UD6 Desarrollo de aplicaciones web con ASP.NET Core.

6.1 Aplicación de ejemplo.

En estos momentos ya hemos adquirido todos los conocimientos básicos necesarios, por lo que estamos casi preparados para afrontar el desarrollo de una aplicación completa.

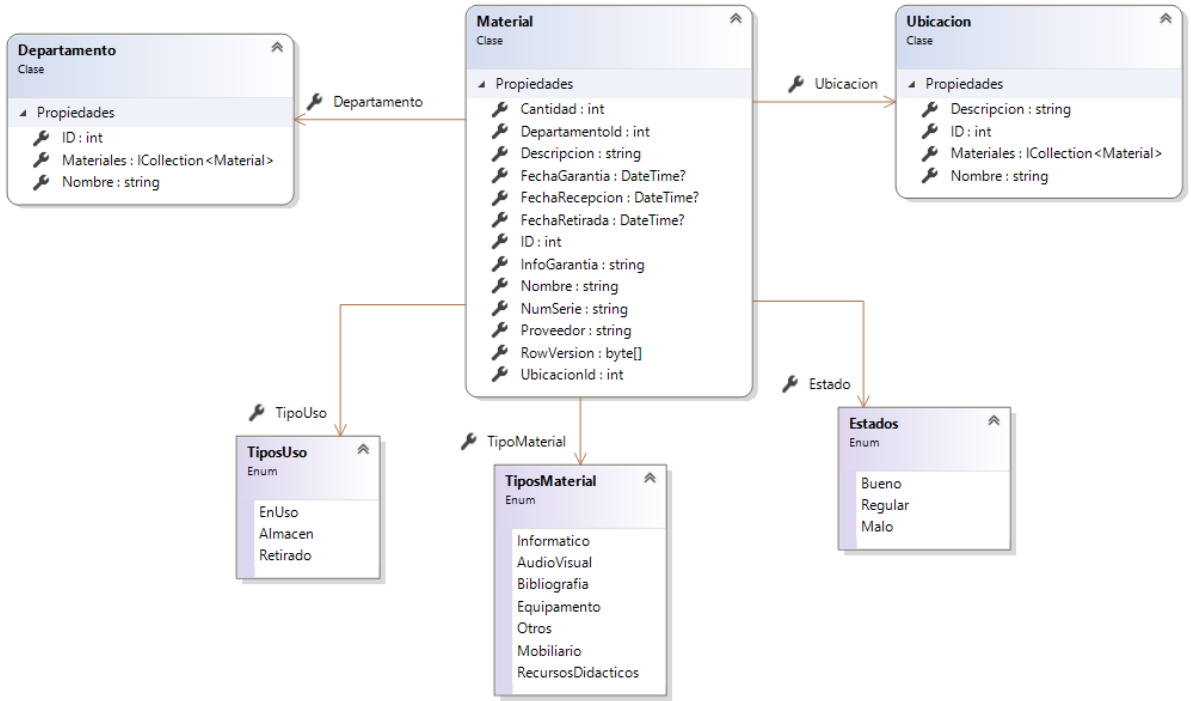
En esta unidad crearemos una simple aplicación de inventario, que nos permitirá afrontar los problemas típicos que nos podemos encontrar en el desarrollo de una aplicación web, sin ser demasiado compleja como para desarrollarla en una única unidad.

Vamos a suponer que necesitamos crear una aplicación para gestionar el inventario del centro y desarrollaremos la misma paso a paso.

Creación del proyecto y modelo de datos.

1. Creamos un nuevo proyecto de ASP.NET MVC Core a partir de la plantilla MVC.

2. A continuación pasamos a crear las entidades de nuestro modelo de datos, que para nuestra aplicación serán las que se muestran en el siguiente diagrama de clases:



3. Todo el material inventariable pertenecerá a un departamento, por lo que necesitamos almacenar los departamentos del centro en la BD.

   Comenzamos creando la clase *Departamento*, en la carpeta Models:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace CIFPACarballeiraWeb.Models
{
    public class Departamento
    {
        public int ID { get; set; }

        [Display(Name = "Nombre")]
        [Required(ErrorMessage = "Se debe introducir el nombre del departamento")]
        public string Nombre { get; set; }

        public List<Material> Materiales { get; set; }
    }
}
```

El atributo *Display* es de utilidad porque permite definir un texto que se mostrará en nuestros formularios en lugar del nombre del campo, de forma que podremos utilizar descripciones mucho más precisas, incluyendo espacios en blanco o signos de puntuación. Más adelante veremos cómo utilizar esta característica.
La propiedad *Materiales* es una propiedad de navegación, que nos permite establecer una relación entre dos entidades del modelo de datos. En este caso, se trata de una relación de tipo *1:n*, de forma que para cada departamento podremos tener multitud de materiales inventariados, mientras que cada elemento del inventario sólo podrá pertenecer a un único departamento. Gracias a esta propiedad, será muy sencillo obtener los datos de un departamento concreto, junto con los materiales del mismo.

4. Del mismo modo que los materiales pertenecen a un único departamento, queremos saber en quó localización se encuentra cada elemento inventariable. Por esta razón definimos una entidad *Ubicacion*, gracias a la que almacenaremos en nuestra BD todas las localizaciones del centro.

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace CIFPACarballeiraWeb.Models
{
    public class Ubicacion
    {
        public int ID { get; set; }

        [Display(Name = "Ubicación")]
        [Required(ErrorMessage = "Debe introducir el nombre de la ubicación")]
        public string Nombre { get; set; }

        [Display(Name = "Descripción")]
        public string Descripcion { get; set; }

        public List<Material> Materiales { get; set; }
    }
 }
```

Como en el caso anterior, encontramos una relación *1:n* entre los materiales que vamos a inventariar y la ubicación en la que se encuentran. Como para una ubicación podemos tener varios materiales, la propiedad de navegación será de tipo lista.

5. Algo muy habitual cuando desarrollamos una aplicación es encontrarnos con propiedades o campos de una entidad, cuyos valores están limitados entre una serie de ellos permitidos. En nuestro caso estas propiedades se refieren al estado del material, al tipo del mismo y al uso que se le está dando.
   Para estas situaciones, lo más sencillo es definir una enumeración o Enum, que contenga los valores permitidos, y utilizar esta enumeración como tipo de la propiedad. De este modo restringimos los valores que podemos introducir, y EF almacenará el valor en la BD como si fuera un texto, sin necesidad de preocuparnos por la conversión. Sin embargo, también tiene una serie de inconvenientes, que veremos a continuación.
   A la hora de definir estas enumeraciones disponemos de diferentes opciones, que dependen únicamente del gusto de los diferentes desarrolladores. Hay quien prefiere hacer como con el resto de entidades, definiendo cada enumeración en un único fichero. Nosotros utilizaremos otra aproximación, para no generar tantos ficheros que contienen una simple enumeración. Vamos a crear un único fichero llamado *Enums.cs* en *Models*, que contendrá todas las enumeraciones de nuestro modelo de datos.

```
using System.ComponentModel.DataAnnotations;

namespace CIFPACarballeiraWeb.Models
{
    public enum TiposMaterial
    {
        [Display(Name = "Informático")]
        Informatico,
        [Display(Name = "Audiovisual")]
        AudioVisual,
        [Display(Name = "Bibliografía")]
        Bibliografia,
        [Display(Name = "Equipamiento")]
        Equipamento,
        [Display(Name = "Otros")]
        Otros,
        [Display(Name = "Mobiliario")]
        Mobiliario,
        [Display(Name = "Recursos didácticos")]
        RecursosDidacticos
    }

    public enum TiposUso
    {
        [Display(Name = "En uso")]
        EnUso,
        [Display(Name = "Almacén")]
        Almacen,
        [Display(Name ="Retirado")]
        Retirado

    }
    public enum Estados
    {
        [Display(Name = "Bueno")]
        Bueno,
        [Display(Name = "Regular")]
        Regular,
        [Display(Name = "Malo")]
        Malo
    }
}
```

6. Por último, creamos la entidad más importante, que contendrá la información relativa a todo el material inventariable.

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

namespace CIFPACarballeiraWeb.Models
{
    public class Material
    {
        public int ID { get; set; }

        [StringLength(60, MinimumLength = 1, ErrorMessage = "Debe introducir el
        nombre del material inventariable")]
        [Display(Name = "Nombre")]
        public string Nombre { get; set; }

        [Display(Name = "Número de serie")]
        public string NumSerie { get; set; }

        [Display(Name = "Tipo de material")]
        [EnumDataType(typeof(TiposMaterial))]
        public TiposMaterial TipoMaterial { get; set; }

        [Display (Name ="Descripción")]
        public String Descripcion { get; set; }

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

        [Display (Name="Proveedor")]
        public String Proveedor { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}",
            ApplyFormatInEditMode = true)]
        [Display(Name = "Fecha de recepción")]
        public DateTime? FechaRecepcion { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}",
            ApplyFormatInEditMode = true)]
        [Display(Name = "Fecha de garantía")]
        public DateTime? FechaGarantia { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:dd-MM-yyyy}",
            ApplyFormatInEditMode = true)]
        [Display(Name = "Fecha de retirada")]
        public DateTime? FechaRetirada { get; set; }

        [Display(Name ="Contacto en garantía")]
        public string InfoGarantia { get; set; }

        [Display(Name = "Situación de uso")]
        [EnumDataType(typeof(TiposUso))]
        public TiposUso TipoUso { get; set; }

        [Display(Name = "Estado")]
        [EnumDataType(typeof(Estados))]
        public Estados Estado { get; set; }
```

```
        [Display(Name = "Departamento")]
        public Departamento  Departamento { get; set; }
        public int DepartamentoId { get; set; }

        [Display(Name = "Departamento")]
        public Ubicacion Ubicacion { get; set; }
        public int UbicacionId { get; set; }
    }
}
```

El atributo *EnumDataType* permite especificar que los datos permitidos para la propiedad están definidos en una enumeración, por lo que el asistente insertará un campo de lista de selección en nuestros formularios.
El campo *DataType* nos permite indicar tipos de datos más específicos que los que utilizamos al definir nuestras propiedades. En este caso, en C# disponemos del tipo *DateTime*, pero no nos interesa para nada el tiempo, simplemente la fecha. De esta forma definimos el tipo Date, para que el asistente genere el control de formulario adecuado. El atributo *DisplayFormat* permite definir el formato de fecha que queremos utilizar.
Las propiedades *DepartamentoId* y *UbicacionId* definen el otro extremo de la relación 1:n, al corresponder a cada material una única ubicación y un departamento al que pertenece.

Contexto de datos y cadena de conexión.

1. Vamos a empezar por el contexto de datos. Creamos una carpeta *Data*, y en ella creamos una nueva clase a la que llamaremos *CFIPACarballeiraContext*:

```
using Microsoft.EntityFrameworkCore;
using CIFPACarballeiraWeb.Models;

namespace CIFPACarballeiraWeb.Data
{
    public class CIFPACarballeiraContext : DbContext
    {
        public CIFPACarballeiraContext(
            DbContextOptions<CIFPACarballeiraContext> options) :
            base(options) { }

        public DbSet<Departamento> Departamentos { get; set; }
        public DbSet<Ubicacion> Ubicaciones { get; set; }
        public DbSet<Material> Materiales { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Ubicacion>().ToTable("Ubicaciones");
            modelBuilder.Entity<Material>().ToTable("Materiales");
        }
    }
}
```

En el método *OnModelCreating* utilizamos la denominada FluentAPI para especificar determinados aspectos que se tendrán en cuenta a la hora de crear la BD correspondiente al modelo de datos. EF crea por defecto los nombres de tablas a partir del nombre de cada entidad, agregando una 's' al final, para pluralizar estos nombres. El problema es que en determinados

nombres esto no funciona, y por esta razón especificamos los nombres para las tablas de ubicaciones y de materiales.

2. Una vez definido el contexto de datos, debemos especificar una cadena de conexión en *appsettings.json*.:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
       "CIFPACarballeiraContext":
       "Server=(localdb)\\mssqllocaldb;Database=CIFPACarballeira;Trusted_Connection=
       True;MultipleActiveResultSets=true"
  }
}
```

3. Por último, añadimos nuestro contexto de datos a los servicios, en la clase *Startup.cs*:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
    // This lambda determines whether user consent for non-essential cookies is
    needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);

    services.AddDbContext<CIFPACarballeiraContext>(
        options => options.UseSqlServer(
            Configuration.GetConnectionString("CIFPACarballeiraContext")));
    }
}
```

Creación de la migración inicial y la BD.

1. Si no está visible, comenzamos por mostrar la **Consola del Administrador de paquetes** en Visual Studio.

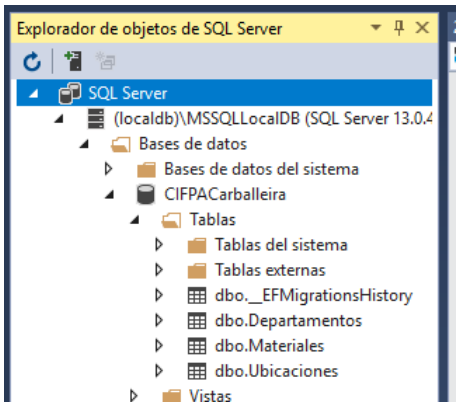
2. Desde la consola, crearemos la migración inicial:

```
Add-Migration InitialCreate
```

3. Ya podemos crear la base de datos conforme a nuestro modelo, recogido en la migración:

```
Update-Database
```

4. Si todo ha sido correcto, podremos ver la BD en el explorador de objetos de SQL Server.



Departamentos.

Comenzaremos nuestra aplicación por lo más sencillo, desarrollando en primer lugar el controlador y las vistas necesarias para crear, modificar y eliminar los departamentos del centro.
Antes de empezar a trabajar con los departamentos propiamente, vamos a hacer algunos cambios en nuestro proyecto.

1. Eliminamos la vista *Home/Privacy*, ya que no necesitaremos ninguna política de privacidad.

2. Realizamos los siguientes cambios en el fichero *_Layout.cshtml*. Comenzamos por la cabecera:

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - CIFP A Carballeira</title>

    <environment include="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-
bootstrap/4.1.3/css/bootstrap.min.css"
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
              asp-fallback-test-class="sr-only"
              asp-fallback-test-property="position"
              asp-fallback-test-value="absolute" crossorigin="anonymous"
              integrity="sha256-eSi1q2PG6J7g7ib17yAaWMcrr5GrtohYChqibrV7PBE="/>
    </environment>
    <link rel="stylesheet" href="~/css/site.css" />
    <link rel="stylesheet"
          href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
    integrity="sha384-  fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
          crossorigin="anonymous">
 </head>
```

   Cambiamos el título de la aplicación en el navegador y añadimos soporte para poder utilizar FontAwesome.

3. A continuación modificamos el encabezado de nuestras vistas:

```
<header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light
        bg-white border-bottom box-shadow mb-3">
```

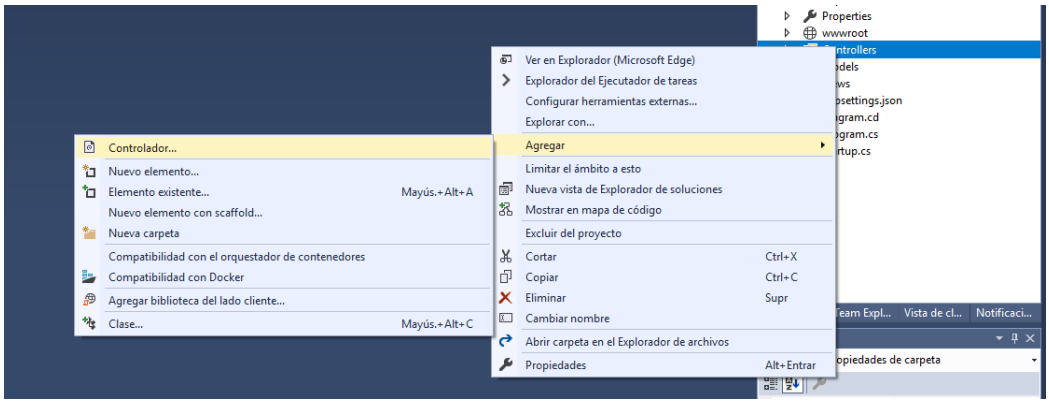
```
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home"
                    asp-action="Index">CIFP A Carballeira</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse"
                    data-target=".navbar-collapse"
                    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">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area=""
                                asp-controller=" Departamentos "
                                asp-action="Index">
                                <i class="fas fa-graduation-cap"></i> Departamentos</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
```

Cambiamos el texto del enlace a la página de inicio, modificamos el primer elemento de la lista para que apunte a la vista principal de departamentos y eliminamos el enlace a la página de la política de privacidad que hemos eliminado.

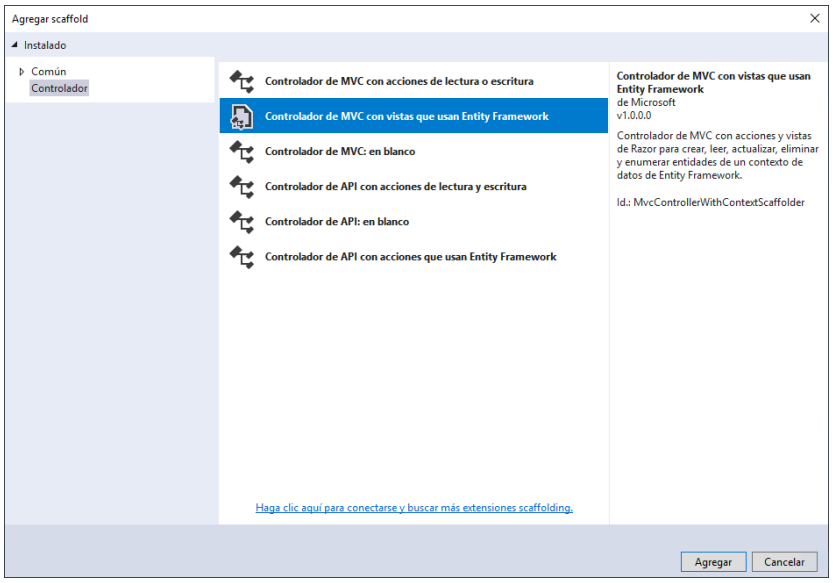
4. Por último, cambiamos también el pie de página:

```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2019 - CIFP A Carballeira
        </div>
    </footer>
```

5. Comenzamos haciendo click derecho en nuestra carpeta Controllers, y seleccionando la opción **Agregar | Controlador...**

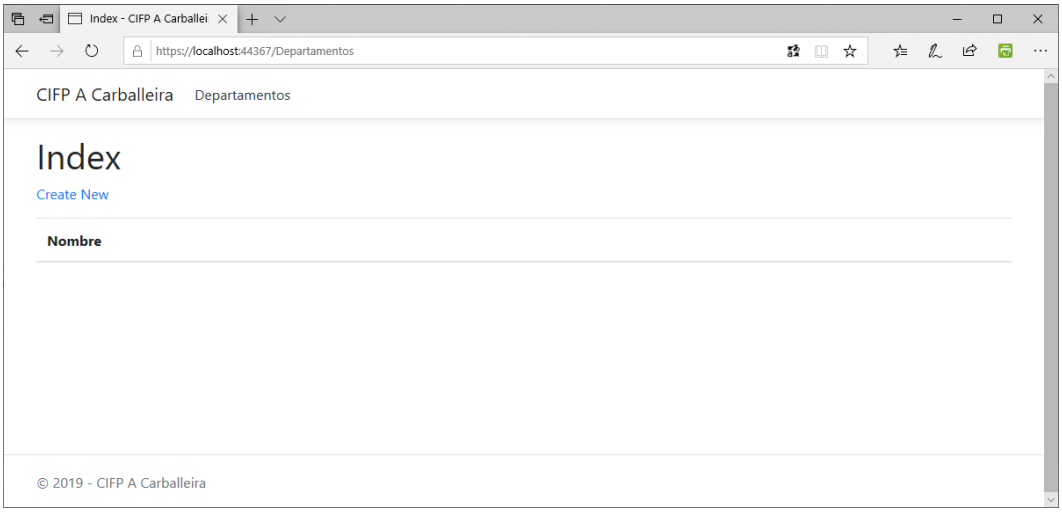


6. En el asistente, seleccionamos el controlador de MVC con vistas que usan Entity Framework.



7. Como clase de modelo seleccionaremos Departamento y en el segundo desplegable CIFPACarballeiraContext. Llamaremos al controlador *DepartamentosController*. Finalmente pulsamos en **Agregar**.

8. Si ejecutamos en este momento nuestra aplicación y pulsamos en el enlace a Departamentos de la cabecera, se mostrará la vista principal que muestra la lista de departamentos del centro, generada por el asistente.



Nuevo departamento.

La entidad departamento es muy sencilla, ya que la única propiedad que nos interesa es su nombre. Por lo tanto, el código que genera el asistente está prácticamente listo. Además, al haber añadido las anotaciones al modelo, tenemos de forma automática el código de validación del lado del servidor y del cliente.

1. Haremos únicamente algunas modificaciones simples en la presentación, editando la vista *Create.cshtml* de *Departamentos*:

```
@model CIFPACarballeiraWeb.Models.Departamento

@{
    ViewData["Title"] = "Nuevo departamento";
}

<h4>Nuevo departamento</h4>
<hr />
<div class="row">
    <div class="col-md-8">
        <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">
                <input type="submit" value="Crear" class="btn btn-primary" /> |
                <a asp-action="Index"> <i class="fas fa-undo-alt"></i> Volver</a>
            </div>
        </form>
    </div>
</div>

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

Vista de detalle.

Para una entidad tan simple como nuestro departamento, no tiene demasiado sentido ofrecer una vista de detalle, ya que toda la información de los departamentos, que consiste únicamente en el nombre, se mostrará en la página principal con la lista de departamentos. Por esta razón, en este caso eliminaremos tanto la vista como el método *Details* del controlador.
También eliminaremos de la vista Index el enlace a los detalles de cada departamento.

Editar departamento.

Al igual que en el caso de la creación de un nuevo departamento, el código proporcionado por el asistente no necesita ser modificado para la edición de departamentos, por lo que únicamente realizaremos pequeños retoques en la presentación.

```
@model CIFPACarballeiraWeb.Models.Departamento

@{
    ViewData["Title"] = "Editar departamento";
}

<h4>Editar departamento</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>
            <div class="form-group">
                <input type="submit" value="Guardar" class="btn btn-primary" /> |
                <a asp-action="Index"> <i class="fas fa-undo-alt"></i> Volver</a>
            </div>
        </form>
    </div>
</div>

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

Eliminar departamento.

El código generado por los asistentes resulta de gran utilidad, especialmente en casos sencillos, como hemos podido comprobar en todas las operaciones que hemos revisado del controlador de departamentos. A continuación se muestra la vista con ligeras modificaciones, el código del controlador permanecerá tal y como está.

```
@model CIFPACarballeiraWeb.Models.Departamento

@{
    ViewData["Title"] = "Eliminar departamento";
}

<h3>Eliminar departamento</h3>

<h4>Estás seguro de eliminar el departamento?</h4>
<div>
    <hr />
    <dl class="row">
        <dt class = "col-sm-1">
            @Html.DisplayNameFor(model => model.Nombre)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.Nombre)
        </dd>
    </dl>

    <form asp-action="Delete">
        <input type="hidden" asp-for="ID" />
        <input type="submit" value="Eliminar" class="btn btn-danger" /> |
        <a asp-action="Index"><i class="fa fa-backward"></i> Volver</a>
    </form>
</div>
```

Lista de departamentos.

La vista principal de departamentos muestra la lista de todos los departamentos del centro, además de enlaces para crear nuevos departamentos, así como editar o eliminar los existentes. A continuación se muestra el código de la vista con modificaciones relacionadas únicamente con la presentación:

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

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

<h2>Departamentos</h2>

<p>
    <a asp-action="Create"><i class="fa fa-plus-circle"></i> Nuevo departamento</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Nombre)
            </th>
            <th></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">
                      <i class="fa fa-edit" title="Editar"></i></a> |
                <a asp-action="Delete" asp-route-id="@item.ID">
                      <i class="fas fa-trash-alt" title="Eliminar"></i></a>
            </td>
        </tr>
       }
    </tbody>
</table>
```

Ubicaciones.

Ahora que hemos terminado con los departamentos, pasaremos a las ubicaciones. Se trata de una entidad muy similar a los departamentos, con una única propiedad, que es el nombre, con la que va a trabajar el usuario. Por lo tanto, para implementar la parte de ubicaciones haremos igual que con los departamentos.

Añadimos un nuevo enlace a ubicaciones en *_Layout.cshtml*, a continuación de departamentos:

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area=""
                asp-controller="Departamentos" asp-action="Index">
                <i class="fas fa-graduation-cap"></i> Departamentos</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area=""
                asp-controller="Ubicaciones" asp-action="Index">
                <i class="fas fa-building"></i> Ubicaciones</a>
        </li>
```

```
        </ul>
    </div>
```

El resto de pasos quedan como ejercicio, siguiendo los mismos pasos descritos para el caso de los departamentos.