    ...
}
```

Como puede apreciarse en el código anterior, se utiliza el método *AddToRoleAsync()* de la clase *userManager<T>*, que pertenece a espacio de nombres de *ASP.NET Core Identity*, para asignar el rol "Usuario" a un usuario nuevo cuando se registra.

### Modificar las características de las contraseñas de los usuarios de la aplicación

Se van a definir unas características concretas para las contraseñas de las cuentas de los usuarios de la aplicación Web. Estas características pueden diferir de las establecidas de forma predeterminada por *ASP.NET Core Identity*. En este caso, se desea que las contraseñas tengan las siguientes características:

- Longitud mínima: 6 caracteres.
- Caracteres válidos: minúsculas y números.
- Caracteres numéricos requeridos.

Las características de las contraseñas de las cuentas de usuario se especifican como opciones en el archivo de inicio *Program.cs*. Para modificar las características de las contraseñas, hacer:

1. Abrir el archivo *Program.cs* ubicado en la carpeta raíz del proyecto.
2. En el archivo *Program.cs*, añadir las líneas de código que aparecen resaltadas a continuación.

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using MvcSoporte;
```

```
using MvcSoporte.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));

// Registrar el contexto de la base de datos
builder.Services.AddDbContext<MvcSoporteContexto>(options =>
    options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

// Deshabilitar confirmación de usuario. Configurar Identity para utilizar roles
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
    options.SignIn.RequireConfirmedAccount = false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddControllersWithViews();

// Configuración de los servicios de ASP.NET Core Identity
builder.Services.Configure<IdentityOptions>(options =>
{
    // Password settings. Configuración de las características de las contraseñas
    options.Password.RequireDigit = true;
    options.Password.RequireLowercase = true;
    //options.Password.RequireNonAlphanumeric = true;
    options.Password.RequireNonAlphanumeric = false;
    //options.Password.RequireUppercase = true;
    options.Password.RequireUppercase = false;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    . . .
```

El código anterior muestra la definición de los valores de las características de las contraseñas.

### Comprobar el funcionamiento de la autenticación de usuarios en la aplicación Web *MvcSoporte*

En este punto, se van a crear nuevos usuarios para poder probar el funcionamiento del sistema de autenticación de usuarios en la aplicación Web. Para ello, realizar las siguientes acciones:

1. Iniciar la depuración de la aplicación Web.
2. Seleccionar la opción **Registrarse** del menú de la aplicación Web para crear los dos nuevos usuarios, cuya información se muestra en la siguiente tabla:

Email (nombre de usuario)	Contraseña
usuario1@empresa.com	"usuario1123"
usuario2@empresa.com	"usuario2123"

Después de registrar el primer usuario, debe cerrarse la sesión para registrar otro usuario.

- Finalizar la depuración de la aplicación Web, una vez registrados los dos nuevos usuarios.
- Empleando el **Explorador de objetos de SQL Server** acceder a la base de datos de la aplicación Web para comprobar la información almacenada en las tablas *AspNetUsers*, *AspNetRoles* y *AspNetUsersRoles*. Deberá comprobarse que los datos almacenados en estas tres tablas son los correctos, considerando las operaciones realizadas. Así, se deberá comprobar que:
  - Se han creado los roles "Administrador" y "Usuario".
  - Se ha creado el administrador predeterminado, denominado `admin@empresa.com`, y que pertenece al rol "Administrador".
  - Se han creados los dos nuevos usuarios, denominados `usuario1@empresa.com` y `usuario2@empresa.com`, y que pertenecen al rol "Usuario".

#### Establecer el menú de la aplicación Web según el rol del usuario que accede a la aplicación Web

A continuación, se actuará sobre la plantilla de diseño de las vistas *\_Layout.cshtml* para establecer un menú diferente en la aplicación Web, según sea el rol al que pertenezca el usuario que inicia la sesión. La siguiente tabla muestra las opciones del menú de la aplicación y las acciones del controlador que podrán iniciar los usuarios, atendiendo al rol al cual pertenezcan.

Opción de menú	Rol del usuario	Controlador	Método de acción
Inicio	Cualquiera	<i>HomeController.cs</i>	<i>Index()</i>
Privacidad	Usuario	<i>HomeController.cs</i>	<i>Privacy()</i>
Avisos	Administrador	<i>AvisosController.cs</i>	<i>Index()</i>
Equipos	Administrador	<i>EquiposController.cs</i>	<i>Index()</i>
Localizaciones	Administrador	<i>LocalizacionesController.cs</i>	<i>Index()</i>
Averías	Administrador	<i>TipoAveriasController.cs</i>	<i>Index()</i>
Empleados	Administrador	<i>EmpleadosController.cs</i>	<i>Index()</i>

Para especificar el menú de la aplicación Web considerando el rol del usuario que se autentifique en la aplicación Web, realizar las siguientes acciones:

- Abrir el archivo de vista *\_Layout.cshtml* que está ubicado la carpeta */Views/Shared*.
- En el archivo *\_Layout.cshtml* modificar y añadir las líneas de código resaltadas a continuación.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - MvcSoporte</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />

```

```
<link rel="stylesheet" href="~/MvcSoporte.styles.css" asp-append-version="true" />
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light
      bg-white border-bottom box-shadow mb-3">
      <div class="container-fluid">
        <a class="navbar-brand" asp-area="" asp-controller="Home"
          asp-action="Index">MvcSoporte</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
          data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
          <ul class="navbar-nav flex-grow-1">

            @if (User.Identity.IsAuthenticated)
            {
              @if (User.IsInRole("Usuario"))
              {
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area=""
                    asp-controller="Home" asp-action="Privacy">Privacidad</a>
                </li>
              }
              @if (User.IsInRole("Administrador"))
              {
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area=""
                    asp-controller="Avisos" asp-action="Index">Avisos</a>
                </li>
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area=""
                    asp-controller="Equipos" asp-action="Index">Equipos</a>
                </li>
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area=""
                    asp-controller="Localizaciones" asp-action="Index">Localizaciones</a>
                </li>
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area=""
                    asp-controller="TipoAverias" asp-action="Index">Averias</a>
                </li>
                <li class="nav-item">
                  <a class="nav-link text-dark" asp-area=""
                    asp-controller="Empleados" asp-action="Index">Empleados</a>
                </li>
              }
            }
            else
            {
              <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
                  asp-controller="Home" asp-action="Index">Inicio</a>
              </li>
            }

          </ul>
          <partial name="LoginPartial" />
        </div>
      </div>
    </nav>
  </header>
  <div class="main">
    @await Component.InvokeAsync("Index")
  </div>
</body>
</html>
```

```
</div>
</div>
</nav>
</header>
<div class="container">
  <main role="main" class="pb-3">
    @RenderBody()
  </main>
</div>

<footer class="border-top footer text-muted">
  <div class="container">
    &copy; 2022 - MvcSoporte - IES Mare Nostrum. DAW-DWS -
    <a asp-area="" asp-controller="Home"
      asp-action="Privacy">Política de privacidad</a>
  </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Como puede comprobarse en el código anterior, los usuarios pertenecientes al rol de Usuario solo podrán iniciar la opción Privacidad desde el menú de la aplicación Web. En apartados posteriores, se irán añadiendo más opciones y funcionalidades al perfil de Usuario. En el código anterior también puede observarse el uso que se hace del objeto *User*. Este objeto representa la información correspondiente al usuario actual, es decir, representa al usuario que está autenticado en este momento en la aplicación Web. El objeto *User* proporciona propiedades y métodos que permiten acceder y actuar sobre la información relativa al usuario actual. Por ejemplo, se observa el uso del método *IsInRole()* para comprobar el rol del usuario actual.

3. Para comprobar que los accesos de los usuarios son los adecuados según su rol, iniciar la depuración y realizar las pruebas de acceso necesarias con usuarios pertenecientes al rol de Usuario y al rol de Administrador. Si fuera necesario se pueden registrar nuevos usuarios.

### Evitar accesos indebidos a las acciones de la Aplicación Web

En las aplicaciones Web, la seguridad tiene dos partes bien diferenciadas: la autenticación y la autorización de usuarios. En los apartados anteriores se han resuelto diversos aspectos relacionados con el acceso de usuarios a la aplicación, es decir, con la autenticación de usuarios. En este apartado se resuelven aspectos relacionados la autorización de usuarios. La autorización se refiere a establecer permisos a los usuarios, de modo que solo puedan ejecutar los procesamientos de la aplicación Web que le correspondan, en este caso, dependiendo del rol al que pertenezca cada usuario.

La forma de evitar accesos indebidos de usuarios a los procesamientos de la aplicación Web, según el rol al que pertenecen, es haciendo uso del filtro *Authorize*. Este filtro pertenece permite proteger la solicitud de los métodos de acción de los controladores, de modo que solo puedan ser iniciados por los usuarios que pertenezcan a un rol determinado. Así, por ejemplo, puede observarse en el siguiente código que la acción *Index()* del controlador *TipoAveriasController.cs* se iniciará solo cuando la solicitud provenga de un usuario autenticado que pertenezca al rol Administrador.

```
// GET: TipoAverias
[Authorize(Roles = "Administrador")]
public async Task<IActionResult> Index()
{
    return View(await _context.TipoAverias.ToListAsync());
}
```

El filtro *Authorize* pertenece al espacio de nombres *Microsoft.AspNetCore.Authorization*. Para realizar una prueba de funcionamiento del filtro *Authorize* del código anterior, se puede añadir el código anterior e iniciar la depuración. Al realizar la solicitud <http://localhost:XXXX/TipoAverias/Index> en la barra de direcciones del navegador, se podrá comprobar que solo se iniciará la ejecución de la acción *Index()* del controlador *TipoAverias* si la solicitud la realiza un usuario perteneciente al rol de Administrador. Si no se ha iniciado sesión o bien, si se ha iniciado sesión un usuario que pertenece al rol de Usuario, entonces la acción *Index()* del controlador *TipoAveriasController.cs* no se inicia y la ejecución se redirige hacia el inicio de sesión o bien, hacia una página de acceso denegado.

El filtro *Authorize* se aplica a los métodos de acción, aunque también puede aplicarse a las clases *Controller*. De manera que, si todos los métodos de acción de un mismo controlador deben ser ejecutados solo por los usuarios pertenecientes a un mismo rol de usuario, entonces el filtro *Authorize* correspondiente puede aplicarse a la clase que define el controlador. Y, por lo tanto, la autorización afectará a todas las acciones del controlador.

En el siguiente ejemplo se muestra la aplicación de un filtro *Authorize* a la clase de controlador *TipoAveriasController*. De esta manera, la autorización aplicada afectaría a todas las acciones que tiene definidas este controlador.

```
namespace MvcSoporte.Controllers
{
    [Authorize(Roles = "Administrador")]
    public class TipoAveriasController : Controller
    {
        private readonly MvcSoporteContexto _context;

        public TipoAveriasController(MvcSoporteContexto context)
        {
            _context = context;
        }

        // GET: TipoAverias
        public async Task<IActionResult> Index()
        {
            return View(await _context.TipoAverias.ToListAsync());
        }
    }
    . . .
}
```

Para incluir un sistema de autorización de usuarios basados en roles en la aplicación Web *MvcSoporte*, realizar las siguientes acciones:

1. Abrir los controladores correspondientes y modificar el código para incluir los filtros *Authorize* necesarios a nivel de acción o de clase del controlador, que se indican en la siguiente tabla:



Rol	Controlador	Método de acción
Administrador	Clase <i>AvisosController</i>	-
Administrador	Clase <i>EquiposController</i>	-
Administrador	Clase <i>LocalizacionesController</i>	-
Administrador	Clase <i>TipoAveriasController</i>	-
Administrador	Clase <i>EmpleadosController</i>	-
Usuario	<i>HomeController</i>	<i>Privacy()</i>

2. Iniciar la depuración de la aplicación Web y realizar las pruebas necesarias para comprobar que la autorización establecida en los controladores es correcta, considerando las opciones de menú especificadas anteriormente para cada rol de usuario: Usuario y Administrador.

### Enlazar la información de los usuarios registrados con la información almacenada en la base de datos que maneja la aplicación Web

Para continuar con el desarrollo de la gestión de la autenticación y la autorización de usuarios en la aplicación Web *MvcSoporte*, es necesario poder establecer un enlace o vínculo entre los usuarios de la aplicación Web y los empleados que gestionan los avisos del soporte informático. El enlace entre ambos contextos de datos se realizará a través de un valor común. Concretamente, este enlace se va a realizar a través del campo *Email* de la tabla *AspNetUsers* del contexto de datos que representa la información de los usuarios, *ApplicatioDbContext*, y del campo *Email* de la tabla *Empleado* del contexto de datos que representa la información que maneja la aplicación Web, *MvcSoporteContexto*.

Por otra parte, para acceder a la aplicación Web se usa el valor del *Email* de la tabla *AspNetUsers*. La clase *User.Identity* de *ASP.NET Core Identity* dispone de la propiedad **Name** que devuelve el nombre de usuario actual. Es decir, devuelve el valor del *Email* del usuario que se ha autenticado y que, por tanto, está utilizando la aplicación Web en la sesión actual. El uso de esta propiedad permite al desarrollador acceder al valor del *Email* del usuario actual desde el código en cualquier momento.

A continuación, se va a establecer el enlace entre los usuarios de la aplicación Web y los empleados que gestionan el soporte informático, empleando para ello el valor común *Email* disponible en los dos contextos de datos y la propiedad *User.Identity.Name*. Para ello, realizar las siguientes acciones:

1. En la acción *Index()* del controlador *Home* se va a incluir el código necesario para hacer que, si no existe el empleado correspondiente al usuario actual, entonces se redirija la ejecución hacia el procesamiento que permite al usuario almacenar sus propios datos. Para ello, abrir el archivo *HomeController.cs* de la carpeta */Controllers* y añadir las líneas de código que aparecen resaltadas a continuación, para establecer la inyección de dependencia del contexto de datos que maneja la aplicación Web, *MvcSoporteContexto*, en el controlador *Home*.

```
namespace MvcSoporte.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        private readonly MvcSoporteContexto _context;
```



```
public HomeController(ILogger<HomeController> logger, MvcSoporteContexto context)
{
    _logger = logger;
    _context = context;
}

public IActionResult Index()
{
    // Busca el empleado correspondiente al usuario actual. Si existe, activa la
    // vista (View) y en caso contrario, se redirige para crear el empleado.
    string? emailUsuario = User.Identity.Name;
    Empleado? empleado = _context.Empleados.Where(e => e.Email == emailUsuario)
        .FirstOrDefault();
    if (User.Identity.IsAuthenticated &&
        User.IsInRole("Usuario") &&
        empleado == null)
    {
        return RedirectToAction("Create", "MisDatos");
    }

    return View();
}

...

```

En el código anterior puede apreciarse que se hace uso de la propiedad *Name* de la clase *User.Identity* para obtener el valor del *Email* del usuario actual. También puede apreciarse, el uso que se hace de los métodos *Where()* y *FirstOrDefault()* de *LINQ To Entities* para poder seleccionar los datos del empleado correspondiente al usuario actual. En el código también puede observarse, que si un usuario autenticado que accede a la aplicación Web, además, pertenece al rol de Usuario y, además, no tiene un empleado relacionado a través del Email, entonces la ejecución se redirigirá a la acción *Create()* del controlador *MisDatos*. Esta acción permite al usuario actual, que pueda introducir sus propios datos como empleado. Deberán añadirse al controlador *HomeController.cs* las cláusulas *using* necesarias. Es importante analizar con detenimiento el código para comprender bien su uso, funcionalidad y utilidad.

2. El siguiente paso es crear el controlador *MisDatosController.cs* y añadir las acciones GET y POST *Create()* correspondientes. Para ello, realizar las siguientes acciones:
  - a. Hacer clic en el botón derecho sobre la carpeta */Controllers*, desplegar la opción **Agregar** y seleccionar la opción **Controlador...**
  - b. En la ventana **Agregar nuevo elemento con scaffolding**, desplegar las opciones **Instalado**, **Común** y **MVC** en el cuadro de la izquierda y seleccionar la opción **Controlador**. Y, a continuación, elegir la opción **Controlador de MVC: en blanco** en el cuadro central de la ventana y hacer clic en **Agregar**.
  - c. En la ventana **Agregar nuevo elemento**, comprobar que se ha seleccionado la opción **Visual C#** en el cuadro de la izquierda y la opción **Controlador de MVC: en blanco** en el cuadro central. En el cuadro de texto **Nombre** situado en la parte inferior, introducir el nombre *MisDatosController.cs* y hacer clic en **Agregar**.
  - d. En el controlador *MisDatosController.cs*, eliminar el código de la acción *Index()* y, a continuación, modificar y añadir el siguiente código para incluir las acciones GET y POST *Create()* y, así poder crear el empleado correspondiente al usuario actual.

```
namespace MvcSoporte.Controllers
{
    [Authorize(Roles = "Usuario")]
    public class MisDatosController : Controller
    {
        private readonly MvcSoporteContexto _context;

        public MisDatosController(MvcSoporteContexto context)
        {
            _context = context;
        }

        // GET: MisDatos/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: MisDatos/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("Id,Nombre,Email,Telefono,FechaNacimiento")] Empleado empleado)
        {
            // Asignar el Email del usuario actual
            empleado.Email = User.Identity.Name;

            if (ModelState.IsValid)
            {
                _context.Add(empleado);
                await _context.SaveChangesAsync();
                return RedirectToAction("Index", "Home");
            }
            return View(empleado);
        }
    }
}
```

En el código anterior, se utiliza la propiedad *Name* de la clase *User.Identity* para asignar el valor del *Email* del usuario actual a la propiedad *Email* de la entidad *empleado*. Puede apreciarse, el uso de los métodos *Add()* y *SaveChangesAsync()* de *LINQ To Entities* para almacenar los datos del empleado correspondiente al usuario actual. Añadir las cláusulas *using* necesarias.

3. A continuación, se creará la vista correspondiente realizando las siguientes acciones:
  - a. Sobre el nombre de la acción *Create()* del controlador *MisDatosController.cs*, hacer clic en el botón derecho y, a continuación, seleccionar la opción **Agregar vista...**
  - b. En la ventana **Agregar nuevo elemento con scaffolding**, seleccionar la opción **Vista de Razor** y hacer clic en **Agregar**.
  - c. En la ventana **Agregar Vista de Razor**, dejar como **Nombre de vista** el valor propuesto *Create*, seleccionar la Plantilla **Empty (sin modelo)**, comprobar que está marcada la opción **Usar página de diseño** y se selecciona *\_Layout.cshtml*. Hacer clic en **Agregar**.
  - d. Una vez que se haya creado la vista correspondiente, comprobar que se habrá creado el archivo de vista *Create.cshtml* en la zona o subcarpeta de vistas */Views/MisDatos*.
  - e. En el archivo de vista *Create.cshtml* de la subcarpeta de vistas */Views/MisDatos*, incluir el siguiente código.

```

@model MvcSoporte.Models.Empleado

@{
    ViewData["Title"] = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Mis datos</h1>

<h4>Empleado</h4>
<p>Es necesario completar sus datos antes de poder empezar a utilizar la aplicación web</p>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Nombre" class="control-label"></label>
                <input asp-for="Nombre" class="form-control" />
                <span asp-validation-for="Nombre" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Telefono" class="control-label"></label>
                <input asp-for="Telefono" class="form-control" />
                <span asp-validation-for="Telefono" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="FechaNacimiento" class="control-label"></label>
                <input asp-for="FechaNacimiento" class="form-control" />
                <span asp-validation-for="FechaNacimiento" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
    
```

Como puede observarse en el código anterior, la clase de datos *Empleado* se ha definido como modelo que se pasa a la vista o **modelo de vista** (*ViewModel*), porque se van a introducir los valores de sus propiedades en esta vista. Además, puede apreciarse que se han incluido varios asistentes de etiquetas (*Tag Helpers*) en el formulario para que puedan introducirse los datos del empleado correspondiente al usuario actual. Los datos del formulario se enviarán a la acción POST *Create()*. Como puede comprobarse, no se ha incluido ningún campo en el formulario para introducir el valor del *Email*, porque su valor se asignará directamente en la acción POST *Create()*, al tratarse del nombre del usuario.

4. Iniciar la depuración y realizar las pruebas para comprobar que al registrar un nuevo usuario, la ejecución se redirige a la acción *Create()* del controlador *MisDatos()*, de modo que se pueden introducir y almacenar sus datos como empleado. Comprobar que, si inicia la sesión un usuario existente que no tiene datos almacenados como empleado, entonces también se redirige la ejecución para introducir y almacenar los datos del empleado correspondiente.

### Procesamiento de modificación de los datos del empleado correspondientes al usuario actual

En este apartado, se incorpora a la aplicación Web *MvcSoporte* el procesamiento que permitirá a un usuario autenticado poder modificar sus propios datos. La información que se podrá modificar está almacenada en la tabla *Empleado* y se accederá a ella a través del campo *Email* del usuario actual. En las aplicaciones Web, este procesamiento suele corresponder con la opción de menú “Mis datos” o similar y solo estará disponible para los usuarios pertenecientes al rol de Usuario. Para ello, hacer:

1. Abrir el archivo *MisDatosController.cs*, puesto que se va a utilizar este mismo controlador para añadir este nuevo procesamiento.
2. Añadir los métodos de acción GET y POST *Edit()* introduciendo el siguiente código.

```
// GET: MisDatos/Edit
public async Task<IActionResult> Edit()
{
    // Se seleccionan los datos del empleado correspondiente al usuario actual
    string? emailUsuario = User.Identity.Name;
    Empleado? empleado = await _context.Empleados
        .Where(e => e.Email == emailUsuario)
        .FirstOrDefaultAsync();
    if (empleado == null)
    {
        return NotFound();
    }
    return View(empleado);
}

// POST: MisDatos/Edit
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
    [Bind("Id,Nombre,Email,Telefono,FechaNacimiento")] Empleado empleado)
{
    if (id != empleado.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(empleado);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!EmpleadoExists(empleado.Id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
    }
}
```

```
        return RedirectToAction("Index", "Home");
    }
    return View(empleado);
}

private bool EmpleadoExists(int id)
{
    return (_context.Empleados?.Any(e => e.Id == id)).GetValueOrDefault();
}
```

En el código anterior, puede observarse el uso que se hace de la propiedad *Name* del objeto *User.Identity*, así como de los métodos *Where()* y *FirstOrDefaultAsync()* de *LINQ To Entities* para seleccionar los datos del empleado correspondiente al usuario actual. Es muy importante observar y analizar el código que va utilizándose para poder comprender su utilidad y alcance.

3. A continuación, crear la vista correspondiente realizando las siguientes acciones:
  - a. En el archivo *MisDatosController.cs* situar el puntero del ratón sobre el nombre de la acción *Edit()*, hacer clic en el botón derecho y seleccionar la opción **Agregar vista...**
  - b. En la ventana **Agregar nuevo elemento con scaffolding**, seleccionar la opción **Vista de Razor** y hacer clic en **Agregar**.
  - c. En la ventana **Agregar Vista de Razor**, dejar el nombre propuesto *Edit*, elegir la Plantilla **Empty (sin modelo)** y la página de diseño *\_Layout.cshtml*. Hacer clic en **Agregar**.
  - d. En el nuevo archivo de vista, denominado *Edit.cshtml*, incluir el siguiente código:

```
@model MvcSoporte.Models.Empleado

@{
    ViewData["Title"] = "Edit";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Mis datos</h1>

<h4>Empleado</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <div class="form-group">
                <label asp-for="Nombre" class="control-label"></label>
                <input asp-for="Nombre" class="form-control" />
                <span asp-validation-for="Nombre" class="text-danger"></span>
            </div>
            @* Email no debe poder modificarse. Se oculta mediante el atributo hidden *@
            <div class="form-group" hidden>
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Telefono" class="control-label"></label>
                <input asp-for="Telefono" class="form-control" />
                <span asp-validation-for="Telefono" class="text-danger"></span>
            </div>
        </div>
    </div>
</div>
```

```
<label asp-for="FechaNacimiento" class="control-label"></label>
<input asp-for="FechaNacimiento" class="form-control" />
<span asp-validation-for="FechaNacimiento" class="text-danger"></span>
</div>
<div class="form-group">
  <input type="submit" value="Save" class="btn btn-primary" />
</div>
</form>
</div>
</div>
</div>
<div>
  <a asp-action="Index" asp-controller="Home">Volver</a>
</div>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

En el código anterior, puede apreciarse que se oculta la edición de la propiedad *Email* del empleado, porque su valor debe corresponder, en todo momento, con el nombre del usuario actual para mantener la vinculación existente entre el usuario y el empleado correspondiente.

4. Abrir y modificar el código del archivo *\_Layout.cshtml* para incluir la opción de menú “Mis Datos”, mediante la solicitud de la acción *Edit* del controlador *MisDatos*. Esta opción de menú solo podrá ser accesible para los usuarios que pertenezcan al rol de Usuario.
5. Modificar el código del archivo *\_Layout.cshtml* para hacer que la opción de menú “Privacidad” solo esté disponible desde la parte pública.
6. Abrir el archivo *HomeController.cs* y eliminar o comentar el filtro *Autorize* aplicado a la acción *Privacy()*, de manera que los usuarios no autenticados puedan ejecutar esta acción.
7. Iniciar la depuración y comprobar el funcionamiento de este procesamiento.

MvcSoporte Mis datos Hello afenoll@empresa.com! Logout

Mis datos

Empleado

Nombre

Alberto Fenoll Llópis

Teléfono

646112277

Fecha de nacimiento

02/11/1984

Save

Volver

## Procesamiento para la gestión de los avisos correspondientes al usuario actual

A continuación, se desarrolla el procesamiento que permitirá a los usuarios gestionar sus propios avisos. Evidentemente, este procesamiento solo estará disponible para los usuarios pertenecientes al rol de Usuario. En las aplicaciones Web esta funcionalidad suele corresponder con la opción de menú denominada “Mis Avisos” o similar. Para ello, realizar las siguientes acciones:

1. Crear un controlador denominado *MisAvisosController.cs* y las vistas asociadas que forman un procesamiento de tipo CRUD sobre la entidad de datos *Aviso*, utilizando para ello el asistente de *Scaffold* y la opción **Controlador de MVC con vistas que usan Entity Framework** para crear el procesamiento. En los siguientes apartados, se modificará el código de este controlador y de sus vistas asociadas, para implementar la gestión de los avisos del usuario actual.
2. En el archivo *MisAvisosController.cs*, añadir el siguiente filtro *Autorize* antes de la declaración de la clase *MisAvisosController*. De este modo, se evitarán los accesos indebidos que puedan producirse a las acciones del controlador *MisAvisos* por parte de usuarios que no pertenezcan al rol de Usuario. Añadir también la cláusula *using* correspondiente.

```
[Authorize(Roles = "Usuario")]
```

3. Modificar el código de la acción *Index()* del controlador *MisAvisosController.cs* para incluir y adecuar el código que aparece resaltado a continuación.

```
// GET: MisAvisos
public async Task<IActionResult> Index()
{
    // Se selecciona el empleado correspondiente al usuario actual
    var emailUsuario = User.Identity.Name;
    var empleado = await _context.Empleados.Where(e => e.Email == emailUsuario)
        .FirstOrDefaultAsync();
    if (empleado == null)
    {
        return RedirectToAction("Index", "Home");
    }

    // Se seleccionan los avisos del Empleado correspondiente al usuario actual
    var misAvisos = _context.Avisos
        .Where(a => a.EmpleadoId == empleado.Id)
        .OrderByDescending(a => a.FechaAviso)
        .Include(a => a.Empleado).Include(a => a.Equipo).Include(a => a.TipoAveria);

    return View(await misAvisos.ToListAsync());

    // var mvcSoporteContexto = _context.Avisos.Include(a => a.Empleado)
    //     .Include(a => a.Equipo).Include(a => a.TipoAveria);
    // return View(await mvcSoporteContexto.ToListAsync());
}
```

Como puede apreciarse en el código anterior, en primer lugar, se seleccionan los datos del empleado que corresponde con el usuario actual. Y, posteriormente, se seleccionan los avisos correspondientes a este empleado y se ordenan por fecha de aviso. Como puede apreciarse, para realizar estas operaciones se emplean diversos métodos de *LINQ To Entities*.



4. A continuación, modificar el código de las acción GET y POST *Create()* del controlador *MisAvisos* para incluir y adecuar el código resaltado a continuación.

```
// GET: MisAvisos/Create
public IActionResult Create()
{
    // ViewData["EmpleadoId"] = new SelectList(_context.Empleados, "Id", "Nombre");
    ViewData["EquipoId"] = new SelectList(_context.Equipos, "Id", "CodigoEquipo");
    ViewData["TipoAveriaId"] = new SelectList(_context.TipoAverias, "Id", "Descripcion");

    return View();
}

// POST: MisAvisos/Create
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
Create([Bind("Id,Descripcion,FechaAviso,FechaCierre,Observaciones,EmpleadoId,TipoAveriaId,E
quipoId")] Aviso aviso)
{
    // Se asigna al aviso el Id del empleado correspondiente al usuario actual
    var emailUsuario = User.Identity.Name;
    var empleado = await _context.Empleados
        .Where(e => e.Email == emailUsuario)
        .FirstOrDefaultAsync();
    if (empleado != null)
    {
        aviso.EmpleadoId = empleado.Id;
    }

    if (ModelState.IsValid)
    {
        _context.Add(aviso);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    // ViewData["EmpleadoId"] = new SelectList(_context.Empleados, "Id", "Nombre",
    //     aviso.EmpleadoId);
    ViewData["EquipoId"] = new SelectList(_context.Equipos, "Id", "CodigoEquipo",
        aviso.EquipoId);
    ViewData["TipoAveriaId"] = new SelectList(_context.TipoAverias, "Id", "Descripcion",
        aviso.TipoAveriaId);

    return View(aviso);
}
```

Como puede apreciarse en el código anterior, en la acción POST se asigna directamente el *Id* de empleado correspondiente al usuario actual a la propiedad *EmpleadoId* de la entidad *aviso*. También puede apreciarse que se comenta la obtención de la lista de empleados para pasar a la vista mediante *ViewData*, porque el empleado es el que corresponde con el usuario actual.

5. Abrir el archivo de vista *Create.cshtml* de la zona de vistas */View/MisAvisos* para comentar, mediante los caracteres *@\* \*@*, el código correspondiente al asistente de etiqueta o *Tag Helper* que facilita la introducción del empleado que realiza el aviso. Este código ya no es necesario, porque el empleado que crea el aviso deberá ser el empleado correspondiente al usuario actual. La porción de código a comentar será similar al que se resalta a continuación.



```
...
<div class="form-group">
  <label asp-for="Observaciones" class="control-label"></label>
  <input asp-for="Observaciones" class="form-control" />
  <span asp-validation-for="Observaciones" class="text-danger"></span>
</div>
@*
<div class="form-group">
  <label asp-for="EmpleadoId" class="control-label"></label>
  <select asp-for="EmpleadoId" class="form-control"
    asp-items="ViewBag.EmpleadoId"></select>
</div>
*@
<div class="form-group">
  <label asp-for="TipoAveriaId" class="control-label"></label>
  <select asp-for="TipoAveriaId" class="form-control"
    asp-items="ViewBag.TipoAveriaId"></select>
</div>
...
```

6. Abrir el archivo de vista *Edit.cshtml* de la zona de vistas */View/MisAvisos* y, a continuación, modificar la línea de código que aparece resaltada a continuación para añadir el atributo *hidden*. Este atributo oculta la división que permite seleccionar el empleado que realiza el aviso. Este código ya no es necesario, porque el empleado que realiza el aviso debe ser el correspondiente al usuario actual y, evidentemente, no debe posible que se pueda modificar.

```
...
<div class="form-group">
  <label asp-for="Observaciones" class="control-label"></label>
  <input asp-for="Observaciones" class="form-control" />
  <span asp-validation-for="Observaciones" class="text-danger"></span>
</div>
<div class="form-group" hidden>
  <label asp-for="EmpleadoId" class="control-label"></label>
  <select asp-for="EmpleadoId" class="form-control"
    asp-items="ViewBag.EmpleadoId"></select>
  <span asp-validation-for="EmpleadoId" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="TipoAveriaId" class="control-label"></label>
  <select asp-for="TipoAveriaId" class="form-control"
    asp-items="ViewBag.TipoAveriaId"></select>
  <span asp-validation-for="TipoAveriaId" class="text-danger"></span>
</div>
...
```

7. Abrir el archivo *\_Layout.cshtml* y, a continuación, modificar el código para incluir la opción de menú "Mis Avisos". Esta opción de menú solicitará la acción *Index()* del controlador *MisAvisos* y solo deberá ser accesible para los usuarios autenticados en la aplicación Web que pertenezcan al rol de Usuario.
8. Iniciar la depuración y comprobar el resultado obtenido. Introducir varios avisos realizados por uno o varios usuarios para poder realizar pruebas de funcionamiento. Es importante observar y analizar con detenimiento el código para ir comprendiendo su funcionalidad y utilidad.

MvcSoporte Mis Datos Mis Avisos Hello afenoll@empresa.com! Logout

## Mis Avisos

[Create New](#)

Descripción del problema	Fecha de aviso	Fecha de cierre	Observaciones	Empleado	TipoAveria	Equipo
No se pone en marcha	02/06/2022			Alberto Fenoll Llópis	No arranca el sistema	PC20167 <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Aparece error grave del sistema	12/04/2022	14/04/2022	Reinstalación de Windows	Alberto Fenoll Llópis	Problema indeterminado	PC12391 <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

© 2022 - MvcSoporte - IES Mare Nostrum. DAW-DWS - [Política de privacidad](#)

9. A continuación, es necesario añadir cierto código en el controlador *MisAvisos* para evitar que el usuario actual pueda editar avisos creados por otros empleados. Este problema de seguridad se produce porque es posible modificar la solicitud URL correspondiente en la barra de direcciones del navegador. Así, se puede comprobar este problema de seguridad si se realizan las siguientes acciones:
- Iniciar la depuración. Y, a continuación, acceder a la aplicación a través de un usuario que pertenece al rol "Usuario" y que haya realizado varios avisos.
  - Seleccionar la opción "Mis Avisos" en el menú de la aplicación Web.
  - Hacer clic en la opción correspondiente para editar uno de los avisos del usuario, de manera que se accede al procesamiento de edición del aviso seleccionado.
  - Podrá comprobarse, entonces, que en la barra de direcciones del navegador aparecerá la dirección de la solicitud URL correspondiente. Esta dirección puede tener, por ejemplo, un valor similar a <https://localhost:44384/MisAvisos/Edit/8>, lo que significa que se va a editar el aviso cuyo valor en la propiedad *Id* es 8.
  - Si ahora, se modifica el valor 8 en la barra de direcciones por otro valor, por ejemplo, el 3 y se pulsa la tecla <Return>, podrá comprobarse que se accederá al procesamiento de edición del aviso cuyo valor *Id* es 3. Debe tenerse en cuenta que, además, el aviso cuyo valor de *Id* es 3 puede pertenecer a otro empleado distinto. Así que, se hace necesario resolver este problema de seguridad en la autorización.

10. Para asegurar que solo se puedan editar los avisos que han sido creados por el empleado que corresponde con el usuario actual, abrir el controlador *MisAvisos* y modificar el código de la acción GET *Edit()* para incluir el código que aparece resaltado a continuación.

```
// GET: MisAvisos/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null || _context.Avisos == null)
    {
        return NotFound();
    }

    var aviso = await _context.Avisos.FindAsync(id);
    if (aviso == null)
    {
        return NotFound();
    }

    // Para evitar el acceso a los avisos de otros empleados
    var emailUsuario = User.Identity.Name;
    var empleado = await _context.Empleados
        .Where(e => e.Email == emailUsuario)
        .FirstOrDefaultAsync();
    if (empleado == null)
    {
        return NotFound();
    }
    if (aviso.EmpleadoId != empleado.Id)
    {
        return RedirectToAction(nameof(Index));
    }

    ViewData["EmpleadoId"] = new SelectList(_context.Empleados, "Id", "Nombre",
aviso.EmpleadoId);
    ViewData["EquipoId"] = new SelectList(_context.Equipos, "Id", "CodigoEquipo",
aviso.EquipoId);
    ViewData["TipoAveriaId"] = new SelectList(_context.TipoAverias, "Id", "Descripcion",
aviso.TipoAveriaId);

    return View(aviso);
}
```

En el código anterior, puede apreciarse que, en primer lugar, se realiza una selección para encontrar el empleado correspondiente al usuario actual. Y, a continuación, se comparan el valor del *Id* del empleado con el valor del *Id* del empleado que está almacenado en la entidad *aviso* que se desea modificar. Si no coinciden estos valores, se redirecciona a la lista de avisos del usuario y se evita el acceso indebido a los avisos realizados por otros empleados.

11. Iniciar la depuración y realizar las pruebas necesarias para comprobar que solo es posible editar los avisos correspondientes al usuario actual.
12. Añadir a un código similar en las acciones GET *Delete()* y GET *Details()* del controlador *MisAvisos*, de manera que se pueda evitar, de un modo similar, los posibles accesos indebidos a la información de los avisos correspondientes a otros empleados.
13. Iniciar la depuración y realizar las pruebas necesarias para comprobar que solo es posible editar, ver los detalles y eliminar los avisos correspondientes al usuario actual.

### Obtener la lista de los usuarios de la aplicación Web

A continuación, se va a realizar el procesamiento para obtener y presentar la información de los usuarios de la aplicación Web y el rol al que pertenecen. Se creará un nuevo controlador denominado *Usuarios*, así como las vistas correspondientes. Las acciones de este controlador solo las podrán ejecutar los usuarios que pertenezcan al rol de “Administrador”. Para presentar la información de los usuarios de la aplicación Web realizar las siguientes acciones:

1. En primer lugar, se va a crear una clase de datos en el modelo que se utilizará para pasar datos desde el controlador hacia la vista. Es habitual crear una clase de datos específica en el modelo para pasar unos datos concretos a la vista, en lugar de utilizar las clases de datos del modelo que representan tablas. Estas clases específicas suelen denominarse **clases de vista** y, por convención, sus nombres comienzan con el prefijo *View*. Crear la clase *ViewUsuarioConRol.cs* en la carpeta */Models* y, a continuación, incluir el siguiente código en la clase.

```
public class ViewUsuarioConRol
{
    public string? Email { get; set; }
    public string? NombreUsuario { get; set; }
    public string? RolDeUsuario { get; set; }
}
```

2. Una vez preparada la clase de vista anterior, crear un controlador en blanco denominado *UsuariosController.cs*, tal como se ha realizado en un ejercicio anterior. Y, posteriormente, modificar y añadir el código que aparece resaltado para obtener el siguiente resultado.

```
namespace MvcSoporte.Controllers
{
    [Authorize(Roles = "Administrador")]
    public class UsuariosController : Controller
    {
        private readonly ApplicationDbContext _context;

        public UsuariosController(ApplicationDbContext context)
        {
            _context = context;
        }

        public IActionResult Index()
        {
            var usuarios = from user in _context.Users
                           join userRoles in _context.UserRoles on user.Id equals userRoles.UserId
                           join role in _context.Roles on userRoles.RoleId equals role.Id
                           select new ViewUsuarioConRol
                           {
                               Email = user.Email,
                               NombreUsuario = user.UserName,
                               RolDeUsuario = role.Name
                           };

            return View(usuarios.ToList());
        }
    }
}
```

Como puede apreciarse en el código, se define una expresión de consulta de *LinQ To Entities* para obtener los datos y asignarlos a una colección de entidades de tipo *ViewUsuarioConRol*. También puede apreciarse que en el controlador *UsuariosController.cs* se utiliza el contexto de usuarios y roles, denominado *ApplicationDbContext*, tal como se especifica en la inyección de dependencia al inicio del código de la clase. Añadir las cláusulas *using* que sean necesarias.

3. Crear la vista *Index.cshtml* correspondiente utilizando la plantilla **Empty (sin modelo)**, tal como se hizo en un ejercicio anterior y, a continuación, añadir el siguiente código.

```
@model IEnumerable<MvcSoporte.Models.ViewUsuarioConRol>

@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Usuarios</h1>

<br />

<table class="table">
    <tr>
        <th>
            Usuario
        </th>
        <th>
            Nombre de Usuario
        </th>
        <th>
            Rol
        </th>
    </tr>

    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.NombreUsuario)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.RolDeUsuario)
            </td>
        </tr>
    }
</table>
```

En el código anterior puede observarse que en la primera línea del código y empleando la directiva *@model*, se especifica el tipo de datos del modelo que se pasa a la vista y que, en esta ocasión, se trata de una colección de entidades de tipo *ViewUsuarioConRol*.

4. Abrir el archivo *\_Layout.cshtml* y modificar el código para incluir la opción de menú “Usuarios”, mediante la solicitud de la acción *Index* del controlador *Usuarios*. Esta opción de menú solo deberá ser accesible para los usuarios que pertenezcan al rol de “Administrador”.
5. Iniciar la depuración para comprobar los resultados obtenidos.

MvcSoporte Avisos Equipos Localizaciones Averias Empleados Usuarios Hello admin@empresa.com! Logout		
Usuarios		
Usuario	Nombre de Usuario	Rol
admin@empresa.com	admin@empresa.com	Administrador
ejuarez@empresa.com	ejuarez@empresa.com	Usuario
usuario1@empresa.com	usuario1@empresa.com	Usuario
usuario2@empresa.com	usuario2@empresa.com	Usuario
afenoll@empresa.com	afenoll@empresa.com	Usuario
prueba1@empresa.com	prueba1@empresa.com	Usuario

### Crear usuarios pertenecientes al rol de Administrador

A continuación, se desarrolla el procesamiento que permitirá crear nuevos usuarios que vayan a pertenecer al rol de “Administrador”. Los métodos de acción necesarios para llevar a cabo este procesamiento se agregarán al controlador *Usuarios*, por lo que solo podrá ser ejecutado por los usuarios que pertenezcan al rol de “Administrador”. Además, se creará la vista correspondiente. Para ello, realizar las siguientes acciones:

1. Abrir el archivo *UsuariosController.cs*, puesto que se va a utilizar este mismo controlador para incluir el procesamiento a desarrollar en este apartado.
2. Añadir y modificar el código que aparece resaltado a continuación.

```
namespace MvcSoporte.Controllers
{
    [Authorize(Roles = "Administrador")]
    public class UsuariosController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly UserManager<IdentityUser> _userManager;

        public UsuariosController(ApplicationDbContext context,
            UserManager<IdentityUser> userManager)
        {
            _context = context;
            _userManager = userManager;
        }
    }
}
```

```
public IActionResult Index()
{
    var usuarios = from user in _context.Users
        join userRoles in _context.UserRoles on user.Id equals userRoles.UserId
        join role in _context.Roles on userRoles.RoleId equals role.Id
        select new ViewUsuarioConRol
        {
            Email = user.Email,
            NombreUsuario = user.UserName,
            RolDeUsuario = role.Name
        };

    return View(usuarios.ToList());
}

// GET: Usuarios/Create
public IActionResult Create()
{
    return View();
}

// POST: Usuarios/Create
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Email,Password")]
    RegisterModel.InputModel model)
{
    // Se crea el nuevo usuario
    var user = new IdentityUser();
    user.UserName = model.Email;
    user.Email = model.Email;
    string usuarioPWD = model.Password;
    var result = await _userManager.CreateAsync(user, usuarioPWD);

    // Se asigna el rol de "Administrador" al usuario
    if (result.Succeeded)
    {
        var result1 = await _userManager.AddToRoleAsync(user, "Administrador");
        return RedirectToAction(nameof(Index));
    }

    return View(model);
}
}
```

En el código anterior pueden apreciarse las acciones GET y POST *Create()* que expresan el procesamiento para la creación de un nuevo usuario que pertenezca al rol “Administrador”. También se aprecia que se ha creado una instancia de la clase *userManager* mediante una inyección de dependencia. Esta clase expone las funcionalidades de la API de *ASP.NET Core Identity* para realizar las operaciones sobre el contexto de usuarios y roles. La instancia *\_userManager* se usa para crear el usuario y asignarle el rol “Administrador”. En el código también se aprecia el uso de la clase *RegisterModel.InputModel* que pertenece al espacio de nombres *MvcSoporte.Areas.Identity.Pages.Account* de *ASP.NET Core Identity*. Esta clase define los datos necesarios para el registro de un usuario. Se utiliza el atributo *Bind()* para vincular el modelo de vista con el parámetro *model* de la acción. Añadir las cláusulas *using* necesarias.

3. Crear la vista *Create.cshtml* correspondiente, utilizando para ello la plantilla de vistas **Empty (sin modelo)**, tal como se hizo en un ejercicio anterior. A continuación, añadir el siguiente código a la vista que se habrá creado en la subcarpeta de vistas */Views/Usuarios*.

```
@model MvcSoporte.Areas.Identity.Pages.Account.RegisterModel.InputModel

@{
    ViewData["Title"] = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Crear Administrador</h1>

<div class="row">
    <div class="col-md-4">
        <form asp-action="Create" method="post">
            <h5>Crea una nueva cuenta de Administrador</h5>
            <hr />
            <div asp-validation-summary="All" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Email"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Password"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>
            <div class="form-group">
                @*<label asp-for="ConfirmPassword"></label> *@
                <label>Confirma Password</label>
                <input asp-for="ConfirmPassword" class="form-control" />
                <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
            </div>
            <br />
            <button type="submit" class="btn btn-primary">Registrar Administrador</button>
        </form>
    </div>
</div>

<br />

<div>
    <a asp-action="Index">Volver</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

En el Código anterior puede apreciarse que mediante la directiva `@model` se especifica como modelo de vista una entidad de tipo *RegisterModel.InputModel*.

4. Finalmente, abrir el archivo de vista *Index.cshtml* de la subcarpeta de vistas */Views/Usuarios* y añadir el código que aparece resaltado a continuación, para incluir el enlace de solicitud hacia la acción que procesa la creación de un usuario perteneciente al rol de Administrador.



```
@model IEnumerable<MvcSoporte.Models.ViewUsuarioConRol>

@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h1>Usuarios</h1>

<br />

<p>
    <a asp-action="Create">Nuevo Administrador</a>
</p>

<table class="table">
    <tr>
        <th>
            Usuario
        </th>
        . . .
```

5. Iniciar la depuración y realizar las pruebas necesarias para comprobar que el funcionamiento de este procesamiento sea correcto.

[MvcSoporte](#) [Avisos](#) [Equipos](#) [Localizaciones](#) [Averias](#) [Empleados](#) [Usuarios](#) [Hello admin@empresa.com!](#) [Logout](#)

## Crear Administrador

Crea una nueva cuenta de Administrador

Email

Password

Confirma Password

[Volver](#)