

## UD4 Formularios y enlace de datos.

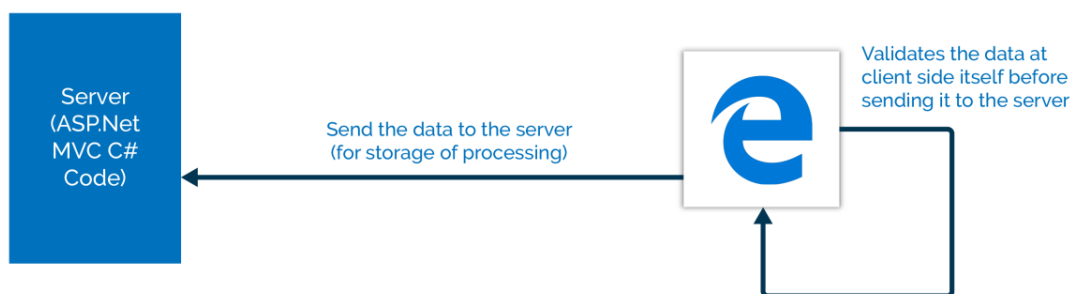
### 4.2 Validación de datos.

Una norma elemental en el desarrollo de aplicaciones dice que nunca debemos confiar en los datos introducidos por los usuarios. En ocasiones podría ocurrir que por desconocimiento de la aplicación o por equivocación un usuario introduzca datos incorrectos. En otros casos, podríamos encontrar usuarios malignos que quieren comprometer nuestra aplicación introduciendo datos inapropiados. En cualquier caso, tendremos que validar siempre la corrección de los datos introducidos en nuestra aplicación cuando son recogidos por el formulario correspondiente, antes de pasar a procesar los mismos.

#### Introducción a la validación.

En un caso ideal, los usuario introducirían siempre datos adecuados y con el formato correcto en nuestra aplicación. Sin embargo, como es de suponer, el mundo real no suele coincidir con el ideal. Como desarrolladores, nunca podemos confiar en que los datos introducidos serán siempre correctos, y es nuestra responsabilidad validar los datos de entrada del usuario. Si éstos no son válidos, informaremos al usuario, indicando el problema, para que pueda corregirlo y enviar los datos de nuevo.

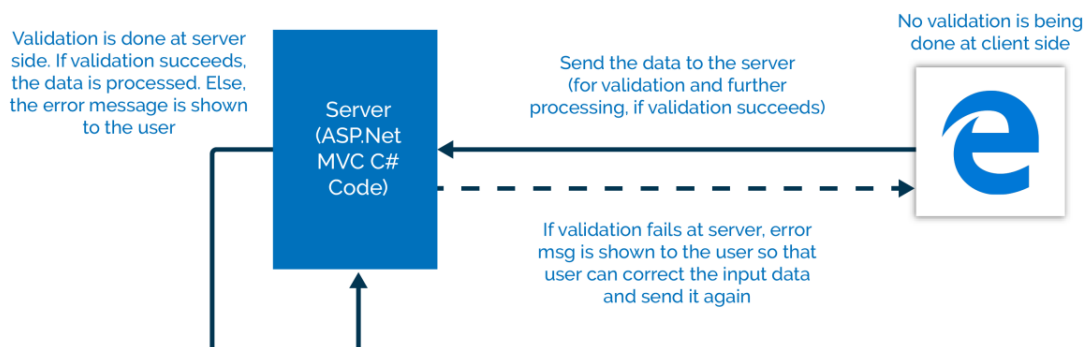
La validación puede tener lugar en el lado cliente, en el lado servidor, o en ambos. Cuando los datos se validan antes de que los datos se envíen al servidor, estamos realizando una validación de lado cliente. Por ejemplo, si un usuario no introduce un dato en un campo requerido, podríamos comprobar esta circunstancia sin necesidad de enviar los datos del formulario al servidor. El lenguaje de programación común para la validación del lado cliente es JavaScript, que puede ser ejecutado directamente en el navegador del usuario.



Cuando la validación se realiza en el servidor, una vez recibidos los datos del formulario, decimos que estamos ante una validación del lado servidor. Un ejemplo podría darse cuando necesitamos realizar algún tipo de comprobación de los datos introducidos por el usuario contra una base de datos, comprobando un nombre de usuario o contraseña, etc..

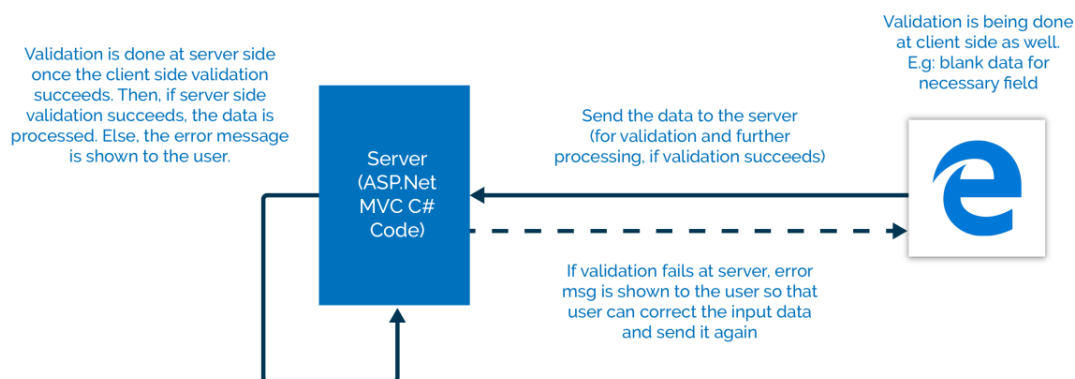
Incluso si no tenemos la necesidad de realizar ninguna comprobación contra una base de datos, podríamos necesitar realizar validación de datos del lado servidor. Un usuario malicioso puede alterar los datos introducidos, incluso el código HTML o JavaScript del navegador y enviar información incorrecta. Podría ocurrir también que un usuario haya desactivado la ejecución de código JavaScript en su navegador, evitando de este modo las validaciones del lado cliente.

Para todos estos casos, la única defensa de la seguridad de nuestra aplicación descansará sobre la validación de lado servidor.



## Validación del lado cliente y del lado servidor.

En una aplicación real, habitualmente no se tratará de decidir si realizamos validación de lado cliente o lado servidor. La validación de lado servidor es importante por razones de seguridad. Por otro lado, la validación de lado cliente es conveniente para el usuario y mejorar el rendimiento de nuestra aplicación, ya que se lleva a cabo directamente en el navegador del usuario, por lo que no tiene impacto en el servidor de la aplicación. Por lo tanto, lo recomendable será que nuestra aplicación implemente mecanismos de validación tanto de lado cliente como servidor.



La imagen anterior muestra un proceso en el que tiene lugar validación en los dos extremos. Si los datos no son introducidos en un campo requerido, podemos detectarlo en el propio navegador, sin necesidad de enviar los datos del formulario al servidor. Una vez que todos los datos obligatorios se han introducido, estos son enviados al servidor, donde se realiza la validación de los mismos en base a la lógica de negocio. Si dicha validación falla, el formulario es enviado de vuelta al navegador del usuario, con un mensaje de error que indique el problema y este pueda ser corregido.

## Validación en el servidor (Server-side validation).

Para llevar a cabo la validación del lado servidor, necesitamos realizar las siguientes acciones:

1. Añadiremos atributos de anotación de datos (Data Annotation) al modelo o ViewModel correspondiente a la vista del formulario. Los datos introducidos serán validados contra esta metainformación automáticamente, facilitando la tarea del desarrollador.
2. Modificaremos la vista para que muestre mensajes de validación para cada uno de los campos. Para ello utilizaremos un tag helper *span*, con el atributo *asp-validation-for* que indique el campo que estamos validando.
3. Editaremos el método de acción del controlador para que verifique el estado del modelo. Si este es válido, podemos proceder a procesar los datos de entrada. En caso contrario, volveremos a mostrar la vista del formulario indicando los errores de validación que se hayan producido, permitiendo que el usuario pueda corregir dichos errores y enviar el formulario una vez más.

## Actualizar ViewModels con atributos de anotaciones (Data Annotation Attribute).

Los atributos Data Annotation definen las reglas de validación para las propiedades del modelo / ViewModel. Si los datos de entrada no concuerdan con la especificación del atributo, la validación fallará, haciendo inválido el estado del modelo. Realmente, la recomendación indica que los modelos no deberían contener atributos para la validación de datos, puesto que los modelos representan entidades de la lógica de negocio, independientes de la tecnología utilizada para la IU y la entrada de datos. Una clase del modelo de negocios no debería contener atributos específicos ASP.NET, ya que iría contra los principios de separación de responsabilidades y disminuye la reutilización del código. La forma más correcta sería crear siempre un ViewModel para cada vista, aún cuando la clase de modelo correspondiente encajara perfectamente.

Para nuestro módulo, por simplicidad comenzaremos por la opción más simple, añadiendo los atributos necesarios al modelo o el ViewModel, según lo que tengamos en cada vista. Más adelante, en una unidad posterior, volveremos a este punto y comenzaremos a seguir la norma, presentando herramientas que nos facilitarán esta tarea.

Existen varias Data Annotation disponibles, que podemos encontrar dentro del namespace *System.ComponentModel.DataAnnotations*. A continuación se enumeran los más comúnmente utilizados:

- **Required:** indica que la propiedad es requerida y por lo tanto su entrada es obligatoria.
- **Range:** define un rango de valores indicando el mínimo y máximo.
- **MinLength:** define la longitud mínima de caracteres de una propiedad.
- **MaxLength:** define la longitud máxima de caracteres de una propiedad.
- **RegularExpression:** permite usar una expresión regular para la validación.

Estas anotaciones permiten especificar un valor para el parámetro *ErrorMessage*, que indica el mensaje que se mostrará si la validación falla. En caso de no especificar ninguno, se mostrará el mensaje de error por defecto.

En el siguiente enlace se puede obtener más información acerca de las Data Annotations:

[https://www.tutorialspoint.com/asp.net\\_mvc/asp.net\\_mvc\\_data\\_annotations.htm](https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_data_annotations.htm)

## Ejemplo: Validación del lado servidor.

Partiremos del ejemplo creado en la unidad 4.1 y seguiremos los pasos necesarios para realizar validación del lado servidor, realizando las acciones indicadas anteriormente en esta unidad:

1. Como paso previo antes de implementar la validación, cambiaremos la ruta por defecto en *Startup* para que al lanzar la aplicación se muestre la página del formulario. Para esto especificamos *Censo* como controlador por defecto, y *Comprobar* como acción.

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Censo}/{action=Comprobar}/{id?}");
});
```

2. Comenzamos añadiendo los Data Annotation necesarios al modelo, en nuestro caso en la clase *Persona*.

```
using System.ComponentModel.DataAnnotations;

namespace Formularios.Models
{
    public class Persona
    {
        [Required(ErrorMessage = "Debe introducir el DNI")]
        [RegularExpression(pattern: "[0-9]{8}[A-Z]",
            ErrorMessage = "El DNI debe ser de la forma 11111111X")]
        public string Dni { get; set; }

        [Required(ErrorMessage = "Debe introducir el nombre")]
        public string Nombre { get; set; }
    }
}
```

Hemos añadido los atributos *Required* a las dos propiedades, indicando que es obligatorio introducir los dos campos en el formulario. Para el DNI hemos añadido también una expresión regular que nos permite definir el formato que debe tener el dato introducido, que sería en este caso una cadena formada por 8 dígitos y una letra en mayúscula.

Las expresiones regulares son un mecanismo muy potente que permite definir complejos patrones que nuestras entradas deben seguir. Para más información acerca de las expresiones regulares se puede consultar el siguiente enlace:

<https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference>

3. A continuación, en la vista, para cada uno de los campos del formulario añadimos una etiqueta *span* que muestre el error de validación cuando este se produzca. Definiremos el estilo de la etiqueta para que el mensaje de error se muestre en color rojo. El atributo *asp-validation-for* representa el nombre de la propiedad para la que se mostrará el mensaje de error de validación.

```
@model Persona
@{
    ViewData["Title"] = "Comprobar datos de censo";
}

<form asp-controller="censo" asp-action="comprobar" method="post">
    <table>
        <tr>
            <td>
                <label asp-for="Nombre">Nombre</label>
            </td>
            <td>
                <input asp-for="Nombre" />
            </td>
            <td><span asp-validation-for="Nombre" style="color:red"></span></td>
        </tr>
        <tr>
            <td>
                <label asp-for="Dni">DNI</label>
            </td>
            <td>
                <input asp-for="Dni" />
            </td>
            <td><span asp-validation-for="Dni" style="color:red"></span></td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" />
            </td>
        </tr>
    </table>
</form>
```

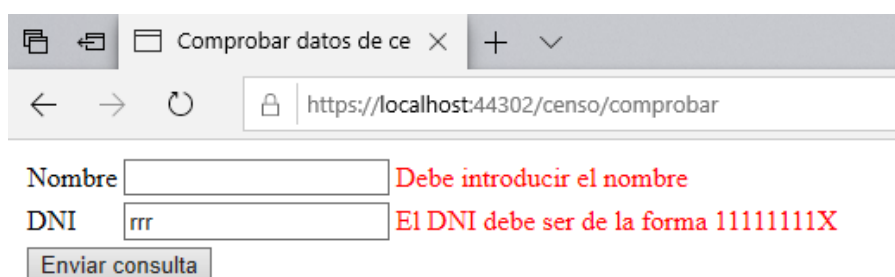
4. Lo último que necesitamos es añadir el código necesario en el controlador para llevar a cabo la validación. Para ello utilizaremos la propiedad *IsValid* del objeto *ModelState*, que se encarga de comprobar que los datos introducidos en el formulario cumplen los requisitos indicados en las Data Annotations. En caso correcto la validación es pasada con éxito y avanzamos a la vista que muestra el resumen de los datos, mientras que si no es así volvemos a mostrar la misma vista, que contiene el formulario, y que de forma automática mostrará los errores de validación detectados por el *ModelState*.

```
[HttpPost]
public ActionResult Comprobar(Persona persona)
{
    if(ModelState.IsValid)
    {
        return RedirectToAction("informacion", persona);
    }

    return View(persona);
}
```

Este código tan simple desempeña una cantidad de trabajo enorme, siendo capaz de comparar los datos de cada propiedad del objeto recibido, en este caso una Persona, validando cada uno de ellos según las anotaciones encontradas. De este modo se libera al desarrollador de tener que realizar una tarea ardua y tediosa.

5. Finalmente, ejecutamos la aplicación y comprobamos que la validación de datos funciona como esperamos, indicando los errores correspondientes en caso de no introducir los datos como se requiere.



## Validación de lado cliente (Client-Side Validation).

Existen escenarios en los que no necesitamos recurrir al servidor para validar los datos de entrada. En el ejemplo que acabamos de realizar, sin ir más lejos, no habría necesidad de hacer una petición al servidor para asegurarnos de que se ha introducido un nombre en el campo correspondiente. Este tipo de validaciones pueden realizarse del lado del cliente, directamente en el navegador, lo que agiliza la aplicación y reduce la carga del servidor.

Para realizar estas validaciones se utilizan funciones JavaScript que se ejecutan en el navegador del usuario. Podríamos definir nuestras funciones para ello aunque el mecanismo recomendado en ASP.NET MVC es utilizar la llamada unobtrusive validation, usando la librería jQuery Validation.

Esta librería utiliza una serie de atributos que se añaden a las etiquetas HTML del formulario, a partir de los cuales la librería de validación es capaz de realizar todo el trabajo en el navegador. Estas etiquetas tienen la forma data-, y todos los campos que contengan dichos campos serán validados automáticamente.

La gran ventaja de utilizar este mecanismo es que, como veremos a continuación mediante un ejemplo, ASP.NET MVC añade por nosotros las etiquetas a partir de las Data Annotations del ViewModel, de forma que las mismas anotaciones nos van a servir para la validación en los dos extremos sin necesidad de realizar trabajo extra. Este mecanismo, de nuevo, nos permite liberarnos de tener que añadir una gran cantidad de código, facilitando nuestro trabajo.

## Ejemplo: Realizar validación de lado cliente.

En este último ejemplo añadiremos validación de lado cliente a nuestro formulario.

1. Antes de modificar nada en nuestro proyecto, comenzaremos por hacer una pequeña comprobación. Vamos a ejecutar nuestra aplicación y una vez se muestre el formulario en el navegador correspondiente, comprobaremos el código HTML de la página. Por simplicidad sólo se mostrará una parte, la referente al campo nombre:

```
<tr>
  <td>
    <label for="Nombre">Nombre</label>
  </td>
  <td>
    <input type="text" data-val="true" data-val-required="Debe introducir
el nombre" id="Nombre" name="Nombre" value="" />
  </td>
  <td>
    <span style="color:red" class="field-validation-valid" data-valmsg-
for="Nombre" data-valmsg-replace="true"></span>
  </td>
</tr>
```

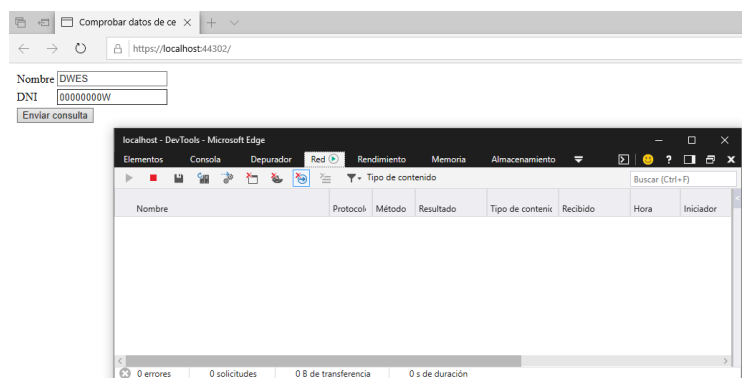
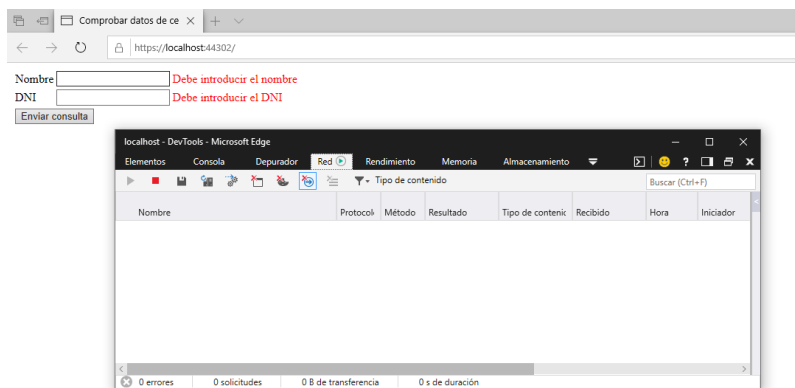
Hay varias cosas interesantes en este código. En primer lugar podemos comprobar cómo, efectivamente, los atributos han sido añadidas al HTML resultante sin tener que hacer nada especial para ello. Los atributos *data-val* indican si el campo debe ser validado. Los campos requeridos incluyen un atributo *data-val-required*, cuyo valor es el mensaje que se mostrará de no introducir un valor. En el *span* en el que se muestra el mensaje de error podemos comprobar como se ha añadido un atributo *data-valmsg-for*, que indica el campo para el que se mostrará el mensaje de error.

2. Una vez que hemos comprobado que las etiquetas necesarias ya están añadidas al HTML, lo único que nos queda es incluir las librerías JavaScript necesarias para que se realice la validación, en nuestro caso la librería jQuery y sus módulos de validación. Hay que recordar que en nuestra aplicación, como es habitual en ASP.NET MVC estamos utilizando una plantilla, especificada en *\_Layout.cshtml*. Esta es la única vista que contiene las etiquetas *<html>*, *<head>* y *<body>*, entre otras, ya que el resto de vistas son "inyectadas" dentro de esta para construir la página que se muestra en el navegador. Por lo tanto, este será el lugar en el que incluyamos nuestros scripts, de modo que cualquier vista los tenga disponibles. Por lo tanto, añadimos las siguientes líneas antes de la etiqueta de cierre *</body>*:

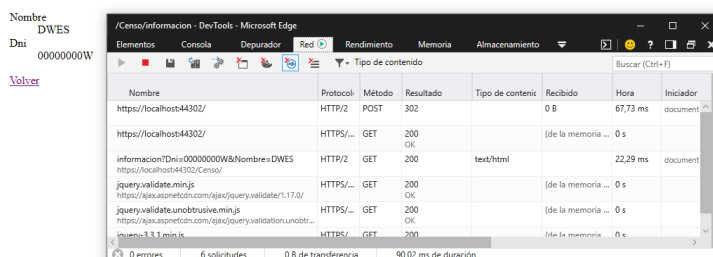
```
<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-
3.3.1.min.js"></script>
<script
src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.17.0/jquery.validate.m
in.js"></script>
<script
src="https://ajax.aspnetcdn.com/ajax/jquery.validation.unobtrusive/3.2.9/jque
ry.validate.unobtrusive.min.js"></script>
```

3. Si ejecutamos ahora nuestra aplicación, veremos que la validación funciona igual que hasta ahora, sólo que en este caso se está realizando directamente en el navegador.

Para comprobarlo podemos abrir las herramientas de desarrollador de nuestro navegador y comprobar como al darle al botón de aceptar para enviar el formulario, los mensajes de error se muestran, pero no se realiza ninguna llamada al servidor. En cuanto escribimos la primera letra en el campo nombre, por ejemplo, el mensaje de error desaparece de forma inmediata. La única llamada al servidor tendrá lugar una vez los datos han sido introducidos correctamente.



#### Información



- En este momento ya tenemos validación en ambos extremos en nuestra aplicación de ejemplo, pero existe un pequeño problema que vamos a solucionar. Al cargar los scripts de validación desde nuestra plantilla, hacemos que estos estén disponibles para todas nuestras vistas, que es lo que en principio puede parecerse correcto. Si lo analizamos con más detenimiento nos daremos cuenta de que esto representa un problema de rendimiento.



Todas nuestras páginas cargarán los scripts de validación, cuando en realidad éstos sólo son necesarios para las páginas que contengan formularios de entrada de datos. En todos los demás casos esto supone un trabajo extra, solicitando al servidor y cargando en nuestro navegador unos scripts que no vamos a utilizar.

Por esta razón, la norma general en ASP.NET MVC consiste en que solamente se cargarán directamente en el layout o plantilla los scripts necesarios para todas nuestras páginas, o en su defecto para casi todas, asumiendo una pequeña carga innecesaria para los casos en los que no se usan. Para los scripts que se utilizan sólo en ciertas páginas, como por ejemplo los que se usan para la validación de lado cliente, utilizaremos una vista parcial o Partial View, permitiendo de este modo especificar las vistas que los usan y las que no.

Comenzaremos por crear una nueva vista llamada *\_ValidationScriptsPartial.cshtml*, que como va a ser utilizada por diferentes vistas estará en *Views/Shared*. Esta vista parcial contendrá únicamente las importaciones de los scripts, que eliminaremos del layout.

```
<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"></script>
<script src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.17.0/jquery.validate.min.js"></script>
<script src="https://ajax.aspnetcdn.com/ajax/jquery.validation.unobtrusive/3.2.9/jquery.validate.unobtrusive.min.js"></script>
```

- En el lugar en el que estaban los scripts en el fichero *\_Layout.cshtml*, es decir, antes del elemento de cierre de la etiqueta *body*, incluiremos una sección, que indica que en ese lugar se podría inyectar una vista parcial si es especificado en una vista. Establecemos el valor *required* a *false*, de modo que sólo las vistas que lo deseen insertarán código desde una vista parcial en la dicha sección.

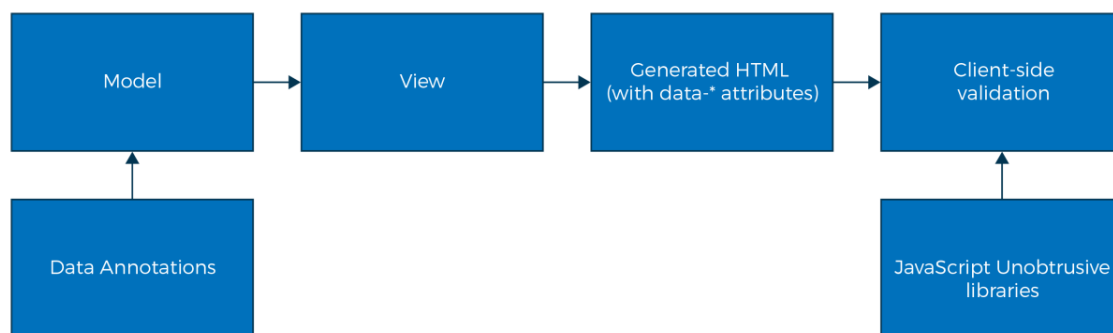
```
@RenderSection("Scripts", required: false)
```

- Por último, sólo nos quedaría especificar, en las vistas que deseemos, que se incluya nuestra vista parcial que contiene los scripts de validación. En nuestro ejemplo lo añadiremos en la vista *Comprobar.cshtml*, añadiendo el siguiente código al final del fichero:

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

Este código indica que queremos insertar, dentro de la sección *Scripts* de nuestra plantilla, la vista parcial cuyo nombre es *\_ValidationScriptsPartial*.

Para finalizar, se muestra un gráfico que resume de forma esquemática el proceso de validación del lado cliente y los elementos que intervienen en el mismo.



#### Bibliografía:

- ASP.NET Core 2 Fundamentals.  
OnurGumus, Mugilan T.S. Ragupathi  
Copyright © 2018 Packt Publishing
- Hands-On Full-Stack Web Development with ASP.NET Core  
Tamir Dresher, Amir Zuker and Shay Friedman  
Copyright © 2018 Packt Publishing