

Prácticas

Tema 6 – Desarrollo de aplicaciones Web MVC (I)

Objetivos	<p>En esta práctica, se aborda la realización de tareas relacionadas con:</p> <ul style="list-style-type: none">• Crear aplicaciones Web basadas en ASP.NET Core MVC empleando el enfoque <i>Code First</i> para el enlace a datos.• Crear controladores y vistas que implementen procesamientos básicos de tipo CRUD (<i>Create, Read, Update and Delete</i>).• Agregar una gestión de los errores en tiempo de ejecución a nivel de aplicación Web.• Realizar migraciones de datos para mantener la coherencia entre el modelo de la aplicación Web y el esquema de la base de datos.
Conocimientos previos	<p>Para realizar esta práctica, es necesario tener conocimientos sobre:</p> <ul style="list-style-type: none">• Características de las aplicaciones Web Interactivas con procesamiento en el servidor Web.• Modelos de arquitectura de software. Características del modelo de arquitectura de software Modelo-Vista-Controlador (MVC).• Lenguaje de programación Microsoft Visual C#.• Programación orientada a objetos y uso de clases.• Bases de datos relacionales.
Escenario	<p>En las prácticas anteriores se ha utilizado la tecnología ASP.NET para desarrollar aplicaciones Web basadas en ASP.NET Web Forms. Esta práctica se centra en el uso del modelo de arquitectura del software Modelo-Vista-Controlador (MVC) para desarrollar aplicaciones Web basadas en la tecnología ASP.NET Core MVC. Aunque ambos modelos de desarrollo son muy diferentes en cuanto a la concepción de la arquitectura del software, tienen en común que permiten desarrollar de aplicaciones Web interactivas con procesamiento en el servidor. La existencia de diversos modelos de desarrollo de aplicaciones no significa substitución entre ellos. Al contrario, el hecho de poder disponer de diversos enfoques de implementación de aplicaciones Web interactivas, amplía las posibilidades en cuanto al desarrollo de aplicaciones Web interactivas.</p> <p>Se presentan diversos ejercicios que van dirigidos a comprender, experimentar e indagar algunos de los aspectos prácticos más básicos del desarrollo de aplicaciones Web basadas en ASP.NET Core MVC. El objetivo de este primer tema dedicado al estudio de las aplicaciones Web basadas en ASP.NET Core MVC es comprender la concepción y el funcionamiento de la arquitectura del software Modelo-Vista-Controlador (MVC).</p>

Ejercicio 1

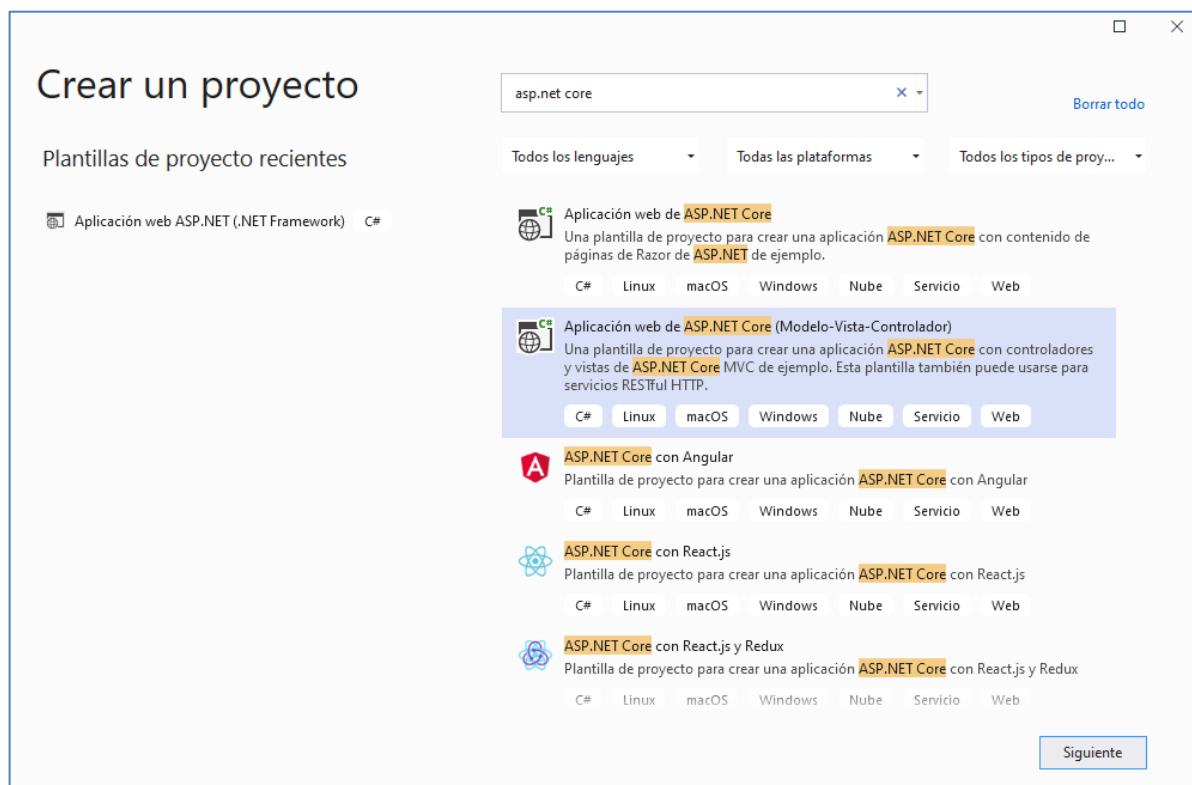
Desarrollo de una aplicación Web basada en ASP.NET Core MVC empleando el enfoque *Code First* para el enlace a datos

En este ejercicio se creará una nueva aplicación Web basada en ASP.NET Core MVC, denominada *MvcAgenda*. Esta aplicación Web facilitará la gestión de la información relativa a la planificación de las tareas que realizan los empleados de una empresa u organización. La nueva aplicación Web utilizará el patrón arquitectónico Modelo-Vista-Controlador (MVC), por lo que su estructura y funcionamiento será completamente diferente al de las aplicaciones Web basadas en ASP.NET Web Forms que se han desarrollado en prácticas anteriores. El desarrollo la aplicación Web incluirá la creación de una base de datos para almacenar los datos que maneja la aplicación Web. Se emplea un enfoque *Code First* para el enlace a datos, de modo que la creación de la base de datos se realizará a partir del código.

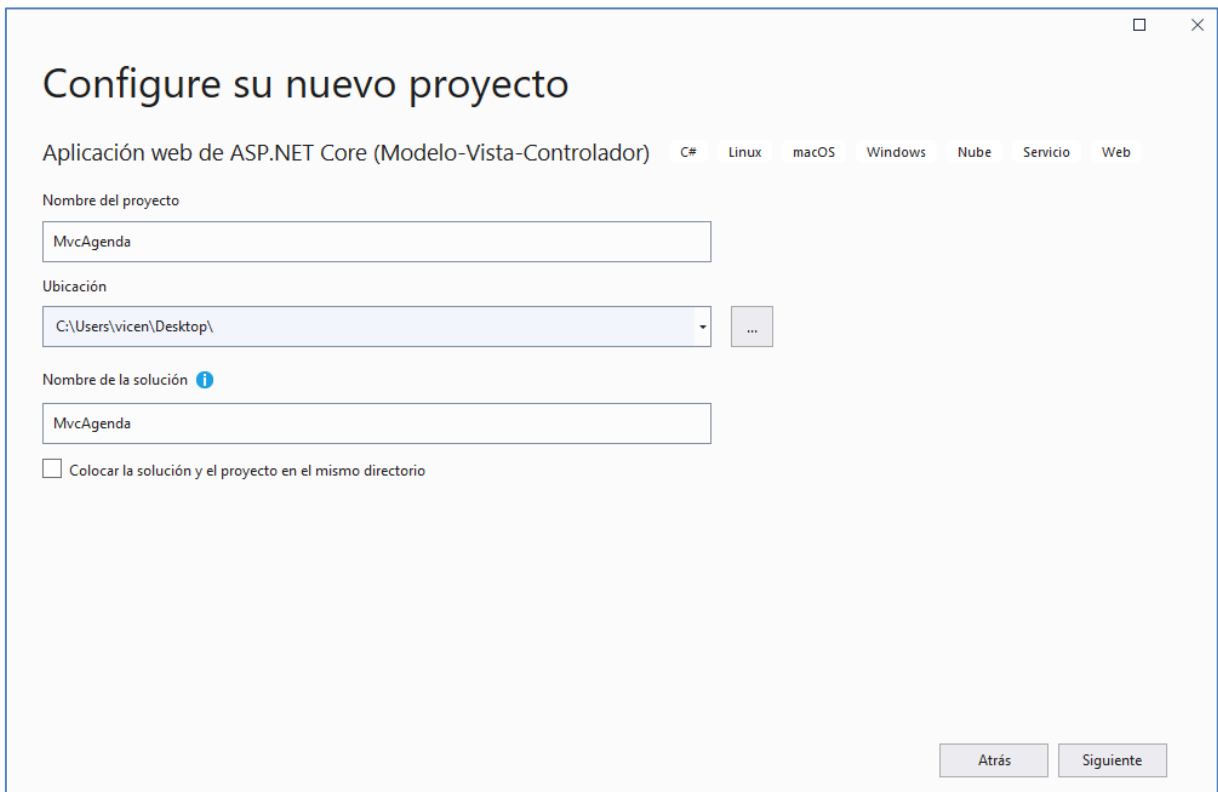
Crear una nueva Aplicación Web basada en ASP.NET Core MVC

En primer lugar, se creará una nueva Aplicación Web basada en ASP.NET Core MVC utilizando la plantilla de proyecto de Visual Studio correspondiente. Para ello, realizar las siguientes acciones:

1. Acceder al menú **Archivo** de Visual Studio, expandir la opción **Nuevo** y seleccionar **Proyecto...**
2. Seleccionar la plantilla **Aplicación web de ASP.NET Core (Modelo-Vista-Controlador)** que utiliza el **lenguaje de programación C#** en la ventana **Crear un Proyecto**, y hacer clic en **Siguiente**. Se facilita la búsqueda si se introduce "asp.net core" en el cuadro de búsqueda.



3. A continuación, en la ventana **Configure su nuevo proyecto** realizar las siguientes acciones:
 - a. Introducir el **Nombre del proyecto** de Aplicación Web que será: *MvcAgenda*.
 - b. Seleccionar una **Ubicación** para la Solución de Visual Studio mediante el botón ... situado a la derecha. Por ejemplo, puede utilizarse el *Escritorio* como ubicación.
 - c. En el cuadro de texto **Nombre de la solución** aparece, de manera predeterminada, el mismo nombre que se le ha dado al proyecto. En este caso, la Solución de Visual Studio y el Proyecto de Aplicación Web de ASP.NET Core MVC tendrán el mismo nombre.
 - d. Comprobar que está desactivada la opción **Colocar la solución y el proyecto en el mismo directorio**.
 - e. Hacer clic en el botón **Siguiente**.



4. En la ventana **Información adicional**, realizar las siguientes acciones:
 - a. En el cuadro combinado **Plataforma de destino**, elegir la opción **.NET 5.0 (Actual)**. Esta opción especifica que se utilizará la versión .NET 5.0 para abordar el desarrollo de la aplicación Web *MvcAgenda*.
 - b. En el cuadro combinado **Tipo de autenticación**, elegir la opción **Cuentas individuales**.
 - c. Comprobar que está activada la casilla de verificación **Configurar para HTTPS**.
 - d. Comprobar que está desactivada la opción **Habilitar Docker**, y que está deshabilitado el cuadro de diálogo **Sistema operativo de Docker**.
 - e. Comprobar que está desactivada la opción **Habilitar compilación en tiempo de ejecución de Razor**.
 - f. Finalmente, hacer clic en **Crear**.

Información adicional

Aplicación web de ASP.NET Core (Modelo-Vista-Controlador) C# Linux macOS Windows Nube Servicio Web

Plataforma de destino

.NET 5.0 (Actual)

Tipo de autenticación

Cuentas individuales

☒ Configurar para HTTPS

☐ Habilitar Docker

Sistema operativo de Docker

Linux

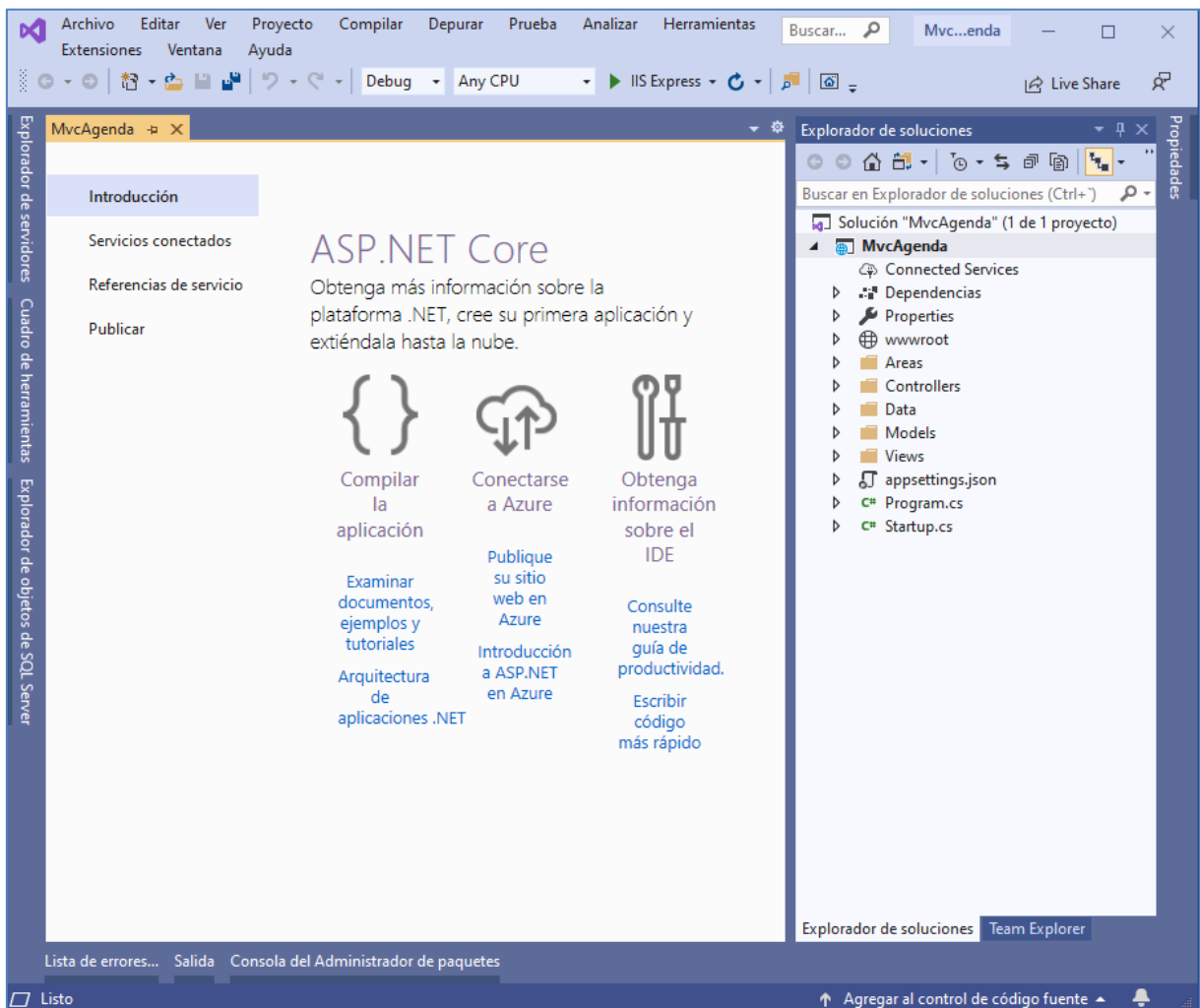
☐ Habilitar compilación en tiempo de ejecución de Razor

Atrás Crear

Una vez finalizado el proceso de creación de la Aplicación Web de ASP.NET Core MVC puede comprobarse, mediante el **Explorador de soluciones** de Visual Studio, que se habrá creado la Solución *MvcAgenda* y, en su interior, el Proyecto de aplicación Web basada en ASP.NET Core MVC denominado también *MvcAgenda*. Además, puede comprobarse que se ha creado una estructura compleja de carpetas y archivos. Esta estructura de componentes o módulos da soporte al *framework* de desarrollo de aplicaciones Web basadas en el modelo de arquitectura de desarrollo del software Modelo-Vista-Controlador, empleando para ello la tecnología de desarrollo ASP.NET Core MVC. A continuación, se repasa brevemente la **estructura de archivos y carpetas** que conforman una aplicación Web basada en ASP.NET Core MVC, tal como se muestra en el **Explorador de soluciones**:

- Las carpetas más importantes y que más se van a utilizar durante el desarrollo de la aplicación Web basada en ASP.NET Core MVC son: **/Controllers**, en la que se crearán los controladores; **/Models**, en la que se creará el modelo; y, **/Views**, en la que se crearán las vistas. De manera predeterminada, un proyecto de aplicación Web basada en ASP.NET Core MVC incluye el controlador *Home*, así como las vistas asociadas en la subcarpeta de vistas */Views/Home*.
- La carpeta **/Data** contiene la definición de las clases del contexto de datos.
- Los módulos **/Properties**, **appsettings.json**, **Program.cs** y **Startup.cs** son carpetas y archivos de configuración de la aplicación Web.
- La carpeta **/wwwroot** contiene contenido estático que usa la aplicación Web como las hojas de estilo externas y los archivos y librerías de Javascript como: jQuery, Modernizr, Bootstrap, etc. que se incorporan de forma predeterminada al proyecto de ASP.NET Core MVC. En esta carpeta también se suelen incluir las imágenes que usa la interfaz y otros archivos estáticos.

- La carpeta **/Dependencias** contiene las referencias a los paquetes y las librerías preinstaladas que son necesarias para la solución.
- La carpeta **/Migrations** contiene los archivos relacionados con la realización de migraciones.
- La carpeta **/Areas** permite definir diferentes áreas en una misma aplicación Web. Cada área es una parte funcional que incorpora de la misma estructura de archivos y carpetas que una aplicación Web basada en ASP.NET Core MVC. Se suele utilizar para estructurar las diferentes funcionalidades de una aplicación Web compleja y, también, para organizar el enrutamiento.



Debe tenerse siempre presente que el desarrollo de aplicaciones Web mediante la utilización de un modelo de arquitectura del software Modelo-Vista-Controlador produce que, tanto la estructura de la aplicación Web, como el funcionamiento de los componentes que la forman, sean específicos para este tipo de arquitectura. A continuación, brevemente, se repasan algunos de los **aspectos y conceptos** más importantes del patrón arquitectónico Modelo-Vista-Controlador (MVC).

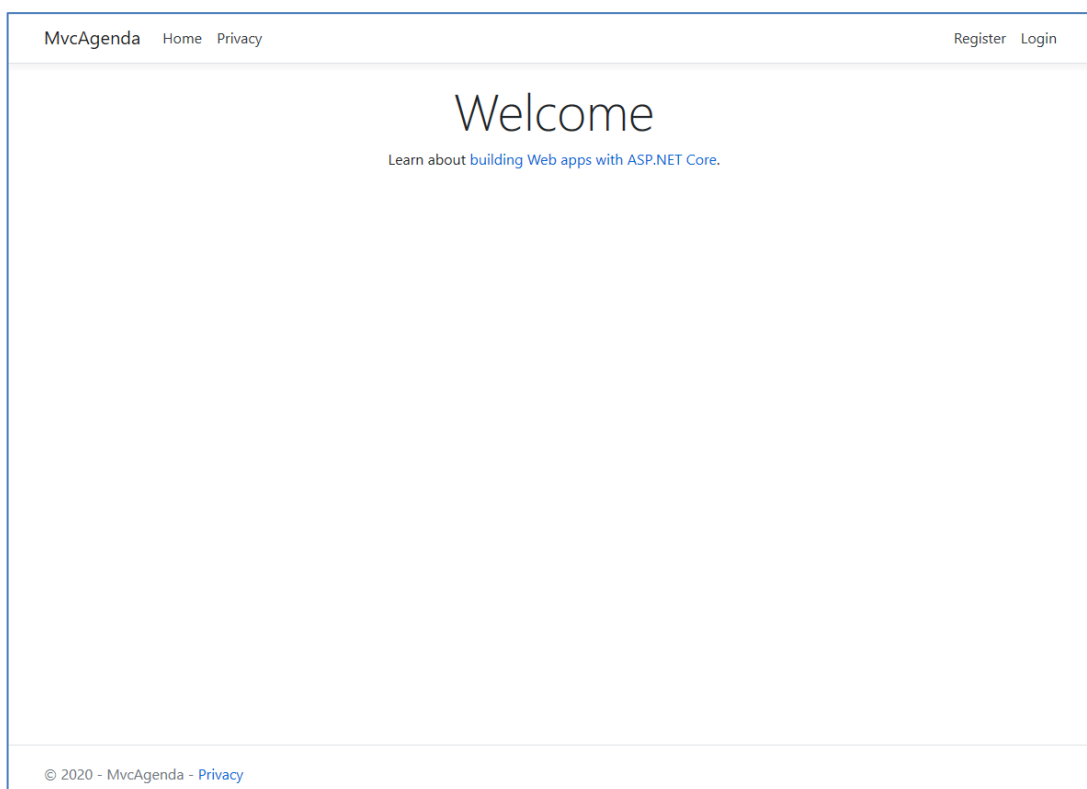
- El **Modelo** es una representación en objetos de la información que maneja la aplicación web en un momento dado y gestiona los accesos a dicha información a través de *Entity Framework*. El Modelo responde a las peticiones de acceso a datos a través de colecciones de objetos.

- El **Controlador** se encarga de procesar las solicitudes del usuario. Una acción de un controlador se inicia mediante una solicitud URL, siguiendo un formato de enrutamiento. Puede actuar sobre el modelo para acceder a los datos que debe manejar al procesar la solicitud resultante de la interacción del usuario. Normalmente, una acción de un controlador dirige el resultado del procesamiento hacia una Vista concreta, considerando una convención de nombres.
- La **Vista** se encarga de generar las páginas HTML de respuesta que constituye la interfaz con la que interactúa el usuario, empleándose para ello un lenguaje o motor de vistas. Las vistas pueden acceder al Modelo para seleccionar y presentar datos auxiliares en la respuesta.

Utilizando el **Explorador de archivos** de Windows se puede comprobar que se habrá creado la carpeta de Solución *MvcAgenda* en la ubicación deseada. Esta carpeta contendrá el archivo de Solución *MvcAgenda.sln*, así como la carpeta del Proyecto *MvcAgenda*, que a su vez contendrá los archivos y carpetas correspondientes al Proyecto de aplicación Web basada en ASP.NET Core MVC.

Probar y adaptar el Proyecto de ASP.NET Core MVC creado

El Proyecto de ASP.NET Core MVC creado anteriormente mediante la plantilla de Visual Studio ya puede ejecutarse, para ello basta con **Iniciar la depuración**.



Los componentes o módulos creados inicialmente en el Proyecto de ASP.NET Core MVC pueden modificarse para poder adaptarlos a las necesidades de implementación específicas en cada caso. A continuación, se van a realizar algunas sencillas adaptaciones específicas en el archivo de plantilla de diseño común, denominada *_Layout.cshtml*. Para ello, realizar las siguientes acciones:

1. Abrir el archivo de plantilla de diseño común `_Layout.cshtml` que se encuentra en la carpeta `/Views/Shared`. Cualquier cambio que se realice en este archivo afectará a la apariencia de todas las vistas, y por tanto a todas las páginas de la aplicación Web. Esto es porque en el archivo `_ViewStart.cshtml` de la carpeta `/Views` se especifica que el archivo de plantilla de diseño común de la aplicación Web es `_Layout.cshtml`, mediante un código similar al siguiente:

```
@{  
    Layout = "_Layout";  
}
```

En el caso de que una vista concreta deba utilizar una plantilla de diseño diferente, puede especificarse el nombre de esa otra plantilla de diseño en el código de la vista concreta.

2. En el archivo `_Layout.cshtml`, realizar los cambios que aparecen resaltados en el código:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="utf-8" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <title>@ViewData["Title"] - MvcAgenda</title>  
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />  
    <link rel="stylesheet" href="~/css/site.css" />  
</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">  
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">MvcAgenda</a>  
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-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 flex-sm-row-reverse">  
                    <partial name="_LoginPartial" />  
                    <ul class="navbar-nav flex-grow-1">  
                        <li class="nav-item">  
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Inicio</a>  
                        </li>  
                        <li class="nav-item">  
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Política de privacidad</a>  
                        </li>  
                    </ul>  
                </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; 2021 - MvcAgenda. DWS/DAW. IES Mare Nostrum -  
            <a asp-area="" asp-controller="Home" asp-action="Privacy">Política de privacidad</a>  
        </div>  
    </footer>  
</body>  
</html>
```



```
</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>
@RenderSection("Scripts", required: false)
</body>
</html>
```

3. Iniciar la depuración para comprobar los cambios realizados. Puede comprobarse que todas las páginas de la aplicación Web, que corresponden con las opciones de menú, se ven afectadas por los cambios realizados en el código del archivo `_Layout.cshtml`. El contenido específico de cada una de las vistas que utilicen la plantilla de diseño `_Layout.cshtml`, se presentará visualmente en la posición marcada por la instrucción de Razor: `@RenderBody()`.

Agregar el modelo a la aplicación Web

A continuación, se agregarán a la aplicación Web, las clases del modelo necesarias para gestionar la planificación de las tareas que realizan los empleados que pertenecen a los diferentes departamentos de una empresa u organización. Las clases de datos y la clase del contexto de datos, que constituyen el Modelo de la aplicación Web, se utilizan junto con *Entity Framework Core (EF Core)* para poder trabajar con una base de datos desde el modelo. *EF Core* es un *framework* ORM (*Object-Relational Mapping*) o, en castellano, un *framework* de Asignación Relacional de Objetos (ARO) que se utiliza para administrar el acceso a los datos almacenados y simplificar el código de acceso a datos.

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



Las clases de datos del modelo se conocen como clases POCO (*Plain Old CLR Objects*), o en castellano Objetos antiguos sin formato, porque no tienen ninguna dependencia de *EF Core*. Simplemente definen las propiedades de los datos que se almacenarán en la base de datos. Se emplea un enfoque *Code First* para establecer el enlace a datos, porque se escribe primero el código de las clases de las entidades de datos y del contexto de datos del modelo y, posteriormente, se empleará *EF Core* para crear el almacenamiento en la base de datos correspondiente.

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

1. Como paso previo, conviene comprobar que los paquetes o extensiones de *NuGet* específicos que permiten utilizar *EF Core* están instalados en la Solución. Para ello:
 - a. En la barra de menú de Visual Studio, seleccionar la opción **Herramientas**, desplegar la opción **Administrador de paquetes NuGet** y, a continuación, seleccionar la opción **Administrar paquetes NuGet para la solución...**

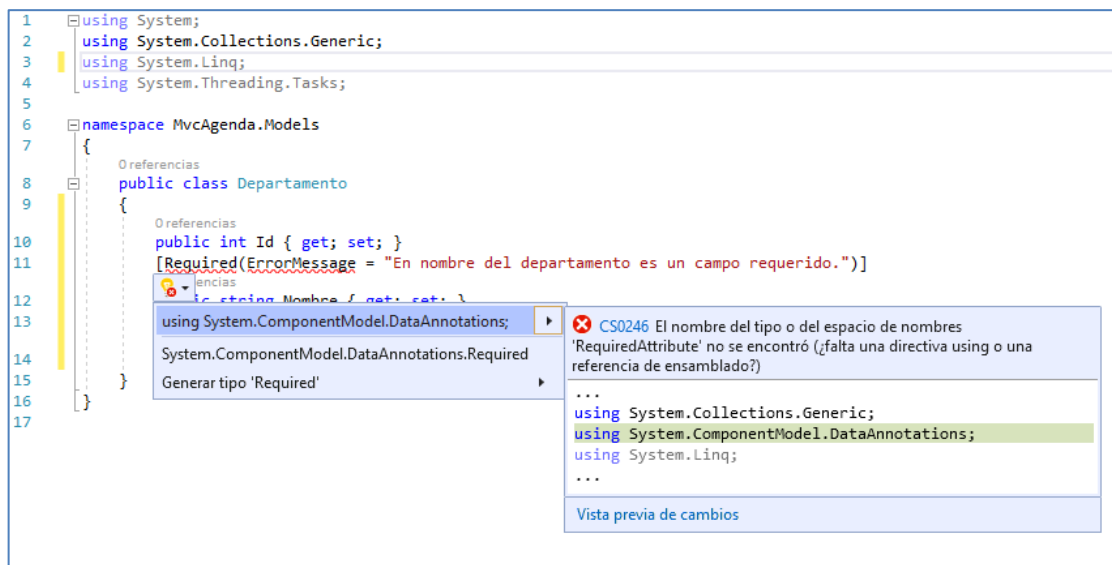
- b. En la ventana **NuGet – Solución**, seleccionar la opción **Instalado** y comprobar que están instalados varios paquetes que incluyen en su nombre *EntityFrameworkCore* e *Identity*. Entre ellos, *EntityFrameworkCore.SqlServer* y *EntityFrameworkCore.Tools*.
El Administrador de paquetes *NuGet* administra la instalación automática y desatendida de los paquetes de *NuGet* cuyo uso sea necesario en la solución.
2. Se utilizarán tres clases para 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 *Departamento.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 de la clase y añadir el siguiente código al archivo de clase creado:

```
public class Departamento
{
    public int Id { get; set; }
    [Required(ErrorMessage = "El nombre del departamento es un campo requerido.")]
    public string Nombre { get; set; }

    public ICollection<Empleado> Empleados { get; set; }
}
```

Como puede apreciarse en el código de la clase anterior, se han añadido anotaciones de validación de datos o *DataAnnotations* para incluir características específicas a algunas propiedades. Al añadir el código se podrá comprobar que se subrayarán en rojo las anotaciones de validación, indicando un error. Este error se produce porque falta añadir una directiva *using* que referencie el espacio de nombres correspondiente al uso de *DataAnnotations*. Para solucionarlo, 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, también podría haberse solucionado el error si se hubiera escrito directamente la línea de código anterior con la instrucción *using* correspondiente al principio del código de la clase.

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

```
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 = "Fecha de nacimiento")]
    [DataType(DataType.Date)]
    public DateTime FechaNacimiento { get; set; }
    [Display(Name = "Departamento")]
    public int DepartamentoId { get; set; }

    public Departamento Departamento { get; set; }
    public ICollection<Tarea> Tareas { get; set; }
}
```

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

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

```
public class Tarea
{
    public int Id { get; set; }
    [Display(Name = "Descripción")]
    public string Descripcion { get; set; }
    [Display(Name = "Fecha de inicio")]
    [Required(ErrorMessage = "La fecha de inicio es un campo requerido.")]
    [DataType(DataType.Date)]
    public DateTime FechaInicio { get; set; }
    [Display(Name = "Fecha de fin")]
    [DataType(DataType.Date)]
    public DateTime? FechaFin { get; set; }
    [Display(Name = "Hora de inicio")]
    [Required(ErrorMessage = "La hora de inicio es un campo requerido.")]
    [DataType(DataType.Time)]
    public DateTime HoraInicio { get; set; }
    [Display(Name = "Hora de fin")]
    [DataType(DataType.Time)]
    public DateTime? HoraFin { get; set; }

    [Display(Name = "Empleado")]
    public int EmpleadoId { get; set; }

    public Empleado Empleado { get; set; }
}
```

De manera predeterminada y atendiendo a la convención de nombres establecida, *EF Core* interpreta que una propiedad de una clase de datos que se denomina *Id* es **clave principal**. Así, por ejemplo, en el código anterior puede apreciarse que la propiedad *Id* de la clase de datos *Tarea* se convertirá en la columna clave principal de la tabla correspondiente. Además, esta convención establece que una propiedad se interpreta como **clave ajena**, si se denomina *EntidadId*. Así, por ejemplo, en el código anterior, la propiedad *EmpleadoId* de la clase de datos *Tarea* se convertirá en una columna de clave ajena en la tabla correspondiente. También puede apreciarse, la existencia de **propiedades de navegación** que permiten expresar una relación con otra entidad de datos. Así, por ejemplo, la propiedad de navegación *Empleado* en la entidad *Tarea* permite expresar la relación entre ambas entidades de datos y, en último término, entre las tablas correspondientes de la base de datos. Finalmente, también puede observarse la existencia de una **colección o lista de entidades de datos** que permite acceder a los datos relacionados de otra tabla. Así, por ejemplo, puede observarse la existencia de la propiedad *Tareas* de tipo *ICollection<Tarea>* en la clase de datos *Empleado*. Esta colección mantendrá la lista de entidades de tipo *Tarea* con las cuales una entidad de tipo *Empleado* se relaciona. Así, si un empleado tiene asignadas varias tareas, entonces la propiedad *Tareas* almacenará la información de las tareas correspondientes a ese Empleado. Es importante comprender el comportamiento predeterminado, así como las convenciones de nombres establecidas de *EF Core*. Ello permite aprovechar todo su potencial y, también, obtener la mayor productividad durante el trabajo de desarrollo de una aplicación Web. Se recomienda analizar y estudiar detenidamente el código de las clases de datos anteriores, de manera que se pueda comprender perfectamente los aspectos más relevantes del modelo especificado.

3. A continuación, se creará la **clase del contexto de datos**. El código de la clase del contexto de datos define las entidades de datos del modelo y permite personalizar el comportamiento de *EF Core* en el proyecto de ASP.NET Core MVC. La clase del contexto de datos se almacena en la carpeta */Data* del proyecto. Para crear la clase del contexto de datos, crear un archivo de clase denominado *MvcAgendaContexto.cs* en la carpeta */Data* y añadir el siguiente código:

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

    public DbSet<Departamento> Departamentos { get; set; }
    public DbSet<Empleado> Empleados { get; set; }
    public DbSet<Tarea> Tareas { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // 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;
        }
    }
}
```

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. Como puede apreciarse en el código anterior, **la clase del contexto de datos es una clase derivada de la clase *DbContext*** y especifica los conjuntos de entidades de datos que maneja la aplicación Web. De manera predeterminada y atendiendo a la convención de nombres establecida, *EF Core* interpreta que los nombres de las propiedades *DbSet* se usan como **nombres de tabla**. Así, los nombres de las tablas serán Departamentos, Empleados y Tareas, respectivamente. En la terminología de *EF Core*, una lista de entidades de datos se corresponde con una tabla de la base de datos o, también, se puede corresponder con el resultado de una consulta, una entidad se corresponde con una fila de la tabla y una propiedad se corresponde con una columna de la tabla. En el código anterior puede observarse, que se ha incluido código a través del método *OnModelCreating()*, para poder personalizar el comportamiento predeterminado de *EF Core*. El comportamiento predeterminado de *EF Core* establece habilitada la eliminación en cascada en tablas relacionadas entre sí. Este comportamiento predeterminado se suele querer evitar, de modo que se haga efectiva la restricción de integridad referencial más habitual, que consiste en mantener deshabilitada la eliminación en cascada. Para modificar este comportamiento se utiliza una repetición de tipo *foreach* para poder realizar un recorrido por cada una de las relaciones del modelo y, en cada caso, se modifica el valor de la propiedad *DeleteBehavior* al valor *DeleteBehavior.Restrict*.

4. Seleccionar la opción **Compilar MvcAgenda** del menú **Compilar** de Visual Studio para compilar el código de las clases añadidas y poder comprobar que no existen errores. El resultado de la compilación se muestra en la ventana de **Salida** de Visual Studio.

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

Todos los servicios de una aplicación Web basada en ASP.NET Core, como es, por ejemplo, el contexto de base de datos, deben registrarse en el contenedor de Inserción de Dependencias durante el inicio de la aplicación Web. El contenedor Inserción de Dependencias (DI) permite conectar fácilmente las clases y sus dependencias. Para registrar el contexto de la base de datos al iniciar la aplicación Web, realizar las siguientes acciones:

1. Abrir el archivo de clase *Startup.cs* que se encuentra ubicado en la carpeta raíz del proyecto.
2. Al final del método *ConfigureServices()*, añadir el código que aparece resaltado a continuación.

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(
                Configuration.GetConnectionString("DefaultConnection")));
    }
}
```

```
services.AddDefaultIdentity<IdentityUser>(options =>
    options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>();
services.AddControllersWithViews();
services.AddRazorPages();

// Registro del contexto de la base de datos
services.AddDbContext<MvcAgendaContexto>(options =>
    options.UseSqlServer(
        Configuration.GetConnectionString("DefaultConnection")));
}

// This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
}
```

En el código anterior puede apreciarse como se realiza el registro del contexto de datos *MvcAgendaContexto* que almacena la información que maneja la aplicación Web, relativa a departamentos, empleados y tareas. En el registro, se configura, como opción, el nombre de la cadena de conexión que va a ser utilizada para conectar este contexto de datos con una base de datos. Puede observarse que se establece la cadena de conexión *DefaultConnection*. Esta cadena de conexión ha sido creada de manera predeterminada y está disponible para ser utilizada por la aplicación Web para facilitar la conexión a la base de datos correspondiente.

Especificar la cadena de conexión de la base de datos

En el archivo *appsettings.json* se especifican las características de la cadena de conexión que enlaza la aplicación Web con la base de datos correspondiente. Para comprobar la especificación de las características de la cadena de conexión predeterminada, denominada *DefaultConnection*, abrir el archivo *appsettings.json* que se encuentra ubicado en la carpeta raíz del proyecto.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-MvcAgenda-07AC9E31-61F9-40E1-A81D-C53830CACE30;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

En el código del archivo *appsettings.json* puede apreciarse como se especifican las características de la cadena de conexión *DefaultConnection*, que se configura de manera predeterminada. Esta cadena de conexión va a ser utilizada por el contexto de datos *MvcAgendaContexto* para conectarse a la base de datos correspondiente. En el código puede apreciarse que se utilizará la instancia de base de datos

de SQL Server denominada *(localdb)\mssqllocaldb* y que se utilizará el archivo de base de datos de SQL Server denominado *aspnet-MvcAgenda-07AC9E31-61F9-40E1-A81D-C53830CACE30.mdf*, o cualquier nombre similar, para almacenar la información. De manera predeterminada, al crear la aplicación Web se asigna un nombre aleatorio al archivo de base de datos, por lo que muy posiblemente el nombre del archivo de base de datos de SQL Server será distinto al mostrado anteriormente. Es conveniente utilizar el nombre de archivo que se haya establecido aleatoriamente en cada caso, por lo que no es necesario modificar el código del archivo *appsettings.json*.

Migración inicial para crear la base de datos

En este momento del proceso de desarrollo de una aplicación web basada en ASP.NET Core MVC se va a crear la base de datos, para lo cual debe realizarse una **Operación de Migración**. Las migraciones son una característica de *EF Core* que incluyen el uso de un conjunto de herramientas que permiten crear y actualizar el esquema de una base de datos para que corresponda exactamente con la definición de las clases del modelo. Efectivamente, en todo momento, debe existir una correspondencia entre el modelo y el esquema de la base de datos. Si esta correspondencia no existiera, entonces se producirá un error en tiempo de ejecución al intentar acceder a la base de datos. En este momento del desarrollo, aún no se ha creado la base de datos, por lo que deberá realizarse una migración inicial para que pueda crearse la base de datos correspondiente. Esta migración inicial del modelo, por tanto, es la que produce la creación de la base de datos según la definición de las clases de modelo. Una vez creada la base de datos, cuando sea necesario modificar la definición de las clases de datos del modelo para, por ejemplo, añadir una nueva propiedad en una de ellas, deberá realizarse una migración para producir reflejo de la modificación del modelo en el esquema de la base de datos.

Para crear la base de datos mediante una migración inicial, realizar las siguientes acciones:

1. Como paso previo, conviene compilar el proyecto, mediante la opción **Compilar MvcAgenda** 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 (PCM)** ejecutar el siguiente comando:

```
PM> Add-Migration Initial -context MvcAgendaContexto
```

El comando **Add-Migration** genera un archivo de migración en la carpeta */Migrations* del proyecto con el nombre de la migración especificado. El primer argumento del comando, que en este caso es *Inicial*, es el nombre de la migración. Se puede usar cualquier nombre, pero, por convención, se selecciona un nombre que describa la migración. Como se trata de la migración inicial, la clase generada en la carpeta */Migrations* contendrá el código necesario para crear el esquema de la base de datos. El esquema de la base de datos a crear se basará en el modelo especificado en la clase del contexto de datos *MvcAgendaContexto*, tal como se especifica a través del valor de la opción *-context* de la instrucción de migración.

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

```
PM> Update-Database -context MvcAgendaContexto
```

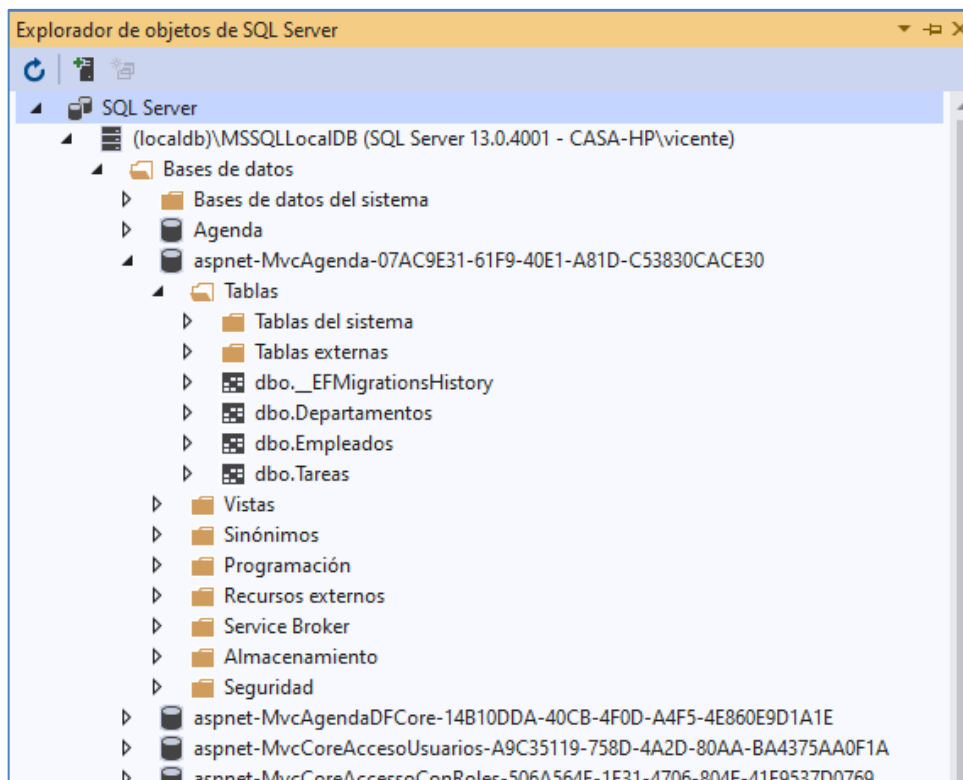

El comando **Update-Database** actualiza el esquema de la base de datos a la migración más reciente. En este caso, ejecuta el método *Up()* del archivo de migración *Inicial.cs* creado en la carpeta /Migrations, como consecuencia de la instrucción Add-Migration ejecutada en el apartado anterior. Al ejecutar la instrucción Update-Database especificada, se creará la base de datos que almacenará la información que maneja la aplicación Web.

Si en un futuro, se produjera algún cambio en el código de las clases del modelo especificado en el contexto de datos *MvcAgendaContexto*, entonces será necesario realizar una nueva migración para 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 y, también para poder interactuar con ella directamente, realizar las siguientes acciones:

1. En el menú **Ver**, abrir **Explorador de objetos de SQL Server** (SSOX). Podrá comprobarse que sobre la ventana correspondiente puede elegirse la opción **Ocultar automáticamente** para hacer que quede disponible en la barra vertical izquierda de Visual Studio.
2. Desplegar la opción **SQL Server**. A continuación, desplegar la opción **(localdb)\MSSQLLocalDB** y desplegar la opción **Bases de datos**.
3. Para acceder a los objetos de la base de datos, desplegar la opción correspondiente al nombre del archivo de la base de datos de SQL Server que está definido en el archivo *appsetting.json*. Este nombre será distinto en cada caso, debido al código numérico aleatorio que incorpora.



4. Al desplegar la opción **Tablas**, puede observarse que se muestran las tablas del contexto de datos que maneja la aplicación Web: **Departamentos**, **Empleados** y **Tareas**.
5. Para visualizar la información almacenada en cada tabla, seleccionar una de las tablas, por ejemplo, la tabla **Tareas**, hacer clic con el botón derecho y seleccionar la opción **Ver datos**. Evidentemente, la tabla está vacía porque aún no se ha introducido ninguna información.
6. También se puede visualizar la información de diseño de cada tabla. Para ello, seleccionar una de las tablas, por ejemplo, la tabla **Empleados**, hacer clic con el botón derecho y seleccionar la opción **Diseñador de vistas**. Es importante tener siempre presente que no convendrá de ningún modo y en ningún caso, modificar el diseño de las tablas directamente sobre la base de datos a través del Explorador de objetos de SQL Server. Si esto se hace, se perderá la coherencia con el modelo de la aplicación web y se producirá un error irreparable en la ejecución de la aplicación web. Si se desea modificar el esquema de datos, deberá modificarse el código correspondiente en las clases del modelo y, a continuación, realizar una operación de migración para producir reflejo de las modificaciones en el esquema de la base de datos.
7. De forma predeterminada, el archivo de la base de datos de SQL Server que almacena la información que maneja la aplicación Web se crea en la capeta del usuario, que en Windows es: c:/Usuarios/{nombre_usuario}. Para comprobarlo, abrir el **Explorador de archivos de Windows** y acceder a esta ubicación. Debe tenerse en cuenta que para realizar una copia de la información que maneja la aplicación web, deberán copiarse los dos archivos con extensión .mdf y .ldf que se corresponden con el nombre del archivo de base de datos de SQL Server establecido en la definición de la cadena de conexión del archivo *appsettings.json*.

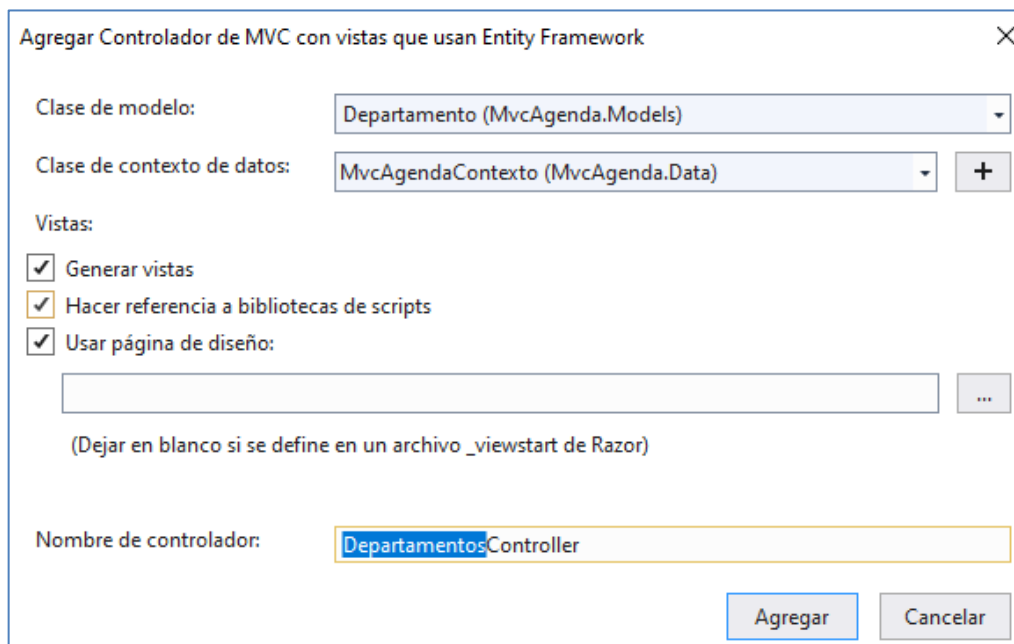
Crear un controlador, y sus vistas asociadas, para generar un procesamiento de tipo CRUD para cada una de las tablas de la base de datos que maneja la aplicación Web

En este apartado del ejercicio se desarrollará el procesamiento de tipo CRUD (*Create Read Update and Delete*) sobre cada una de las clases de datos que forman el modelo de la aplicación Web. El acrónimo CRUD se usa para referirse a un procesamiento que combina las operaciones básicas de manipulación de datos (Crear, Consultar, Modificar y Eliminar) sobre las filas o registros de una tabla de una base de datos. Este procesamiento típico de las aplicaciones informáticas, en general, facilita el mantenimiento de los datos almacenados las tablas que forman una base de datos. Los Proyectos de ASP.NET Core MVC disponen de un asistente que permite crear automáticamente los controladores y las vistas asociadas a los procesamientos de tipo CRUD para cada una de las tablas de la base de datos.

Para crear un procesamiento de tipo CRUD de la tabla *Departamentos*, realizar las siguientes acciones:

- ~~1. En primer lugar, es necesario compilar el proyecto de ASP.NET Core MVC para poner las clases de entidad y del contexto a disposición de los controladores que se van a crear. Para ello, seleccionar la opción **Compilar MvcAgenda** del menú **Compilar** de Visual Studio. Además, debe comprobarse que el resultado de la compilación no devuelve ningún error.~~
1. Para crear un procesamiento de tipo CRUD de la tabla *Departamentos*, se comienza creando un controlador. Para ello, en el **Explorador de soluciones**, hacer clic en el botón derecho sobre la carpeta */Controllers*, desplegar la opción **Agregar** y, a continuación, seleccionar la opción **Nuevo elemento con Scaffold...**
2. En la ventana **Agregar nuevo elemento con scaffolding**, seleccionar la opción **Controlador de MVC con vistas que usan Entity Framework** y hacer clic en **Agregar**.

3. En la ventana **Agregar Controlador de MVC con vistas que usan Entity Framework**, hacer:
 - a. Seleccionar *Departamento* como Clase de modelo.
 - b. Seleccionar *MvcAgendaContexto* como Clase de contexto de datos.
 - c. Comprobar que están activadas las tres casillas de verificación: Generar vistas, Hacer referencia a bibliotecas de scripts y Usar página de diseño.
 - d. Introducir *DepartamentosController* como Nombre de controlador. El nombre que se propone será *DepartamentoesController*, porque, de manera predeterminada, se pluralizan los nombres en inglés, lo cual no parece lo más adecuado en castellano.
 - e. Hacer clic en **Agregar**.



Finalizado el trabajo del asistente, se habrá creado el controlador *DepartamentosController.cs* en la carpeta */Controllers* y, las vistas asociadas en la subcarpeta de vistas */Views/Departamentos*. El controlador *DepartamentosController.cs* incluye las acciones que implementan las operaciones de manipulación de datos del procesamiento CRUD. Las acciones de un controlador especifican las operaciones que realiza ese controlador y pueden invocarse como acciones GET o POST. En general, las acciones GET se refieren a obtener información del servidor, mientras que las acciones POST se relacionan con enviar información al servidor desde el cliente. Además, cada método de acción del controlador activará la vista correspondiente. La zona o subcarpeta de vistas */Views/Departamentos* incluye las vistas que activan las acciones del controlador *DepartamentosController.cs*. Estas vistas permiten generar dinámicamente en el servidor Web, el conjunto de páginas Web con las que interactúa el usuario cuando utiliza el procesamiento CRUD de la tabla *Departamentos*.

En este punto del desarrollo, se puede probar el funcionamiento del procesamiento de tipo CRUD de la tabla *Departamentos*, solicitando la ejecución del controlador *DepartamentosController.cs* desde la barra de direcciones del navegador Web. Para ello, realizar las siguientes acciones:

1. Iniciar la depuración de la aplicación Web.

- Introducir el texto <http://localhost:xxxxx/Departamentos/Index> en la barra de direcciones del navegador Web para solicitar la acción *Index* del controlador *DepartamentosController.cs*. El valor *xxxxx* se refiere al número de puerto del servidor Web de depuración y será diferente en cada caso, dependiendo de las condiciones del entorno de ejecución. Cabe recordar que el formato del enrutamiento o mapeo MVC, y sus valores predeterminados, se encuentran especificados en el archivo *appsettings.json*. El enrutamiento MVC establecido de forma predeterminada es: [http://servidor:NúmeroPuerto/\[NombreControlador\]/\[NombreAcción\]](http://servidor:NúmeroPuerto/[NombreControlador]/[NombreAcción]). Y los valores predeterminados para el nombre del controlador es *Home* y para el nombre de la acción es *Index*. De modo que al introducir <http://localhost:xxxxx/Departamentos> en la barra de direcciones, se obtendrá en mismo efecto.

[MvcAgenda](#) [Inicio](#) [Política de privacidad](#) [Register](#) [Login](#)

Index

[Create New](#)

Nombre	
Sistemas de Información	Edit Details Delete
Producción	Edit Details Delete
Administración	Edit Details Delete
Calidad	Edit Details Delete
Marketing y Ventas	Edit Details Delete
Aprovisionamiento	Edit Details Delete

© 2020 - MvcAgenda. DWS/DAW. IES Mare Nostrum - [Política de privacidad](#)

- Probar el procesamiento CRUD para añadir, consultar, modificar y eliminar datos. Introducir información en la tabla *Departamentos*, de modo que se disponga de información almacenada suficiente para poder realizar pruebas de funcionamiento.
- Acceder al código del archivo *DepartamentosController.cs* para identificar y asimilar el uso del código en los distintos métodos de acción que conforman este controlador. Asimismo, acceder a los diversos archivos que se han creado en la carpeta */Views/Departamentos* para identificar y asimilar el uso de las instrucciones de código Razor de las vistas para crear las páginas.

5. Abrir el archivo *Index.cshtml* de la carpeta de vistas */Views/Departamentos* y modificar el código que aparece resaltado a continuación, para mejorar la presentación visual del texto que se mostrará en la página Web resultante.

```
@model IEnumerable<MvcAgenda.Models.Departamento>

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

<h1>Departamentos</h1>

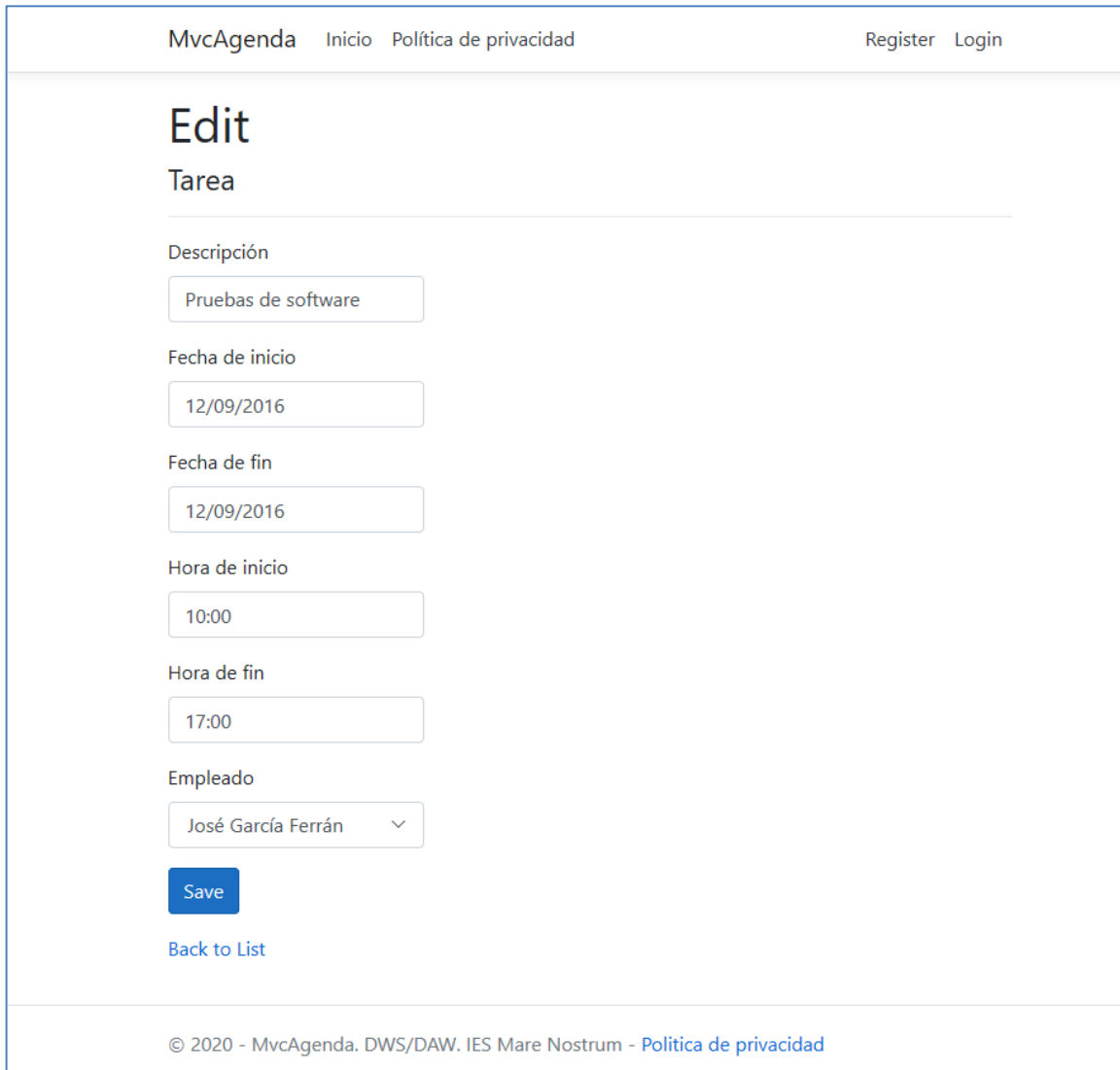
<p>
    <a asp-action="Create">Nuevo</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Nombre)
            </th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.Nombre)
                </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>
```

- ~~6. Iniciar nuevamente la depuración para comprobar los resultados.~~

Una vez finalizada la creación del procesamiento de tipo CRUD de la tabla *Departamentos*, a continuación, se agregarán los procesamientos de tipo CRUD del resto de las tablas de la base de datos al Proyecto de ASP.NET Core MVC. Para ello, realizar las siguientes acciones:

1. Agregar el procesamiento de tipo CRUD para la tabla *Empleados*, tal como se ha procedido anteriormente. En este caso, se creará el controlador *EmpleadosController.cs*, así como la carpeta de vistas asociada a este controlador.
2. Probar el procesamiento CRUD creado para añadir, consultar, modificar y eliminar datos.
3. Agregar el procesamiento de tipo CRUD para la tabla *Tareas*, tal como se ha procedido anteriormente. En este caso, se creará el controlador *TareasController.cs*, así como la carpeta de vistas asociadas a este controlador.
4. Probar el procesamiento CRUD creado para añadir, consultar, modificar y eliminar datos.

En la siguiente ilustración, puede apreciarse que, al crear un procesamiento de tipo CRUD de una tabla en la que hay campos que actúan como clave ajena, como ocurre en las tablas *Empleados* o *Tareas*, el asistente ajusta el código de los controladores y de las vistas implicadas, de manera que la implementación de la interfaz incorpora cuadros combinados que permiten seleccionar uno de los valores existentes en la tabla relacionada en los procesos de creación o edición de registros.



MvcAgenda Inicio Política de privacidad Register Login

Edit

Tarea

Descripción

Pruebas de software

Fecha de inicio

12/09/2016

Fecha de fin

12/09/2016

Hora de inicio

10:00

Hora de fin

17:00

Empleado

José García Ferrán

Save

[Back to List](#)

© 2020 - MvcAgenda. DWS/DAW. IES Mare Nostrum - [Política de privacidad](#)

Crear las opciones de menú para los controladores creados

Unos cambios sencillos en el archivo *_Layout.cshtml* permiten añadir los enlaces de las opciones de menú para la gestión de Departamentos, Empleados y Tareas. Para añadir las opciones de menú, hacer:

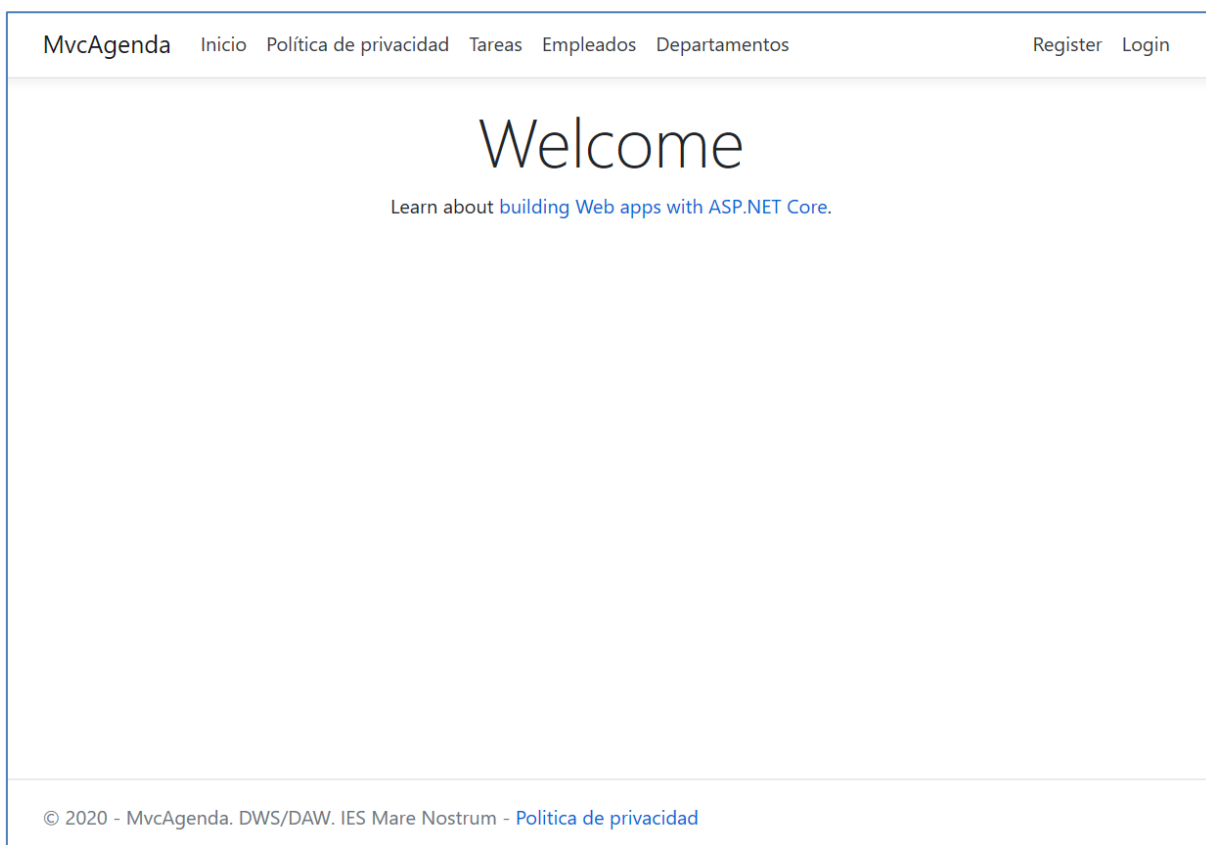
1. Abrir el archivo *_Layout.cshtml* situado en la carpeta */Views/Shared* que especifica la plantilla de diseño común de la aplicación web
2. Agregar el código que aparece resaltado a continuación.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - MvcAgenda</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
</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">
        <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">MvcAgenda</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
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 flex-sm-row-reverse">
          <partial name="_LoginPartial" />
          <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home"
asp-action="Index">Inicio</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Home"
asp-action="Privacy">Política de privacidad</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Tareas"
asp-action="Index">Tareas</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Empleados"
asp-action="Index">Empleados</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-controller="Departamentos"
asp-action="Index">Departamentos</a>
            </li>
          </ul>
        </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; 2020 - MvcAgenda. DWS/DAW. IES Mare Nostrum - <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>
  @RenderSection("Scripts", required: false)
</body>
</html>
```

Como puede comprobarse en el código anterior, se utilizan elementos de enlace de tipo `<a>` para facilitar a los usuarios la solicitud de ejecución de las acciones *Index* de los controladores *TareasController.cs*, *EmpleadosController.cs* y *DepartamentosController.cs*, según el mapeo MVC establecido. Puede apreciarse que se utilizan los atributos *asp-controller* y *asp-action* para especificar el nombre del controlador y el nombre de la acción, respectivamente, que se desea solicitar cuando el usuario active el enlace que se genera en la página resultante.

3. Iniciar la depuración para comprobar los cambios realizados, así como el funcionamiento de los procesamientos de tipo CRUD de las tablas *Departamentos*, *Empleados* y *Tareas*.
4. Si no se ha realizado ya anteriormente, **introducir información** en las tablas *Departamentos*, *Empleados* y *Tareas*. De este modo, se dispondrá de información almacenada suficiente para realizar las pruebas de funcionamiento. Se recomienda introducir unas 10 filas en cada tabla.



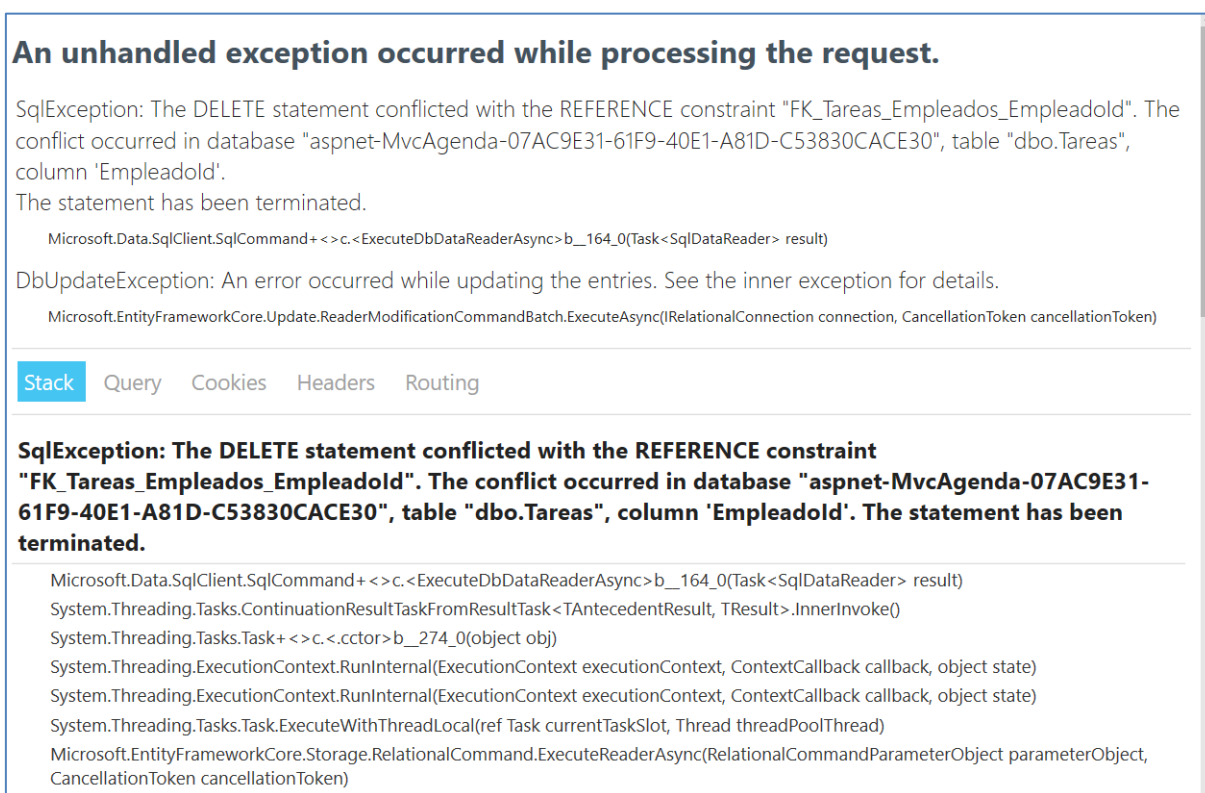
Como podrá apreciarse, el diseño visual que proporciona la plantilla de diseño es bastante minimalista y austero. Si se desea, puede actuarse sobre el código de la hoja de estilo externa denominada *site.css* que está ubicada en la carpeta `/wwwroot/css` del proyecto, para personalizar el diseño. A la hora de mejorar las presentaciones visuales de las interfaces, debe tenerse en cuenta que los objetivos de este módulo profesional se dirigen hacia estudio del procesamiento en el servidor y no hacia el diseño de interfaces Web, por lo que se recomienda que no se pierda excesivo tiempo en ello.

En los siguientes ejercicios de esta práctica y, sobre todo, en el tema siguiente se profundizará en diversos aspectos concretos del desarrollo de aplicaciones Web basadas en ASP.NET Core MVC.

Ejercicio 2

Gestión de errores en tiempo de ejecución a nivel de aplicación

En este ejercicio se aborda la gestión de errores en tiempo de ejecución sobre la aplicación Web basada en ASP.NET Core MVC *MvcAgenda*. Las aplicaciones Web basadas en ASP.NET Core MVC incorporan un mecanismo predeterminado de control de errores a nivel de aplicación. Puede comprobarse que al intentar eliminar un empleado que tiene tareas asignadas, se va a producir un error en tiempo de ejecución. Este error es consecuencia de las restricciones de integridad referencial especificadas en la base de datos. El efecto es que se muestra una página de error en la que se explica la naturaleza del error y muestra diversa información que puede ayudar al desarrollador a solucionar el problema.



An unhandled exception occurred while processing the request.

SqlException: The DELETE statement conflicted with the REFERENCE constraint "FK_Tareas_Empleados_Empleadold". The conflict occurred in database "aspnet-MvcAgenda-07AC9E31-61F9-40E1-A81D-C53830CACE30", table "dbo.Tareas", column 'Empleadold'.

The statement has been terminated.

Microsoft.Data.SqlClient.SqlCommand+<>c.<ExecuteDbDataReaderAsync>b__164_0(Task<SqlDataReader> result)

DbUpdateException: An error occurred while updating the entries. See the inner exception for details.

Microsoft.EntityFrameworkCore.Update.ReaderModificationCommandBatch.ExecuteAsync(IRelationalConnection connection, CancellationToken cancellationToken)

Stack Query Cookies Headers Routing

SqlException: The DELETE statement conflicted with the REFERENCE constraint "FK_Tareas_Empleados_Empleadold". The conflict occurred in database "aspnet-MvcAgenda-07AC9E31-61F9-40E1-A81D-C53830CACE30", table "dbo.Tareas", column 'Empleadold'. The statement has been terminated.

Microsoft.Data.SqlClient.SqlCommand+<>c.<ExecuteDbDataReaderAsync>b__164_0(Task<SqlDataReader> result)
System.Threading.Tasks.ContinuationResultTaskFromResultTask<TAntecedentResult, TResult>.InnerInvoke()
System.Threading.Tasks.Task+<>c.<ctor>b__274_0(object obj)
System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, object state)
System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, object state)
System.Threading.Tasks.Task.ExecuteWithThreadLocal(ref Task currentTaskSlot, Thread threadPoolThread)
Microsoft.EntityFrameworkCore.Storage.RelationalCommand.ExecuteReaderAsync(RelationalCommandParameterObject parameterObject, CancellationToken cancellationToken)

En la imagen anterior puede verse un ejemplo de página de error. En ella puede comprobarse la información que se presenta sobre la naturaleza del error producido, es importante para tratar de solucionar el problema en tiempo de desarrollo, aunque, sin embargo, carece de interés para el usuario final en tiempo de explotación de la aplicación Web.

La gestión de errores en tiempo de ejecución, sin embargo, tiene un doble objetivo. Por una parte, en tiempo de desarrollo, informar al desarrollador sobre la naturaleza del error producido para ayudar a solucionar el problema producido. Y, por otro lado, en tiempo de explotación o producción, informar de forma sencilla al usuario final de los errores que se puedan presentar en tiempo de ejecución. Las aplicaciones Web basadas en ASP.NET Core MVC incorporan, de manera predeterminada, la gestión de estas dos vertientes del tratamiento de errores en tiempo de ejecución.

En método *Configure* el archivo *Startup.cs*, puede observarse el código que permite realizar la gestión de los errores en tiempo de ejecución.

```
. . .  
  
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
        app.UseDatabaseErrorPage();  
    }  
    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();  
    . . .  
}
```

En el código anterior, puede apreciarse que se mostrará información de depuración dirigida al desarrollador, cuando se está trabajando con el entorno de desarrollo. O bien, se mostrará información destinada al usuario final, cuando se utiliza otro tipo de entorno. En efecto, el uso del método *IsDevelopment()* permite seleccionar la información de error que se mostrará, según el tipo de entorno de trabajo que se esté empleando en la ejecución actual de la aplicación Web.

Dado que se está utilizando el entorno de desarrollo, para poder comprobar qué información se mostrará al usuario final cuando se utilice un entorno de explotación, basta con comentar el código correspondiente, de la siguiente forma:

```
. . .  
  
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    //if (env.IsDevelopment())  
    //{  
    //    app.UseDeveloperExceptionPage();  
    //    app.UseDatabaseErrorPage();  
    //}  
    //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();  
    . . .  
}
```

Y, a continuación, ejecutar de nuevo la aplicación Web para generar el mismo error anterior.

[MvcAgenda](#) [Inicio](#) [Política de privacidad](#) [Tareas](#) [Empleados](#) [Departamentos](#) [Register](#) [Login](#)

Error.

An error occurred while processing your request.

Request ID: |84c01b4a-4f0b7c58f4ef7a19.

Development Mode

Swapping to **Development** environment will display more detailed information about the error that occurred.

The Development environment shouldn't be enabled for deployed applications. It can result in displaying sensitive information from exceptions to end users. For local debugging, enable the **Development** environment by setting the **ASPNETCORE_ENVIRONMENT** environment variable to **Development** and restarting the app.

© 2020 - MvcAgenda. DWS/DAW. IES Mare Nostrum - [Política de privacidad](#)

Como puede comprobarse, la información de error que va destinada al usuario final cuando se utiliza el entorno de explotación, es más sencilla y debe estar adaptada al tipo de usuario final de la aplicación Web. Para generar la página de error anterior se utiliza la vista *Error.cshtml* que está ubicada en la subcarpeta de vistas */Views/Shared*. El código de esta vista puede modificarse para personalizar el mensaje de los errores producidos en tiempo de ejecución que van destinados al usuario final.

Finalmente, editar de nuevo el código del archivo *Startup.cs* para deshacer los cambios realizados en el código al comentar el código, de manera que se pueda obtener también la información de error dirigida al desarrollador cuando se esté utilizando el entorno de desarrollo. Una vez que el código del archivo *Startup.cs* haya quedado en su estado original, a continuación, iniciar la depuración de la aplicación Web para comprobar los resultados obtenidos.

Ejercicio 3

Operaciones de migración empleando el enfoque *Code First* para el enlace a datos

En este ejercicio, se va a modificar la estructura de la clase de datos *Empleado* del modelo para poder adecuar sus propiedades a las necesidades de información requeridas. **En todo momento, debe existir una coherencia entre la estructura de datos del modelo y el esquema de la base de datos.** Cuando se emplea un enfoque *Code First* para el enlace a datos, como es el caso, debe realizarse una operación de migración para que se apliquen las modificaciones realizadas en el código que define las clases de datos del modelo, sobre el esquema de la base de datos que maneja la aplicación Web.

Operaciones de migración. Añadir nuevas propiedades a la clase de datos *Empleado*

Las operaciones de migración de *Entity Framework Core* facilitan que los cambios realizados en el código de las clases del modelo se reflejen sobre el esquema de la base de datos. En este ejercicio, se añadirán las propiedades *Telefono* y *Email* a la clase de datos *Empleado* y, además, se hará que la propiedad *FechaNacimiento* no sea obligatoria, de modo que admita valores *null*. Una vez realizados estos cambios en el código, a continuación, se realizará una operación de migración para que se produzca reflejo de estos cambios en el esquema de la base de datos que maneja la aplicación Web *MvcAgenda*. Para ello, realizar las siguientes acciones:

1. Abrir el archivo *Empleado.cs* de la carpeta */Models* para añadir y modificar el código resaltado que se muestra a continuación.

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

    public Departamento Departamento { get; set; }
    public ICollection<Tarea> Tareas { get; set; }
}
```

En el código anterior puede apreciarse que se añaden las nuevas propiedades *Telefono* y *Email* y, además, se añade un signo de cierre de interrogación '?' al final del tipo de datos *DateTime* correspondiente a la propiedad *FechaNacimiento* para conseguir que admita también valores *null* y, por tanto, no sea un campo requerido. El hecho de realizar estos cambios en la clase de datos *Empleado* supone cambiar la estructura de datos del modelo de la aplicación Web. Y, consecuentemente, produce discrepancias entre la estructura del modelo y el esquema de la

base de datos, de manera que si se ejecutará el controlador *EmpleadosController.cs* se producirá un error en tiempo de ejecución que es consecuencia de estas discrepancias. Para resolver este problema, cuando se emplea el enfoque *Code First*, es necesario realizar una **operación de migración** que ajuste el esquema de la base de datos a partir de la estructura de datos definida en el modelo de la aplicación Web.

2. Antes de comenzar a realizar la operación de migración, como paso previo, conviene compilar el proyecto mediante la opción **Compilar MvcAgenda** del menú **Compilar** para asegurarse de que no existen errores de compilación en el código.
3. Abrir la Consola del Administrador de paquetes para activar, generar y aplicar las operaciones de migración. Para ello, desplegar la opción **Administrador de paquetes NuGet** del menú **Herramientas** y, seleccionar la opción **Consola del Administrador de paquetes**.
4. En la ventana **Consola del Administrador de paquetes** introducir el siguiente comando, para comparar el modelo actual con el esquema de la base de datos existente y crear el archivo de migración que incluye el código necesario para migrar la base de datos a partir del modelo:

PM> Add-Migration TelefonoEmail -context MvcAgendaContexto

El nombre que aparece como primer argumento del comando es arbitrario y se utiliza para nombrar esta migración en concreto. Se recomienda utilizar un nombre significativo. Cuando finalice la ejecución de este comando, podrá comprobarse mediante el **Explorador de soluciones** que se ha añadido a la carpeta /Migrations un nuevo archivo cuyo nombre finaliza con el nombre dado para la migración. Este archivo se conoce como archivo de migración y define una clase derivada de la clase *DbMigration*, de manera que en el método *Up()* de esta clase, se especifican los cambios a realizar sobre el esquema de la base de datos.

5. Si se desea, se puede compilar nuevamente el proyecto para asegurarse que no se produce ningún error de compilación.
6. Acceder nuevamente a la **Consola del Administrador de paquetes** e introducir el siguiente comando para ejecutar las instrucciones especificadas en el archivo de migración:

PM> Update-Database -context MvcAgendaContexto

7. Una vez finalizada la ejecución del comando anterior, mediante el **Explorador de objetos de SQL Server** comprobar que se ha modificado la definición de la tabla *Empleados* de la base de datos a la nueva estructura del modelo.
8. Mediante el **Explorador de objetos de SQL Server** comprobar también que los nuevos campos añadidos a la tabla *Empleados*, evidentemente, no tienen ninguna información almacenada.
9. Finalmente, es necesario adecuar el procesamiento de tipo CRUD de la tabla *Empleados* considerando las nuevas características y propiedades del modelo. Para ello, realizar las siguientes acciones:
 - a. Eliminar el controlador *EmpleadosController.cs* de la carpeta /Controllers.
 - b. Eliminar la subcarpeta de vistas /Views/Empleados y todo su contenido.
 - c. Volver a crear el controlador *EmpleadosController.cs*, y sus vistas asociadas, para así poder implementar un procesamiento de tipo CRUD adecuado a la nueva definición del modelo y que incorpore los cambios realizados en la información.
10. Iniciar la depuración para comprobar que el nuevo CRUD de la tabla *Empleados* funciona correctamente.

MvcAgenda	Inicio	Política de privacidad	Tareas	Empleados	Departamentos	Register	Login
<h2>Empleados</h2> Nuevo							
Nombre	Fecha de nacimiento	Teléfono	Correo electrónico	Departamento			
José García Ferrán	12/12/1978	699777333	jgarcia@empresa.com	Sistemas de Información	Editar	Detalles	Eliminar
Juan Sirvent Picó	21/08/1969	566333888	jsirvent@empresa.com	Producción	Editar	Detalles	Eliminar
Eva Martínez Juárez	21/05/1977	866111333	emartinez@empresa.com	Sistemas de Información	Editar	Detalles	Eliminar
Rosa García Gomez de Parga	14/03/1969	600222333	rgarcia@empresa.com	Administración	Editar	Detalles	Eliminar
Vicente Fernández López	21/02/1978	866555444	vfernandez@empresa.com	Producción	Editar	Detalles	Eliminar
Juan Llopis Pérez	24/12/1996	655888999	jllopis@empresa.com	Calidad	Editar	Detalles	Eliminar
María Mirambell Lucas	13/05/1969	955666777	mmirambell@empresa.com	Administración	Editar	Detalles	Eliminar
Pablo Moreno Sánchez	15/03/1988	633222111	pmoreno@empresa.com	Calidad	Editar	Detalles	Eliminar
Tamara Juárez	15/10/1997	611222333	tjuarez@empresa.com	Marketing y Ventas	Editar	Detalles	Eliminar

11. Iniciar la depuración y, a continuación, introducir información en los campos añadidos de los empleados ya existentes.

En los casos en que se modifique la estructura o las características de las propiedades de varias clases de datos, será necesario eliminar todos los procesamientos de tipo CRUD implicados. En estos casos, en primer lugar, se realizarán los cambios en el código de las clases de datos del modelo. A continuación, se realizará la operación de migración. En tercer lugar, se eliminarán los controladores y las subcarpetas de vistas que implementan los procesamientos de tipo CRUD de todas las tablas afectadas por los cambios. Y, finalmente, se generarán nuevamente los procesamientos de tipo CRUD de las tablas afectadas.