

# Programación Web .NET Core

Módulo 7

# Trabajar con formularios

# Trabajar con formularios

Cuando creas una **clase de modelo**, defines las **propiedades** y los **métodos** que son apropiados para el tipo del objeto que describe la clase modelo. Basado en las propiedades y métodos de la clase, MVC puede determinar cómo representar el modelo en una página web.

ASP.NET Core MVC Framework incluye muchas funciones en **HTML Helpers** y en los **Tag Helpers** que puedes utilizar en las **vistas**. Puedes utilizar *Helpers* para representar valores, etiquetas y controles de entrada, como cuadros de texto.

## Uso de anotaciones para visualizar y editar datos

Una clase de modelo generalmente contiene varias propiedades. Cada una, generalmente, incluye lo siguiente:

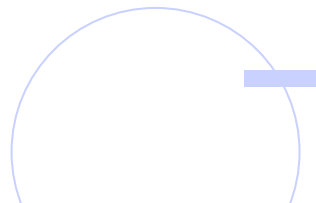
- El **nombre** de la propiedad, como dirección de correo electrónico.
- El **tipo de datos** de la propiedad, como cadena.
- Los **niveles de acceso** de la propiedad, como el obtener y establecer palabras clave para indicar lectura y acceso de escritura.

Además, puedes proporcionar más **metadatos** al describir las propiedades de los modelos en ASP.NET Core MVC. El tiempo de ejecución de ASP.NET Core MVC usa estos metadatos para determinar cómo representar cada propiedad en vistas para su visualización y edición. Estos atributos se denominan de la siguiente manera: ***mostrar y editar anotaciones***.

Por ejemplo, los nombres de propiedad en C# no pueden contener espacios. En una página web renderizada, es posible que desees incluir espacios en una etiqueta de una propiedad. Por ejemplo, es posible que desees renderizar una propiedad llamada `EmailAddress` con la etiqueta *Email Address*. Para proporcionar esta información a MVC, puedes utilizar el atributo **`DisplayAttribute`**.

Es posible que desees proporcionar **información de tipo adicional para una propiedad**.

Por ejemplo, una propiedad contraseña tiene un tipo de datos de cadena. Sin embargo, probablemente sea importante marcar esta propiedad como una contraseña para informar al navegador que necesita mostrar un asterisco o círculo cuando un usuario la escriba. Para proporcionar esta información a MVC, puedes utilizar el atributo **`DataTypeAttribute`**.



El código debajo muestra cómo anotar las propiedades de un modelo con

**DisplayAttribute** y los atributos

**DataTypeAttribute**:

```
public class Person
{
    [Display(Name="MyName")]
    public string Name{ get; set; }
    [DataType(DataType.Password)]
    public string Password{ get; set; }
    [DataType(DataType.Date)]
    public DateTime Birthdate{ get; set; }
    [Display(Name="Email Address")]
    public string EmailAddress{ get; set; }
    [DataType(DataType.MultilineText)]
    public string Description{ get; set; }
}
```

El siguiente código muestra una clase de controlador simple llamada **PersonController**:

```
public class PersonController :Controller
{
    [Route("Person/GetDetails")]
    public IActionResult GetDetails()
    {
        return View();
    }
}
```



El siguiente código ilustra cómo mostrar las propiedades del modelo mediante el HTML `HelperHtml.EditorForModel`:

### La vista `GetDetails`

```
@model ModelNamespace.Person  
@Html.EditorForModel()
```



Si un usuario solicita la URL relativa siguiente: ***/Person/GetDetails***, se muestra un formulario, en el navegador, que contiene los elementos:

- Una etiqueta con el texto ***MyName*** y una caja de texto debajo. Observa que aunque el nombre de la propiedad es **Name**, en el navegador se muestra el texto ***MyName*** porque la propiedad **Name** tiene un atributo **DisplayAttribute**.
- Una etiqueta con el texto ***Password*** y una caja de texto debajo. En el caso de que un usuario ingrese caracteres en la caja de texto, aparecen como asteriscos o círculos en el navegador porque la propiedad **Contraseña** posee un atributo **DataTypeAttribute** con un tipo de datos de contraseña.

- Una etiqueta con el texto ***Birthdate*** y una caja de texto debajo. La caja de texto es un elemento HTML de entrada con un atributo **type="date"** porque la propiedad **Birthdate** tiene un atributo **DataTypeAttribute** con un tipo de datos de fecha.
- Una etiqueta con el texto ***EmailAddress*** y una caja de texto debajo. Observe que aunque el nombre de la propiedad es **EmailAddress**, en el navegador se muestra el texto ***Email Address*** porque la propiedad **EmailAddress** tiene un atributo **DisplayAttribute**.

- Una etiqueta con el texto ***Descripción*** y un área de texto debajo porque la propiedad **Descripción** tiene un atributo denominado **DataTypeAttribute** con un tipo de datos **MultilineText**.

**Nota:** Ten en cuenta que las anotaciones de datos que se mencionan anteriormente se incluyen en el espacio de nombres **System.ComponentModel.DataAnnotations**.



## Usar HTML Helper

Los *HTML Helper* son métodos simples que **normalmente retornan una cadena**. Esta cadena es una pequeña sección de HTML que el motor de visualización puede insertar en la página web. Puedes escribir las vistas con cualquier contenido HTML sin usar un solo Helper si lo prefieres. Sin embargo, los Helpers HTML simplifican la tarea de administrar el contenido HTML al representar HTML común. **MVC incluye varios Helpers que muestran propiedades de clases modelos.**

Puedes utilizar estos Helpers para crear vistas que muestren detalles de productos, detalles de comentarios, detalles de usuarios, y otros.

**Html.DisplayNameFor** representa el nombre de una propiedad de clase modelo.

**Html.DisplayFor** representa el valor de una propiedad de la clase modelo. Ambos *Helpers* examinan la definición de la propiedad en la clase de modelo, incluidos los datos mostrar anotaciones, para asegurarse de que representan el HTML más apropiado.





## Helper *Html.DisplayNameFor*

Puedes utilizar el Helper `Html.DisplayNameFor` para **representar el nombre de una propiedad del modelo**. Si tu vista está fuertemente tipada, Visual Studio verifica si la clase modelo contiene una propiedad con el nombre correcto mientras escribes el código. De lo contrario, debes utilizar una propiedad que exista o verifique que no sea nulo antes de usarlo. Especifica la propiedad de la clase modelo con el Helper `Html.DisplayName` mediante una expresión **lambda**.

### Una clase de modelo

El siguiente código muestra una clase modelo denominada `Persona`:

```
public class Person
{
    public string FirstName{ get; set; }
    public string LastName{ get; set; }
    public bool ContactMe{ get; set; }
}
```

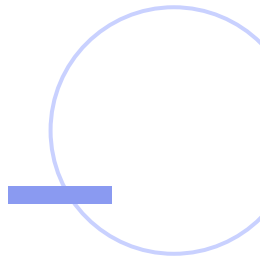



El siguiente código muestra un controlador llamado **PersonController** que crea una instancia del **modelPerson** y lo pasa a una vista llamada **ShowDetails**:

```
public class PersonController :Controller
{
    [Route("Person/ShowDetails")]
    public IActionResult ShowDetails()
    {
        Personmodel = new Person() { FirstName = "James", LastName = "Smith", ContactMe = true };
        returnView(model);
    }
}
```

El siguiente ejemplo de código muestra una vista denominada **ShowDetails** que representa el nombre para mostrar las propiedades del modelo **Person** mediante el Helper **Html.DisplayNameFor**:

```
@model ModelNamespace.Person
<h1>Personproperties</h1>
@Html.DisplayNameFor(model =>model.FirstName)
<br />
@Html.DisplayNameFor(model =>model.LastName)
<br />
@Html.DisplayNameFor(model =>model.ContactMe)
```

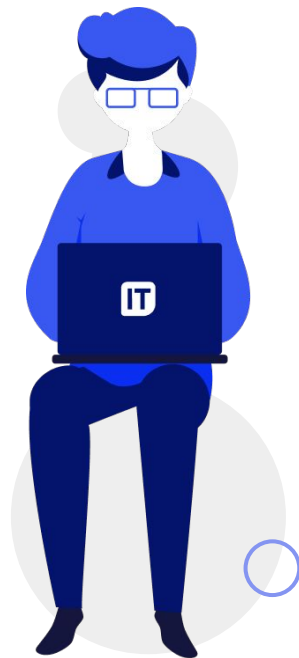


Si un usuario solicita la URL relativa */Persona/ShowDetails*, se mostrará esto en el navegador:

### Pantalla del navegador

```
FirstName  
LastName  
ContactMe
```

El texto representado por el Helper **Html.DisplayNameFor** depende de la clase modelo. Si usaste un atributo **DisplayAttribute** para dar un nombre más descriptivo a una propiedad, **Html.DisplayNameFor** utilizará el valor del parámetro **Nombre**. De lo contrario, mostrará el nombre de la **propiedad**.

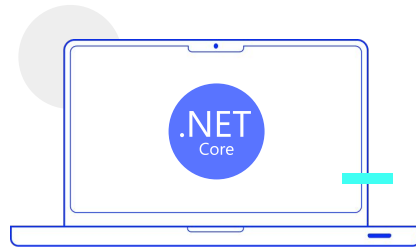


El siguiente código muestra una clase de modelo llamada **Persona** que tiene anotaciones de datos:

```
public class Person
{
    [Display(Name="FirstName: ")]
    public string FirstName{ get; set; }
    [Display(Name="LastName: ")]
    public string LastName{ get; set; }
    [Display(Name="Contact me? ")]
    public bool ContactMe{ get; set; }
}
```

Si un usuario solicita la URL relativa siguiente: **/Person/ShowDetails**, se mostrará en el navegador, lo que vemos a continuación:

```
FirstName:
LastName:
Contact me?
```



## Helper *Html.DisplayFor*

El Helper **Html.DisplayFor** es la **visualización que muestra el valor de la propiedad**. Genera diferentes marcas de HTML según el tipo de datos de la propiedad que se está presentando. Por ejemplo, si la propiedad es de tipo `bool`, representa un elemento de entrada HTML con una casilla de verificación.

Veamos un ejemplo en la siguiente pantalla, el cual muestra una vista que se denomina **ShowDetails**, la cual usa **Html.DisplayFor**.



```
@model ModelNamespace.Person
<h1>Persondetails</h1>
    @Html.DisplayNameFor(model =>model.FirstName)
    @Html.DisplayFor(model =>model.FirstName)
<br />
    @Html.DisplayNameFor(model =>model.LastName)
    @Html.DisplayFor(model =>model.LastName)
<br />
    @Html.DisplayNameFor(model =>model.ContactMe)
    @Html.DisplayFor(model =>model.ContactMe)
```

Si un usuario solicita la URL relativa ***/Person/ShowDetails***, se mostrará en el navegador:

```
FirstName: James
LastName: Smith
Contact me?
```



## Usar Helper Editor

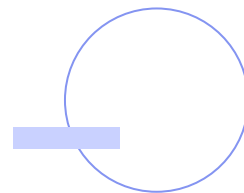
Dentro de los formularios HTML, hay muchos controles HTML que puedes utilizar para recopilar datos de usuarios. En las **vistas de Razor**, los **HTML helpers: `Html.LabelFor` y `Html.EditorFor`** facilitan la renderización de los controles de entrada más apropiados para las propiedades de una clase modelo. Para **comprender cómo estos Helpers renderizan HTML**, primero debes entender los controles de entrada de HTML.





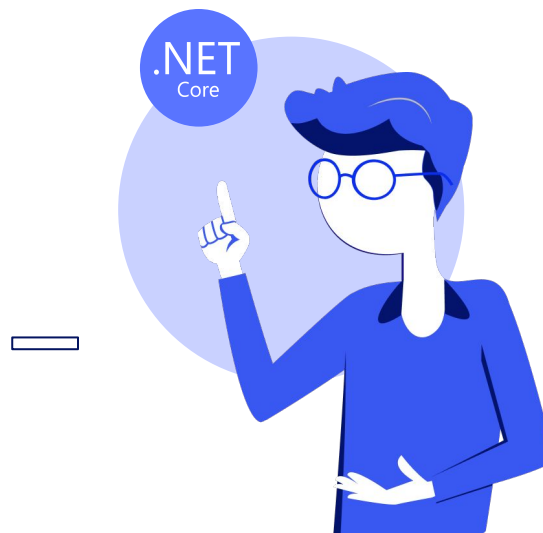
La siguiente tabla describe los controles de entrada de HTML más comunes:

Control	Ejemplo
Text box	<code>&lt;input type="text" name="Title"/&gt;</code>
Multi-linetextbox	<code>&lt;textareaname="Description" rows="20" cols="80" /&gt;</code>
Check box	<code>&lt;input type="checkbox" name="ContactMe" /&gt;</code>



## HTML Helper *Html.LabelFor*

El HTML Helper `Html.LabelFor` es similar al `Html.DisplayNameFor` porque representa el nombre de la propiedad que especifique, considerando el atributo **DisplayAttribute** si está especificado en el modelo. Pero, a diferencia del HTML Helper `Html.DisplayNameFor`, el HTML Helper `Html.LabelFor` representa un elemento `<label>`.



## HTML Helper *Html.EditorFor*

El HTML Helper **Html.EditorFor** representa los **elementos de entrada HTML más apropiados y otros controles de formulario para cada tipo de datos de la propiedad en un modelo.**

Por ejemplo, el HTML Helper **Html.EditorFor** representa `<input type = "text">` para una propiedad del tipo cadena. Si la propiedad cadena está anotada con la decoración `[DataType (DataType.MultilineText)]`, el Helper **Html.EditorFor** representa un elemento `<textarea>`.

La siguiente tabla describe el HTML que **Html.EditorFor** representa para diferentes propiedades del modelo:

Control	ModelClassProperty	HTML RenderedbyEditorFor()
Text box	<code>publicstringTitle{ get; set; }</code>	<code>&lt;input type="text" name="Title" /&gt;</code>
Multi-linetextbox	<code>[DataType(DataType.MultilineText)] publicstringDescription{ get; set;}</code>	<code>&lt;textareaname="Description" rows="20" cols="80" /&gt;</code>
Check box	<code>publicboolContactMe{ get; set; }</code>	<code>&lt;input type="checkbox" name="ContactMe" /&gt;</code>



Si la acción pasa un objeto modelo existente a la vista, el Helper **Html.EditorFor** también completa cada control con los valores actuales de cada propiedad. El siguiente código muestra un modelo **Persona** que tiene anotaciones de datos:

```
public class Person
{
    [Display(Name="FirstName: ")]
    public string FirstName{ get; set; }
    [Display(Name="LastName: ")]
    public string LastName{ get; set; }
    [Display(Name="Contact me? ")]
    public bool ContactMe{ get; set; }
}
```

El siguiente código muestra un controlador llamado **PersonController**:

```
public class PersonController :Controller
{
    [Route("Person/GetDetails")]
    public IActionResultGetDetails()
    {
        return View();
    }
}
```



El siguiente código muestra una vista denominada **GetDetails** que usa el Helper **Html.LabelFor** y el Helper **HTML.Editor**:

```
@model ModelNamespace.Person
<h1>Getpersondetails</h1>
@Html.LabelFor(p =>p.FirstName)
@Html.EditorFor(p =>p.FirstName)
<br />
@Html.LabelFor(p =>p.LastName)
@Html.EditorFor(p =>p.LastName)
<br />
@Html.LabelFor(p =>p.ContactMe)
@Html.EditorFor(p =>p.ContactMe)
```

Si un usuario solicita la URL relativa siguiente:  
**/Person/GetDetails**, se mostrará en el navegador:

```
Getpersondetails
FirstName:
LastName:
Contact me?
```

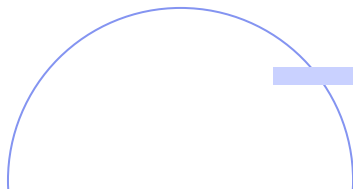


## Tag Helpers *LabelTagHelper* e *InputTagHelper*

Los *Tag Helpers* son una alternativa a los **HTML Helpers**. Al igual que los **Helpers HTML**, la función de los **Tag Helpers** incrusta código del lado del servidor dentro de una vista. Sin embargo, para los programadores y diseñadores es mucho **más fácil comprenderlos porque los Tag Helpers se parecen a elementos HTML normales**.

### *LabelTagHelper*

El Tag Helper **LabelTagHelper** es una alternativa al HTML **Html.LabelFor**. Genera un elemento **<label>** para una propiedad en un modelo y puedes usarlo agregando un atributo **asp-for** a un elemento **<label>**. El valor del atributo HTML generado coincide con el nombre de la propiedad en la clase modelo. El contenido del elemento **<label>** coincide con la propiedad **Name** del atributo **DisplayAttribute** que se especifica en la clase de modelo.



## InputTagHelper

El Tag Helper **InputTagHelper** es una alternativa al HTML Helper **Html.EditorFor**. Genera un elemento **<input>** para una propiedad de un modelo y puedes usarlo agregando un atributo **asp-for** a un elemento **<input>**. El Tag Helper **InputTagHelper** agrega una identificación y un nombre basado en el nombre de la propiedad especificado en el atributo **asp-for**. Similar al HTML Helper **Html.Editor**, el tipo del elemento de entrada se determina en función del tipo .NET de la propiedad de la clase de modelo.

**Por ejemplo:** si el tipo .NET de la propiedad es una cadena, el tipo del elemento de entrada es texto, mientras que si el tipo .NET de la propiedad es **bool**, el tipo de elemento de entrada es **checkbox**.





El siguiente código muestra cómo reescribir la vista **GetDetails** para usar **Tag Helper** en lugar de usar HTML Tag Helper:

```
@addTagHelper *,Microsoft.AspNetCore.Mvc.TagHelpers
@model ModelNamespace.Person
<h1>Get person details</h1>
<label asp-for="FirstName"></label>
<input asp-for="FirstName" />
<br />
<label asp-for="LastName"></label>
<input asp-for="LastName" />
<br />
<label asp-for="ContactMe"></label>
<input asp-for="ContactMe" />
```

Si un usuario solicita la URL relativa **/Person/GetDetails**, se mostrará en el navegador:

```
Getpersondetails
FirstName:
LastName:
Contact me?
```



## Usar Tag Helper de formulario

Para aceptar la entrada del usuario, puedes proporcionar un formulario en tu página web. Una forma típica consta de un **conjunto de etiquetas y controles de entrada**. Las etiquetas indican al usuario la propiedad para la que debe proporcionar un valor. Los controles de entrada permiten al usuario ingresar un valor. Los controles de entrada pueden ser cajas de texto, cajas de verificación, botones de opción, selectores de archivos, listas desplegables u otros tipos de control. Allí suele haber un botón *Enviar* y un botón *Cancelar*.

### HTML helper `Html.BeginForm`

El `Html.BeginForm` se utiliza para crear un formulario en HTML. Debe comenzar con un elemento `<form>` en la página web HTML y todas las etiquetas y los controles de entrada deben estar dentro del elemento `<form>`. En una vista MVC, puedes usar `Html.BeginForm` para representar este elemento y establecer la acción del controlador a la que el formulario envía datos.

También puedes especificar el método HTTP que utiliza el formulario para enviar datos. Si el formulario usa el método **post**, que es el predefinido, el navegador envía los valores al servidor web en el cuerpo del formulario. Si usa el método **get**, el navegador envía los valores al servidor web en la cadena de consulta en la URL.

En el HTML renderizado, el elemento **<form>** debe cerrarse con una etiqueta **</form>**. En las vistas de **Razor**, asegúrate de que el elemento del formulario esté cerrado con un bloque de código Razor **@using**, el cual representa la etiqueta **</form>**.

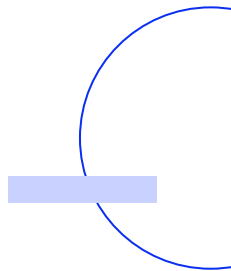
Este código muestra una clase modelo llamada **Persona** que tiene anotaciones de datos:

```
public class Person
{
    [Display(Name="FirstName: ")]
    public string FirstName{ get; set; }
    [Display(Name="LastName: ")]
    public string LastName{ get; set; }
    [Display(Name="Contact me? ")]
    public bool ContactMe{ get; set; }
}
```



El siguiente código muestra un controlador denominado **PersonController**:

```
public class PersonController :Controller
{
    [Route("Person/GetDetails")]
    public IActionResultGetDetails()
    {
        return View();
    }
    [Route("Person/ShowDetails")]
    public IActionResultShowDetails(Personperson)
    {
        return View(person);
    }
}
```



El siguiente ejemplo de código muestra cómo representar un formulario que envía datos a la acción **ShowDetails** en el controlador llamado **PersonController**.

El formulario usa el verbo **HTTP post**:



## Uso del ayudante `Html.BeginForm`

```
@addTagHelper *,Microsoft.AspNetCore.Mvc.TagHelpers
@model ModelNamespace.Person
<h1>Getpersondetails</h1>
@using (Html.BeginForm("ShowDetails", "Person"))
{
    <label asp-for="FirstName"></label>
    <input asp-for="FirstName" />
    <br />
    <label asp-for="LastName"></label>
    <input asp-for="LastName" />
    <br />
    <label asp-for="ContactMe"></label>
    <input asp-for="ContactMe" />
    <br />
    <input type="submit" value="Submit my details" />
}
```

El siguiente ejemplo de código muestra una vista denominada **ShowDetails**:

### La vista ShowDetails

```
@model ModelNamespace.Person
<h1>Person details</h1>
@Html.DisplayNameFor(model =>model.FirstName)
@Html.DisplayFor(model =>model.FirstName)
<br />
@Html.DisplayNameFor(model =>model.LastName)
@Html.DisplayFor(model =>model.LastName)
<br />
@Html.DisplayNameFor(model =>model.ContactMe)
@Html.DisplayFor(model =>model.ContactMe)
```

Si un usuario solicita la URL relativa siguiente: **/Person/GetDetails**, se muestra un **formulario** en el navegador. Si un usuario ingresa su nombre y apellido y hace clic en el botón **Enviar**, se llama a la acción **ShowDetails** del controlador **PersonController**.

**Nota:** el HTML Helper **Html.BeginForm** está sobrecargado. Por ejemplo, para especificar el método HTTP que usa el formulario para enviar datos, ingresar:

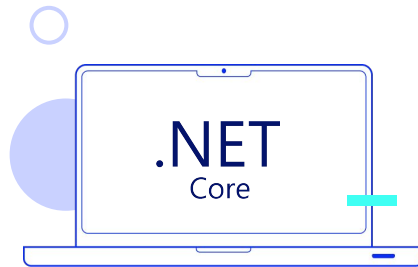
```
Html.BeginForm ("ShowDetails", "Persona",
    FormMethod.Post))
```

## El Tag Helper *FormTagHelper*

El **FormTagHelper** es una alternativa al llamado **Html.BeginForm**. Genera un elemento HTML **<form>**. Puede agregar al **FormTagHelper** un atributo **asp-controller** para enlazar a un controlador específico. También puede agregar al **FormTagHelper** un atributo **asp-action** para enlazar a una acción específica.

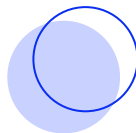
Aunque el valor del método de formulario predeterminado es **post**, puede especificar otro método HTTP en el formulario.

El código de la siguiente pantalla, muestra cómo es posible reescribir la vista **GetDetails** para usar un **FormTagHelper** en lugar de utilizar el **Html.BeginForm**.



## Uso de FormTagHelper

```
@addTagHelper *,Microsoft.AspNetCore.Mvc.TagHelpers
@model ModelNamespace.Person
<h1>Get person details</h1>
<form asp-controller="Person" asp-action="ShowDetails">
  <label asp-for="FirstName"></label>
  <input asp-for="FirstName" />
  <br />
  <label asp-for="LastName"></label>
  <input asp-for="LastName" />
  <br />
  <label asp-for="ContactMe"></label>
  <input asp-for="ContactMe" />
  <br />
  <input type="submit" value="Submit my details" />
</form>
```





**¡Sigamos  
trabajando!**

