UD3 ASP.NET Core MVC.
3.3 Vistas.

Las vistas son la salida de una aplicación, que es mostrada al usuario. Son lo que el usuario realmente verá en su pantalla y el medio por el cual accederá e interactuará con la aplicación. La experiencia de usuario a la hora de utilizar la aplicación es fundamental para el éxito de la misma, por lo que las vistas desempeñan una labor fundamental en nuestras aplicaciones. En una aplicación web son responsables de generar la salida para una petición, devolviendo habitualmente contenido HTML/CSS.

Ficheros de vistas y su localización.

Las vistas en ASP.NET MVC son ficheros con la extensión *.cshtml*, lo que implica que contienen tanto código C# (cs de C sharp) y HTML.

Por convención, se guardarán todos en la carpeta *Views*. Dentro de esta carpeta tendremos una serie de subcarpetas, siendo el nombre de cada una el mismo que el de cada controlador al que pertenecen las vistas. Por ejemplo, si tenemos en nuestro proyecto un controlador llamado *HomeController*, las vistas para este controlador se encontrarán dentro de la carpeta *Views/Home*.

El nombre del fichero, a su vez, tendrá el nombre de la acción o método del controlador para el cual generará la salida. Hemos visto en varias ocasiones que es habitual que una acción termine con el código `return View();`

Esta línea indica a ASP.NET Core MVC que debe buscar un fichero cuyo nombre sería *<nombre de acción>.cshtml* que se encuentra en la carpeta *Views/<nombre del controlador>.* Si por ejemplo tuviéramos un controlador *HomeController* y estamos en el método de acción *Index(),* el fichero de la vista sería *Views/Home/Index.cshtml*.

El motor de vistas Razor.

Como ya hemos explicado, un navegador web sólo es capaz de entender páginas HTML, hojas de estilos CSS y código del lado cliente JavaScript. El propósito del motor de vistas es generar HTML y enviarlo de vuelta al navegador del cliente.

El motor de vistas Razor es el motor de vistas por defecto utilizado en ASP.NET Core. Gracias a este motor podremos mezclar código C# y HTML en nuestras vistas, siendo el motor Razor capaz de distinguir entre los dos y generar la salida esperada. En determinados escenarios será necesario proporcionar información adicional, pero en la gran mayoría de los casos, bastará con indicar el comienzo de un bloque de código C# mediante el símbolo '@', sin necesidad de un símbolo de cierre.

Programar en una vista Razor es como programar normalmente en C#, pudiendo utilizar declaraciones de variables, construcciones condicionales o bucles por ejemplo. La única diferencia es que en una vista Razor, el código C# se combinará con código HTML para generar la salida deseada.

En el siguiente enlace podemos consultar la sintaxis Razor completa, y es muy conveniente tener esta documentación como referencia.

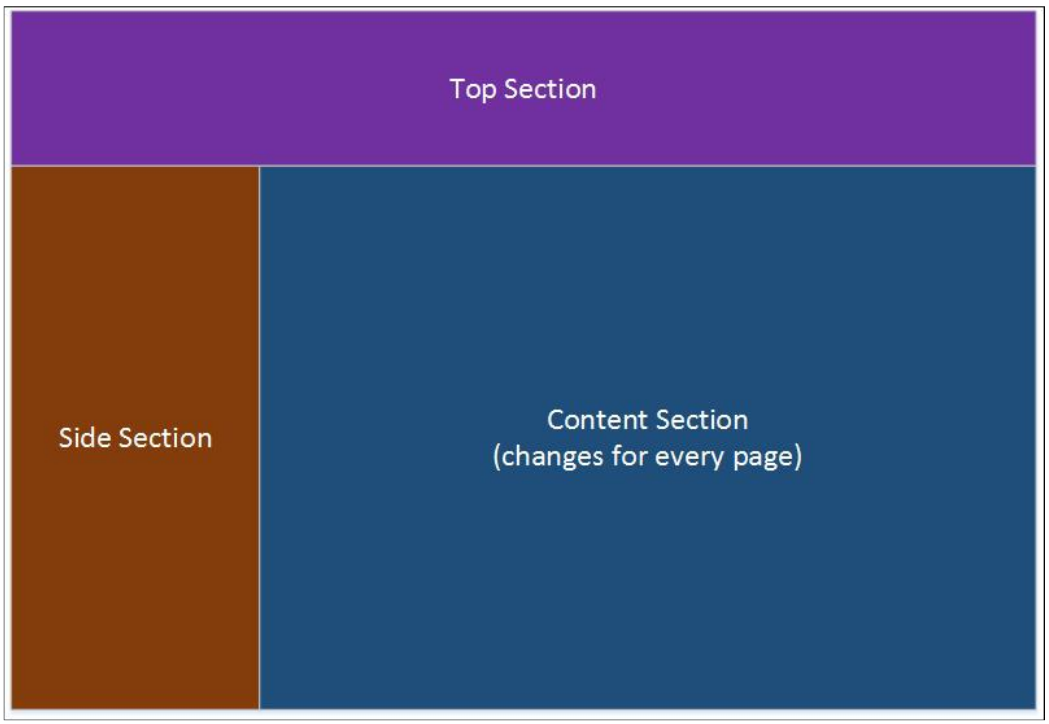
https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-2.1

Para el que disponga del tiempo suficiente, la recomendación es la lectura completa de este documento. Como la sintaxis de Razor no es realmente complicada, y hemos visto cómo utilizar gran parte de las construcciones en los ejercicios de anteriores unidades, no vamos a hacer ningún ejemplo en este apartado. Para cualquier duda de Razor, consultad el enlace anterior.

Layout.

En los ejemplos y ejercicios que hemos realizado hasta ahora, todas nuestras vistas constaban de un único fichero que contenía todo el código. Para escenarios muy simples podemos utilizar esta aproximación, pero trabajar de este modo produce una falta de flexibilidad y reduce la reutilización de nuestro código.

Imaginemos una estructura como la siguiente, en la que tenemos una sección superior que contiene por ejemplo el logo y el nombre de nuestra compañía y una sección lateral con un menú o enlaces a diversas partes de nuestra aplicación. La parte principal de la aplicación sería la sección de contenido, que cambiaría en cada página.



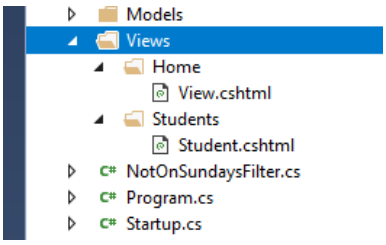
Si reproducimos esta estructura completa en una única página, tendríamos que duplicar el código de las secciones superior y lateral en todas nuestras vistas. Y lo peor, si necesitamos un cambio en una de las secciones, tendríamos que cambiarlo en todos los ficheros.

Para solucionar este y otros problemas utilizaremos un fichero que especifique el layout o estructura general de nuestras páginas, de modo que pueda ser reutilizado por todas ellas. El fichero de layout contiene una estructura simple HTML, que contendrá el contenido común, y definiendo un espacio en el que cada vista individual incluirá su contenido específico.

Ejemplo: Nuestro primer Layout.

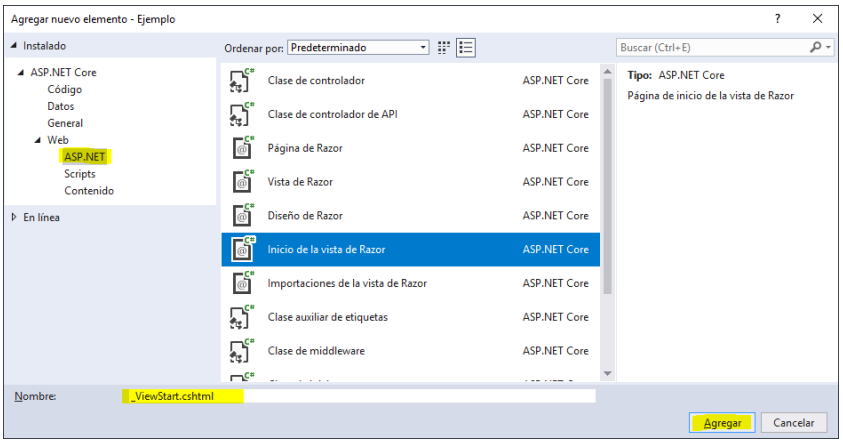
Una vez más, vamos a recurrir a la práctica mediante un ejemplo para ilustrar el proceso de creación y uso de un fichero de layout, además de explicar sobre la marcha las diversas convenciones utilizadas en ASP.NET MVC. Es muy importante no limitarse al estudio de esta documentación e intentar realizar todos los ejemplos uno mismo.

Si creamos un proyecto nuevo con la plantilla aplicación web MVC, ya tendremos creado nuestro layout, que será utilizado por defecto en nuestras vistas. Esta es la aproximación habitual y recomendada, así que será lo que hagamos a partir de ahora. Como en nuestro caso nos interesa conocer los detalles, comenzaremos en un proyecto que se haya creado con una plantilla vacía. Aprovecharemos nuestro viejo proyecto Ejemplo, utilizado en los apartados anteriores, por lo que tendríamos un proyecto con un par de vistas, para los controladores *Home* y *Students*.

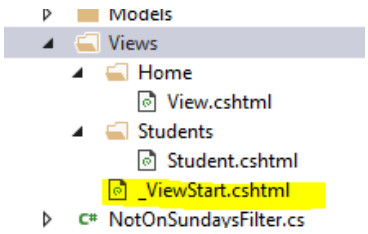


Para crear y utilizar un layout, seguiremos los siguientes pasos:

1. Creación de un fichero **_ViewStart.cshtml**. Este es un fichero especial que nos permite definir código común que será ejecutado al comienzo de todas nuestras vistas. Gracias e este fichero podemos definir comportamiento común a las páginas en un único lugar, lo que simplifica el código y facilita el trabajo al evitarnos repetir una y otra vez el mismo código. Si queremos aplicar un layout determinado a todas nuestras páginas, este es por lo tanto el lugar ideal donde especificarlo.
   Para crearlo hacemos click derecho en nuestra carpeta vistas y seleccionamos **Agregar | Nuevo elemento…**:



En la ventana de selección hemos escogido un "Inicio de la vista de Razor", que es una traducción bastante desafortunada, y en realidad sería una vista de inicio de Razor. Podemos ver que el nombre por defecto es *_ViewStart.cshtml*, así que lo dejamos así.
Por convención, todos los ficheros compartidos entre diferentes vistas comenzarán por guión bajo, y concretamente el fichero de inicio estará situado directamente en la carpeta *Views*, de modo que tendríamos algo como esto:

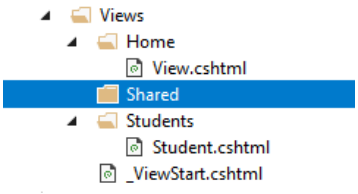


Si abrimos nuestro fichero, podemos ver cómo contiene el código siguiente:

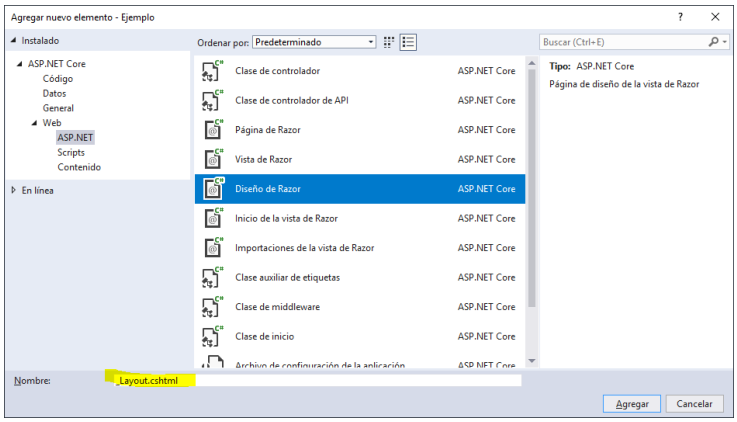
```
@{
    Layout = "_Layout";
}
```

Este código especifica que vamos a utilizar un fichero de layout en nuestra página, al estar contenido en _ViewStart.csthml, estamos haciendo desde un único lugar que todas nuestras vistas utilicen dicho layout.

2. Hemos especificado que nuestras vistas utilicen un fichero de layout, pero ese fichero no existe todavía, así que el siguiente paso será crearlo. En este caso, tenemos una nueva convención que indica que todas las vistas compartidas, así como las vistas parciales, que veremos más adelante, deben estar contenidas en una carpeta de vistas llamada *Shared*, de modo que lo primero es crear dicha carpeta:



Ahora, dentro de esa carpeta agregamos un nuevo elemento, en este caso, un fichero de layout, cuyo nombre en la plantilla es "Diseño de Razor".



CIFP A Carballeira-Marcos Valcárcel
Rúa A Carballeira s/n
Ourense CP 32002
Tfno. 988788470, Fax 988227935

El código que contendrá nuestro layout será el siguiente:

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

Como podemos ver, se trata de una página HTML, que define la estructura que seguirán todas nuestras páginas. En este caso tenemos dos elementos Razor importantes. En el primero vemos que dentro de la etiqueta <title> mostramos el elemento *@ViewBag.Title*, de este modo nos ofrece un mecanismo simple para definir el título de una vista, ya que sólo debemos pasar a la misma un valor para el título en el ViewData / ViewBag (recordamos que es lo mismo) y como todas las vistas utilizan este layout, el título será incrustado en la cabecera automáticamente.
La parte más importante de este fichero es la instrucción *@RenderBody(),* que es una expresión Razor que incrusta el código contenido en una vista dentro del layout, en el lugar que se encuentra dicha expresión. En resumen, cuando abrimos una vista, lo primero que hacemos es cargar el layout, que dota de la estructura general a la misma, y a continuación, en el lugar en que encontramos un *@RenderBody()*, se incluirá el código específico de la vista que queremos mostrar.

3. Vamos a editar nuestro fichero de layout para definir una estructura similar a la indicada, con una cabecera y un lateral:

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>

    <link rel="stylesheet"
href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
/>
</head>
<body>
    <div id="header" style="border-bottom: 2px solid #a63e00; text-
align:center;">
        <h2>Superior</h2>
    </div>

    <div id="row">
        <div id="side" class="col-lg-2">
            <h2>Lateral</h2>
        </div>

        <div class="col-lg-10" style="min-height:650px; margin-top:40px;">
            @RenderBody()
        </div>
    </div>
</body>
</html>
```

La estructura es bastante simple, y hemos aplicado una serie de estilos directamente a las etiquetas en lugar de definir una hoja de estilos. Esto es únicamente para definir las secciones del ejemplo, pero no es el objetivo de este módulo la apariencia de las vistas, si no la generación de contenido del lado servidor. Hemos incluido una referencia a Bootstrap, un framework JavaScript del que hablaremos en unidades posteriores.

4. Crear un fichero de vista que utilice el layout. En este caso, nuestra vista será una vista como cualquier otra, con la única particularidad de que al no tratarse ya de una página completa, sólo debe contener el código específico del contenido que queremos mostrar. No necesitaremos que contenga ninguna etiqueta <html>, <body> o <title>, ya que estas ya están especificadas en el layout.
   En nuestro ejemplo, en lugar de crear una nueva vista, que ya hemos hecho en múltiples ocasiones, vamos a modificar la vista *Index.cshtml* del controlador *Home*, que ya tenemos, para que utilice correctamente el layout.

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    Hola, hemos creado <b>nuestra primera vista!!</b>
</body>
</html>
```

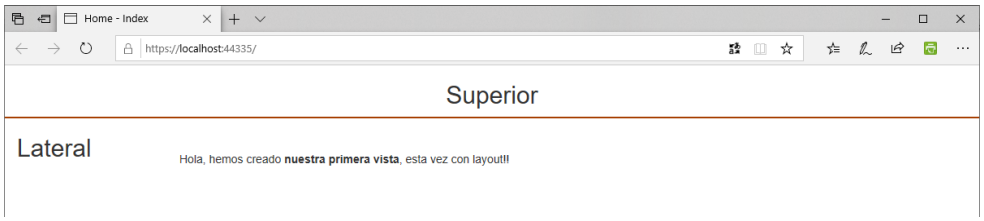
   Vamos a editar el código tal y como se indicó. La primera parte indica que esta página no utiliza layout. Lógicamente ya no queremos este código, y además tampoco necesitamos

   indicar explícitamente que si lo vamos a utilizar, ya que lo hemos hecho en la página de inicio, así que simplemente eliminamos esa parte. tampoco necesitamos nuestras etiquetas html, que ya contiene el layout, si no simplemente el código que es específico de la página. De este modo nuestra página sería simplemente algo así:

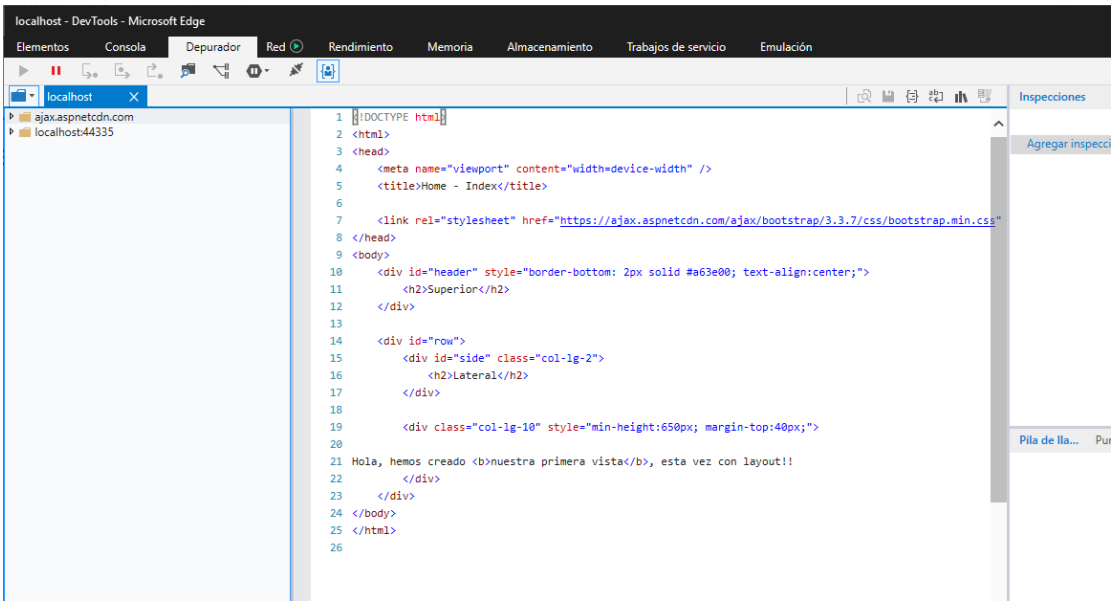
```
@{ViewBag.Title = "Home - Index";}

Hola, hemos creado <b>nuestra primera vista</b>, esta vez con layout!!
```

   Como podemos comprobar, este mecanismo simplifica enormemente la creación de vistas. Simplemente definimos el valor que será utilizado por el layout para mostrar como título y el contenido específico de la vista.

5. Por último ejecutamos nuestro proyecto para ver el resultado.



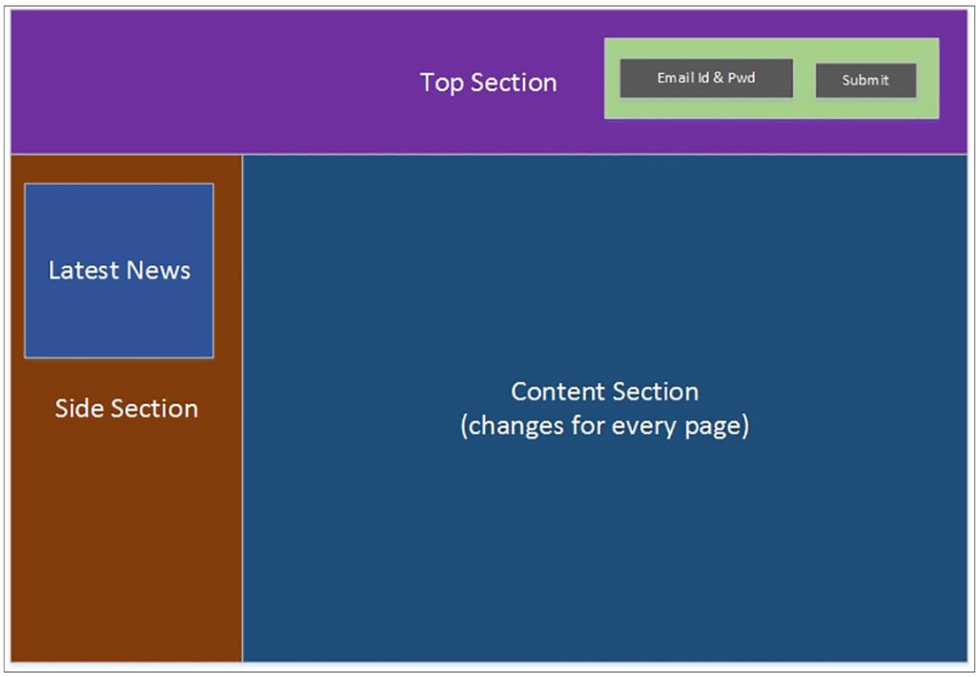
6. Si pulsamos F12 en nuestro navegador podremos ver el código generado y observar cómo se ha incrustado el contenido de la vista en la estructura definida en nuestro layout.



Vistas parciales.

Una vista parcial es una vista que puede ser reutilizada a lo largo de nuestra aplicación. Se puede pensar en ellas como bloques reutilizables que podemos insertar en cualquier lugar en que queramos que nuestro contenido parcial sea mostrado.

Consideremos la estructura de nuestro ejemplo anterior e imaginemos que queremos mostrar, por ejemplo un bloque de noticias en el lateral y otro de login en la cabecera o bloque superior. Para esto podríamos definir vistas parciales que serían incluidas en dichas secciones.



Las vistas parciales no están restringidas en su uso al layout, de modo que podemos utilizar también vistas parciales desde nuestras vistas. Además no tienen por qué contener únicamente contenido estático. En nuestro ejemplo, por ejemplo, la sección de noticias recibiría la lista de noticias de un controlador para generarla de forma dinámica, mientras que la vista de login incluiría un formulario para enviar los datos del usuario a un controlador.

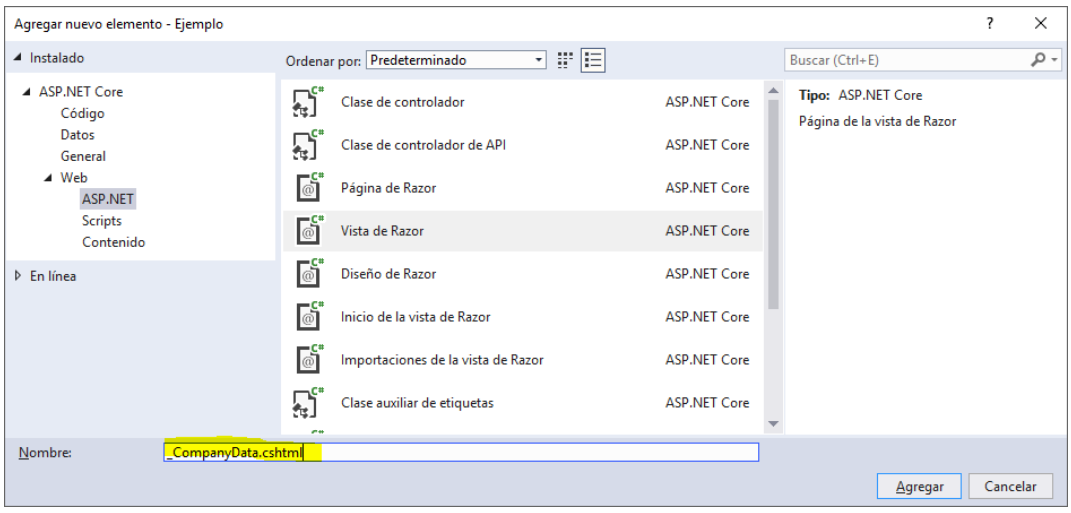
Por convención, el nombre de las vistas parciales debe comenzar por guión bajo, al ser vistas compartidas en diversas partes de la aplicación, pero está permitido que estas sean guardadas o bien en la carpeta *Shared*, o en la carpeta del controlador que hace uso de las mismas. La norma indica que si una vista parcial es utilizada únicamente por un controlador, la guardaríamos en la carpeta de dicho controlador, mientras que en caso contrario lo haremos en la carpeta *Shared*.

En la práctica, su uso es cada vez menor, debido a la aparición de un nuevo elemento llamado *ViewComponent* que ofrece una serie de ventajas sobre las vistas parciales. Pero hay casos en los que su uso sigue siendo recomendable, debido a que son más sencillas de crear que un *ViewComponent*, por lo que son preferibles para mostrar contenido estático, ya que no necesitaremos crear un componente que genere el contenido, y algunos desarrolladores prefieren utilizarlas para formularios simples como logins, etc..

Ejemplo: Creación y uso de una vista parcial.

Para nuestro ejemplo crearemos una vista parcial que contenga el nombre de nuestra empresa y el logo. Puede parecer que crear una vista parcial con contenido estático no es una gran ventaja, ya que podríamos simplemente introducir ese contenido directamente el layout. Pero si lo pensamos con detenimiento, estamos favoreciendo la reutilización, ya que podríamos tener varias aplicaciones web con el mismo layout, y en los que únicamente cambiara el contenido de las partes. De esta forma reutilizaríamos el layout y sólo cambiaríamos el nombre de empresa y logo en la vista parcial.

1. Empezamos añadiendo la nueva vista en la carpeta Shared. Como siempre seleccionamos **Agregar** | **Nuevo elemento…**



En este caso no existe un elemento específico llamado vista parcial, porque en realidad una vista parcial no es distinta a otra vista, salve en su uso. Por lo tanto escogemos vista de Razor y recordamos que el nombre debe comenzar por guión bajo.

2. Con la vista creada, editaremos el código para que muestre la información que queremos.

```
<h2>DWES - DAW Distancia<img src="~/images/logo.png" /></h2>
```

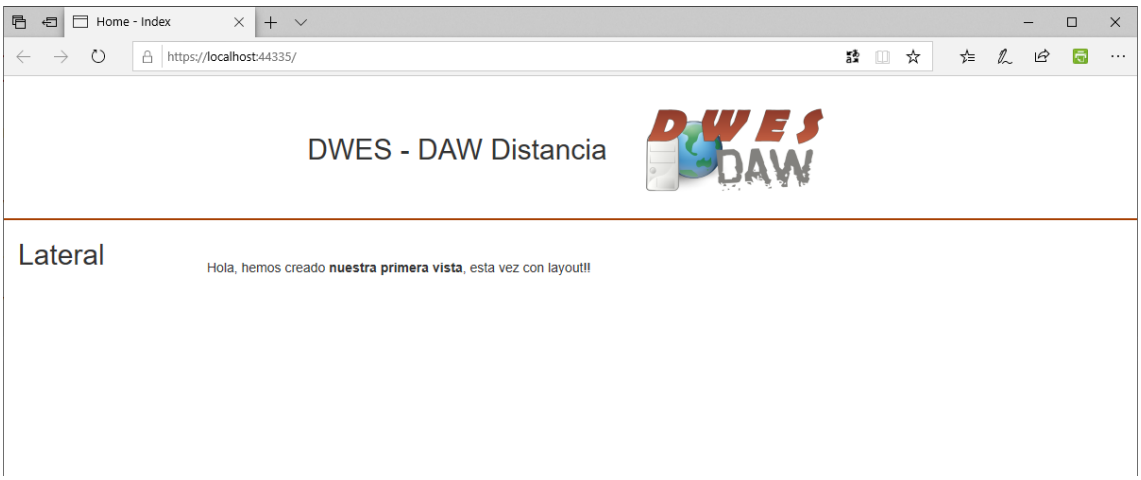
Hemos añadido el nombre de nuestro módulo y el ciclo, y a continuación el logo. Para que se muestre, lógicamente tendremos que añadirlo en la carpeta de contenido estático *wwwroot*, y como podemos ver en la ruta, en mi caso dentro de una carpeta images.

3. Nuestra vista parcial ya está creada, pero nos quedaría llamar a dicha vista para indicarle que se muestre donde queremos. En nuestro caso esto sería en la cabecera definida en el layout, por lo que cambiamos el contenido de esa parte y en su lugar escribimos el siguiente código:

```
<div id="header" style="border-bottom: 2px solid #a63e00; text-align:center;">
    @Html.Partial("_CompanyData")
</div>
```

Para insertar el contenido de una vista parcial utilizamos *@Html.Partial.* Las instrucciones Razor que comienzan por *@Html* se denominan Html Helpers, y facilitan la generación de código HTML en nuestra vista. Más adelante hablaremos de ellos, pero de momento nos quedamos con que este en concreto permite incluir el contenido de una vista parcial en cualquier parte de otra vista. Como argumento pasaremos el nombre de la vista parcial, sin la extensión.

4. Por último ejecutamos el proyecto para comprobar el resultado.



CIFP A Carballeira-Marcos Valcárcel
Rúa A Carballeira s/n
Ourense CP 32002
Tfno. 988788470, Fax 988227935

Componentes de vista (ViewComponent).

Los componentes de vista o ViewComponent son una nueva característica introducida en ASP.NET Core. Son similares a las vistas parciales, pero son más potentes y flexibles.

Cuando se usan vistas parciales, éstas dependen del controlador. Los ViewComponent, por el contrario, no dependen de ningún controlador. Esto ayuda a mantener la separación de tareas y simplificar la aplicación de tests sobre nuestro código. Aunque el uso de vistas parciales sigue estando soportado en ASP.NET MVC, el preferible el uso de un componente de vista en su lugar.

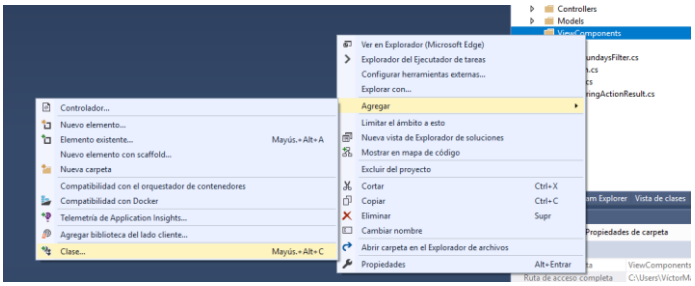
Un ViewComponent se compone de dos partes, una clase que puede heredar de la clase base ViewComponent o ser decorada con el atributo [ViewComponent], y el resultado que se mostrará, que será habitualmente un fichero de vista.

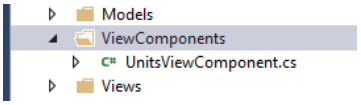
Ejemplo: Creación y uso de un ViewComponent.

En este ejemplo veremos cómo crear un ViewComponent, que muestre la lista de unidades del módulo, y lo utilizaremos en el panel lateral.

1. Comenzamos por crear una carpeta *ViewComponents*, donde alojar nuestros componentes.

2. A continuación, creamos una nueva clase en dicha carpeta. El nombre de esta clase terminará en *ViewComponent* por convención.





3. Editaremos nuestra clase para que herede de la clase base ViewComponent. Toda clase que funcione como ViewComponent debe implementar un método InvokeAsync(), como se muestra a continuación:

```
public async Task<IViewComponentResult> InvokeAsync() {}
```

4. Ahora añadimos el código que devuelva una lista de cadenas de texto que contenga las unidades, con lo que finalmente la clase quedaría como se muestra a continuación:

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

namespace Ejemplo.ViewComponents
{
    public class UnitsViewComponent : ViewComponent
    {
        public async Task<IViewComponentResult> InvokeAsync()
```

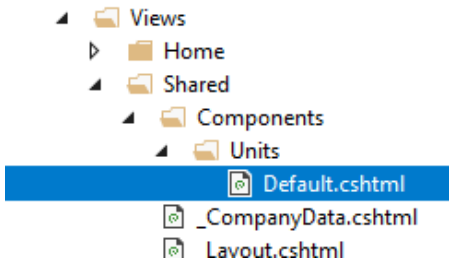
```
        {
            var units = new List<string> {
                "UD 1. Plataformas de programación web en entorno servidor.",
                "UD 2. ASP.NET Core. Características. El lenguaje C#.",
                "UD 3. ASP.NET Core MVC.",
                "UD 4. Formularios y enlace de datos.",
                "UD 5. Acceso a datos con ASP.NET Core. ",
                "UD 6. Desarrollo de aplicaciones web con ASP.NET Core.",
                "UD 7. Servicios web.",
                "UD 8. Aplicaciones web dinámicas. SPA´s.",
                "UD 9. Aplicaciones web híbridas."
            };

            return View(units);
        }
    }
}
```

Se puede observar que el funcionamiento es muy similar al de un controlador, primero obtenemos los datos y una vez hecho devolvemos una vista a la que hemos pasado dichos datos para ser mostrados.

5. Vamos ahora a crear la vista para nuestro componente. Los ficheros de vista serán buscado por el sistema en la siguiente carpeta:

   Views\Shared\Components\<nombre_view_component>

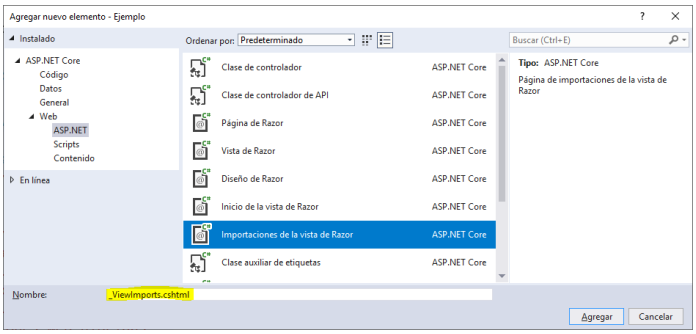
   El nombre por defecto para la vista del componente es *Default.cshtml*, por lo que crearemos las carpetas necesarias y en su interior un fichero de vista de nombre *Default.cshtml*.



6. Ahora editaremos la vista para que muestre los elementos recibidos en una lista. El mecanismo de paso de datos es el mismo que para las vistas de los controladores, a través del objeto model. El código de la vista quedará como se muestra a continuación:

```
@model List<string>

<h3>Unidades</h3>
<ul>
@foreach (var unit in Model)
{
    <li>@unit</li>
}
```

7. Este paso es opcional pero muy recomendable, y además nos servirá para ilustrar el uso de otro fichero común en un proyecto ASP.NET MVC. Desde la carpeta vistas haremos **Agregar | Nuevo elemento…**, y en este caso seleccionaremos un elemento “Importaciones de la vista de Razor”, dejando el nombre por defecto.



8. Este fichero se utiliza para simplificar el desarrollo de las vistas. En una vista Razor podemos incluir código C# genérico, incluyendo el uso de diferentes objetos definidos en clases contenidas en espacios de nombre diferentes. Como en cualquier programa C#, para utilizar una clase que pertenece a otro espacio de nombres, tendremos que incluir una sentencia *using*. El objetivo del fichero *_ViewImports.cshtml* es especificar las sentencias using e importaciones de elementos que queramos utilizar en nuestras vistas, de forma que podamos hacerlo en un único lugar y una única vez, en lugar de hacerlo en cada vista individual. De este modo podremos utilizar en todas nuestras vistas, todas las clases de los espacios de nombres importadas en este fichero. En nuestro caso incluiremos el espacio de nombres en el que se encuentra nuestro *ViewComponent*.

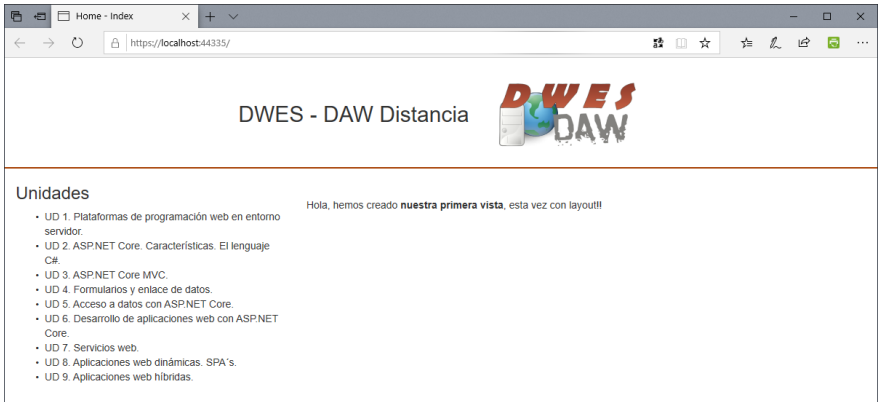
```
@using Ejemplo.ViewComponents
```

9. En estos momentos ya tenemos nuestro ViewComponent listo, y hemos añadido la sentencia using necesaria para usarlo desde cualquier vista en *_ViewImports.cshtml*. Así que sólo nos queda utilizar nuestro componente. Tal y como acordamos, queremos que se muestre en la barra lateral, así que editaremos el código de nuestro fichero layout. Para utilizar un ViewComponent, llamaremos al método InvokeAsync() del objeto genérico Component, y pasaremos como parámetro el nombre del componente a mostrar. El código resultante sería el siguiente:

```
<div id="side" class="col-lg-2">
    @await Component.InvokeAsync("Units")
</div>
```

La instrucción *await* se utiliza para realizar una llamada asíncrona y esperar el resultado. De esta forma, al mostrar nuestra vista, se llamará al componente, que obtendrá sus datos en segundo plano, y una vez los datos estén disponibles, estos se incrustarán en el HTML de la vista.

10. Finalmente ejecutamos el proyecto y comprobamos el resultado.



Generación de código HTML.

Como ya hemos dicho en multitud de ocasiones, los navegadores solo son capaces de interpretar el código HTML, CSS y JavaScript, independientemente de la tecnología utilizada para desarrollar nuestra aplicación web.

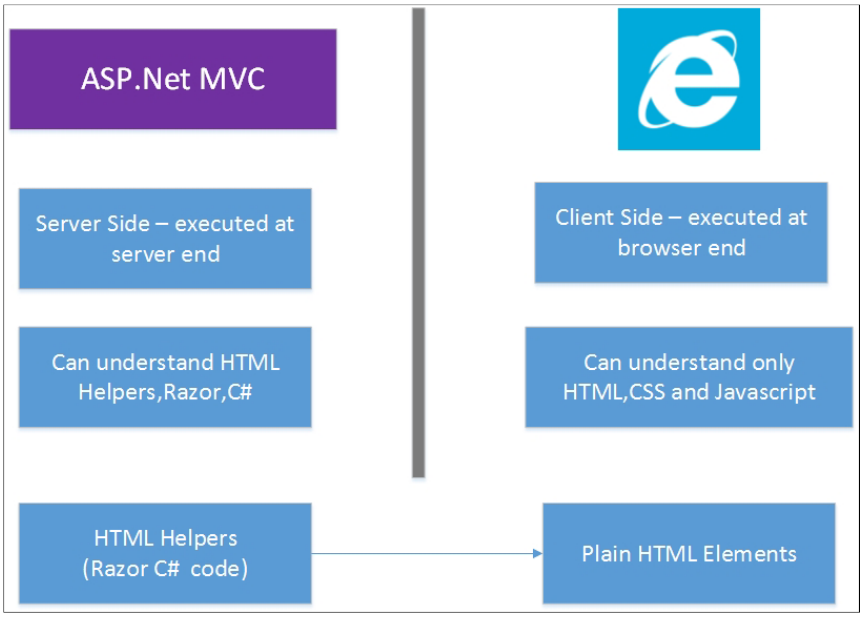
ASP.NET Core MVC proporciona dos mecanismos que nos ayudan a generar código HTML en nuestras vistas, estos son los HTML Helpers y los Tag Helpers.

HTML Helpers.

Los HTML Helpers son métodos de lado servidor que generan código HTML. Estos métodos permiten simplificar la generación de código HTML en determinados casos. Fueron el principal mecanismo de ayuda para la generación de elementos HTML en las vistas hasta la versión ASP.NET MVC 5.

Anteriormente hemos visto como incrustar el código contenido en una vista parcial dentro de otra vista mediante el helper Html.Partial(). Este método, como todos los métodos de lado servidor se ejecuta en nuestro servidor antes de ser devuelto al navegador del cliente. A partir del nombre de la vista parcial compone el resultado final de la vista y lo devuelve como resultado.

A continuación se muestra un detalle de la tarea de los HTML Helpers:



Todos los HTML Helpers están definidos como métodos dentro de la clase Html. Hay una gran cantidad de helpers definidos, y se pueden usar en diferentes tareas, como generación de enlaces, y otros muchos. En cualquier caso, su uso más importante es el trabajo con formularios, por lo que los veremos en más detalle en la siguiente unidad.

Tag Helpers.

Los Tag Helpers son una nueva característica introducida en ASP.NET Core. Con los HTML Helpers, escribíamos código C#/Razor para que generase HTML. La gran desventaja de esta aproximación es que los desarrolladores de frontend o los diseñadores tendrían que conocer el código C#/Razor, cuando están habituados a trabajar con HTML, CSS y JavaScript. Los Tag Helpers tratan de evitar este problema ofreciendo capacidades de generación de código pero utilizando la misma sintaxis que cualquier elemento HTML, utilizando atributos dentro de las etiquetas HTML, de modo que los diseñadores que trabajan con las vistas no tienen por que preocuparse de elementos de código Razor que no conocen, simplemente tendrán que diseñar su código HTML y añadir atributos adicionales a los mismos.

Se recomienda el uso de Tag Helpers en lugar de los equivalentes HTML Helpers siempre que sea posible.

En el siguiente enlace podremos acceder a toda la documentación relativa a los Tag Helpers:

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-2.1

Tag Helpers vs HTML Helpers.

Vamos a mostrar un ejemplo en el que compararemos el uso de un HTML Helper con un Tag Helper equivalente, para ilustrar cómo el último se integra mucho mejor dentro del código HTML.

Imaginemos que desde la vista de la acción Index de un controlador Home, queremos añadir un enlace a otra acción diferente del mismo método, por ejemplo Index2, de modo que permitamos la navegación entre acciones dentro de nuestra acción.

Vamos a generar ese enlace primero utilizando un HTML Helper. El método recibe dos parámetros, el primero será el texto a mostrar dentro del enlace y el segundo sería el nombre de la acción. Si no especificamos un tercer parámetro con el nombre del controlador asumirá que la acción pertenece al controlador actual y será el que utilice para la ruta del enlace.

En nuestro caso el código utilizando el HTML Helper sería como sigue:

```
@Html.ActionLink("Enlace a Index2", "Index2")
```

Y el código HTML generado:

```
<a href="/Home/Index2">Enlace a Index2</a>
```

Para obtener el mismo código utilizando un Tag Helper haríamos lo siguiente:

```
<a asp-action="Index2">Enlace a Index2</a>
```

Otro ejemplo podría ser el uso de un Tag Helper para mostar una vista parcial. Podríamos sustituir el HTML Helper de nuestra cabecera por el siguiente Tag Helper:

```
<partial name="_CompanyData" />
```