educación IT
CODE YOUR FUTURE

Programación Web .NET Core

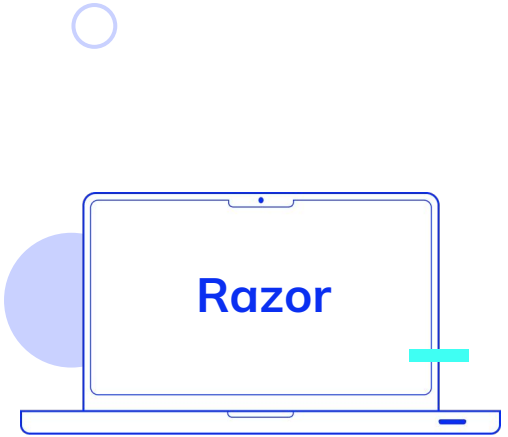
Módulo 6

Creación de vistas con sintaxis de Razor

Creación de vistas con sintaxis de Razor

En una aplicación MVC, **los controladores manejan toda la lógica de la aplicación.** Cada controlador contiene uno o más métodos de acción, y una acción maneja las solicitudes web entrantes.

Las vistas manejan la presentación, no deben contener ninguna lógica de la aplicación y se utilizan para renderizar la respuesta al navegador.



Agregar vistas

- Las **vistas** son la **capa de presentación de la aplicación**.
- Las **vistas** tienen una extensión ***.cshtml***.
- Dentro de un **controlador**, se puede usar el método **`View()`**.

```
public class ProductController: Controller
{
	public IActionResult Index()
	{
	return View();
	}
}
```

En una aplicación MVC, generalmente, **hay un controlador para cada clase de modelo**. Por ejemplo, una clase **`modeloProducto`** por lo general tiene un controlador cuyo nombre es **`ProductController`** que contiene todas las acciones relevantes para *productos.*

Pueden existir algunos **controladores que no correspondan a ninguna clase del modelo.** Sin embargo, **cada controlador puede tener varias vistas**.

Por ejemplo, es posible que desees crear las siguientes vistas para los productos:

- **`Details`:** Puede mostrar un producto, su precio, número de catálogo y otros detalles.
- **`Create`:** Puede permitir a los usuarios agregar un nuevo producto al catálogo.
- **`Edit`:** Puede permitir a los usuarios modificar las propiedades de un producto existente.
- **`Delete`:** Puede permitir a los usuarios eliminar un producto del catálogo.
- **`Index`:** Puede mostrar todos los objetos de producto del catálogo.

Por convención, **todas las vistas en una aplicación MVC residen dentro de la carpeta *Views* del nivel superior.** Dentro de ella, hay una carpeta para cada **controlador** que se utiliza en la aplicación.

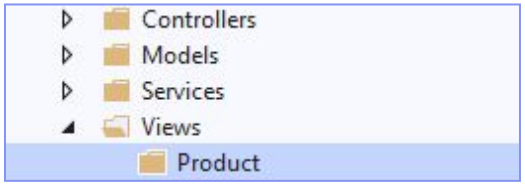
En el ejemplo anterior, ***Views*** contendría una carpeta ***Producto*** y esta carpeta contendría las vistas **`Details`**, **`Create`**, **`Edit`**, **`Delete`** e **`Index`**.

Si utilizas el motor de vista de **Razor** y el lenguaje **C#**, las vistas se crean con una extensión ***.cshtml***.

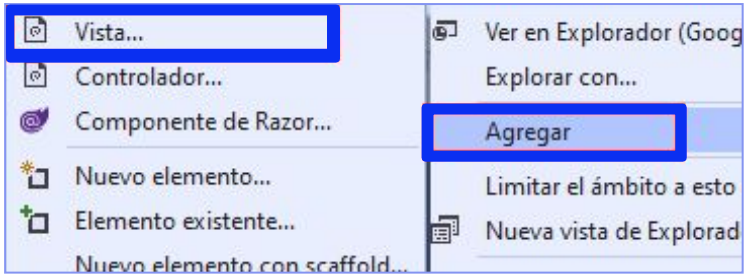
Cómo crear un archivo de vista

Puedes crear un archivo de vista en **Microsoft Visual Studio** siguiendo estos pasos:

1. En el ***Explorador de soluciones***, selecciona la carpeta donde crearás la **vista**. Por ejemplo, para un controlador cuyo nombre es **`ProductController`**, **MVC** espera que las vistas se encuentren en la carpeta llamada ***Views/Product***. Si esa carpeta aún no existe, deberás crearla.

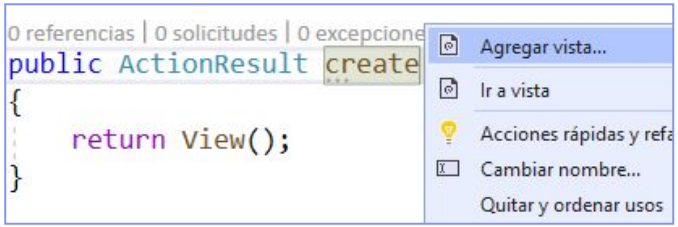


2. Haz clic con el botón derecho en la carpeta seleccionada, elige ***Agregar*** y luego ***Vista***.



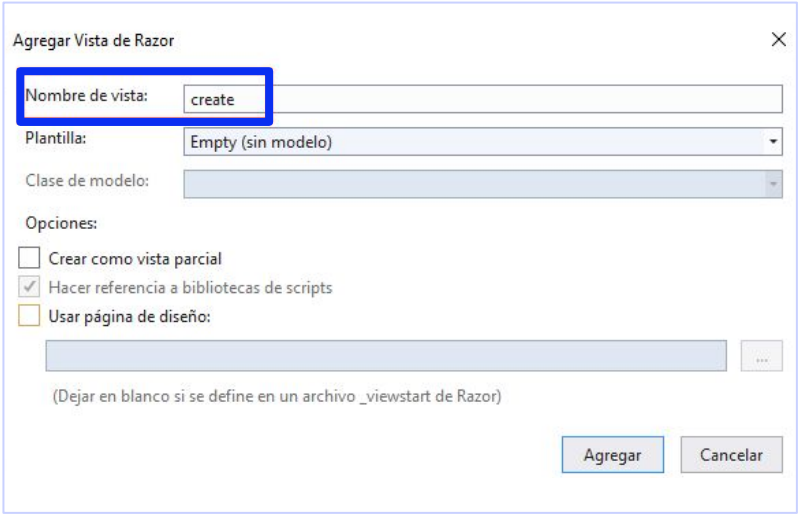
Puedes abrir el controlador y crear una vista para una acción en particular. Haz clic con el botón derecho en la acción y luego en ***Agregar vista***.

1.



2.

3.



- ***Nombre de vistas:*** El nombre de la vista que elijas **debe coincidir con el nombre devuelto por la acción del controlador** correspondiente. Si el método de acción del controlador no especifica el nombre de la vista a usar, MVC asume que el nombre de la vista coincide con el nombre de la acción del controlador.
- ***Clase de modelo:*** Si creas una vista que sea fuertemente tipada, necesitas especificar un modelo para enlazar a la vista.
- ***Plantilla:*** Si especificas un modelo para la vista, Visual Studio crea plantillas simples para `Crear`, `Editar`, `Detalles`, `Eliminar` y `Listar`. Si no especificas un modelo para la vista, puedes construir una **vista vacía** (sin modelo).
- ***Bibliotecas de scripts de referencia:*** Cuando seleccionas esta casilla de verificación en MVC, las referencias a los archivos de *script* del lado del cliente se incluyen en el proyecto.

  > Nota: En **ASP.NET Core MVC**, esta casilla de verificación no tiene ningún efecto.

- ***Crear como vista parcial:*** Una **vista parcial** es una sección del código de Razor que puedes **reutilizar en múltiples vistas** de tu aplicación.
- ***Usar una página de diseño:*** Es posible usar una **vista de diseño** y se utilizará en muchas vistas dentro de la aplicación web.

Ejemplo de Vista Index:

El siguiente código muestra una clase de **controlador** simple que llama al método **`View`** para crear un objeto **`ViewResult`**:

```
public class ProductController :Controller
{
       [Route(“Product/Index”)]
public IActionResult Index()
{
return View();
       }
}
```

Si un usuario solicita la URL relativa, **`/Product/Index`**, el texto, *“Hola desde la vista de índice”*, se enviará al navegador.

El siguiente código muestra el contenido de un archivo llamado ***Index.cshtml*** que se encuentra en carpeta ***Views/Product***:

```
<!DOCTYPE html>
<html>
<head>
<meta name=“viewport”
content=“width=device-width” />
<title> Index </title>
</head>
<body>
Hola desde la vista de índice
</body>
</html>
```

Vista ***Index.cshtml***.

Diferenciar el código del lado del servidor del HTML

El motor de vista de Razor interpreta la vista y ejecuta el código que contiene, del lado del servidor. Para hacer esto, debe distinguir ese código del contenido HTML. El contenido HTML debe enviarse al navegador sin cambios. **El motor de vista de Razor identifica el código del lado del servidor buscando el símbolo @.**

```
<body>
@for (int i = 0; i < 5; i++)
  {
<span>@i</span>
  }
</body>
```

En el siguiente ejemplo, la vista **Razor** ejecuta la línea que tiene el símbolo @ como código **C#** en el servidor web. El texto ***Resultado: ab*** es enviado al navegador. Se utiliza @ para declarar el código del lado del servidor.

```
Result: @String.Concat("a", "b")
```

Una sección de código marcada con el símbolo @ se conoce como ***expresión de código de Razor***. Al inicio de una expresión de código Razor, va el símbolo @.

Uso de Razor

Es posible escribir bucles **for** con la sintaxis de Razor. Por ejemplo: El siguiente código envía el texto ***01234*** al navegador:

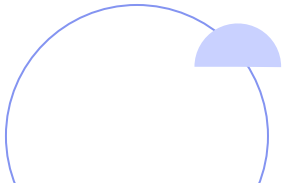
```
@for (int i = 0; i < 5; i++)
{
  @i
}
```

Uso del símbolo @

En la sintaxis de **Razor**, el @ tiene varios usos:

- @ Para identificar el código **C#** del lado del servidor.
- @@ Representar un símbolo @ en una página **HTML**.
- **@:** Para declarar explícitamente una línea de texto como contenido y no como código.

Para declarar explícitamente varias líneas de texto como contenido y no como código se utiliza: **<text>**.



Características de la sintaxis de Razor

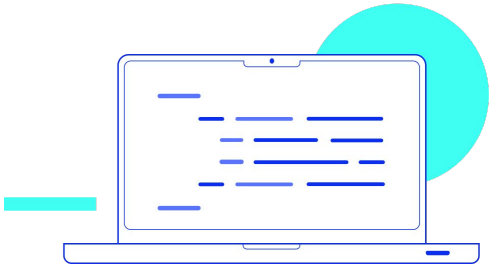
Este bloque de código de ejemplo muestra las características de **Razor**:

```
@*Some more Razor examples*@
<span> Price including Sale Tax: @ViewBag.Price * 1.2 </span>
<span> Price including Sale Tax: @(ViewBag.Price * 1.2)</span>
@{
int i = 5;
int j = 6;
int z = i + j;
@z
}
```

Cómo utilizar la sintaxis de Razor

En esta demostración, aprenderás a:

- Agregar una vista a una aplicación **MVC**.
- Agregar código a la vista usando la sintaxis de **Razor**.
- Usar **Razor** para leer un valor de la propiedad **`ViewBag`** y representarlo en el navegador.



Inyección de dependencias en vistas

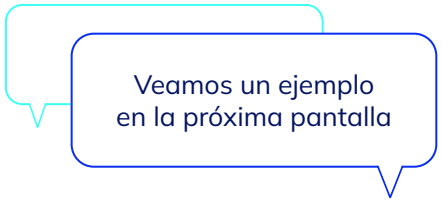
ASP.NET Core admite la inyección de dependencias en las vistas. Se puede inyectar un servicio en una vista usando la directiva **`@inject`**.

```
@inject<tipo><nombre de instancia>
```

El `tipo` es la interfaz del servicio que deseas inyectar. Para encontrar un **`tipo`**, deberás utilizar la ruta para el espacio de nombres de la interfaz.

El **`nombre de la instancia`** es un **alias** que se crea para hacer referencia a un servicio. A lo largo de una **vista**, llamar al alias otorgará acceso a los métodos expuestos por una interfaz.

Después de **inyectar una interfaz**, puedes acceder al servicio en cualquier momento dentro del código contenido por símbolo **@**.



El siguiente código es un ejemplo de un servicio que formatea números para separarlos por comas:

```
Namespace Services
{
public interface IFormatNumber
  {
       string GetFormattedNumber(int number)
  }
public class FormatNumber :IFormatNumber
  {
       public string GetFormattedNumber(int number)
     {
return string.Format(“{0:n0}”, number);
     }
  }
}
```

El siguiente código es un ejemplo de una **vista** que llama al servicio **`FormatNumber`**:

```
@inject Services.IFormatNumber formatNumber

@formatNumber.GetFormattedNumber (1234535334)
```

El ejemplo presenta un servicio simple para formatear números como cadenas. Específicamente, agregará comas a números cada tercer dígito. En el segundo fragmento, el servicio se inyecta en una vista. Si se llama a esta vista, la cadena *1,234,535,334* se mostrará en el navegador.

Se debe tener en cuenta que el espacio de nombres del servicio, coincide con el espacio de nombres en la directiva **`@inject`**.

Práctica recomendada: En general, querrás que los servicios inyectados se coloquen en la parte superior del HTML. Esto es principalmente para mejorar la legibilidad de la vista. El motor Razor puede manejar la inyección al final de una página, pero esto puede generar un código poco claro.

Inyectar servicios en vistas

Servicios para inyectar en vistas:

- Servicios que transforman o dan formato al texto.
- Servicios que impactan en la apariencia visual.
- Servicios que se requieren para usos repetidos dentro de una vista.

Servicios para inyectar en controladores:

- Servicios que recuperan datos.
- Servicios que dependen de recursos externos.
- Servicios que devuelven resultados a una vista.

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!