Práctica 5: Binding y validaciones

Asignatura: Desarrollo web en entorno servidor

Ciclo Superior Desarrollo de Aplicaciones Web.

Objetivo.

El objetivo de esta práctica es que te familiarices con algunos conceptos básicos para la implementación de controladores, el uso del binding y el sistema de validaciones integrado de ASP.NET Core MVC.

Binding

1. Crea una entidad de datos como la siguiente. Añade otras propiedades y las restricciones del modelo que consideres oportuno usando data annotations o implementando IValidatableObject:

```
public class Friend
{
    [StringLength(30), Required] public
    string Name { get; set; }

    [Range(1, 120)]
    public int Age { get; set; }

    [Required, EmailAddress]
    public string Email { get; set; }

    // Other properties
}
```

2. Crea un componente controlador capaz de procesar una petición de tipo GET hacia la URL /friends/create, retornando al usuario un formulario de edición de datos de la entidad. Esquemáticamente, esta vista contendrá el siguiente código:

```
<form method="post" action="/friends/create">
    ...
    Name: <input type="text" name="name" /><br/>
    Age:  <input type="text" name="age" /><br/>
    ...
    <input type="submit" value="Submit">
</form>
```

3. Introduce en el controlador el código necesario para procesar las peticiones de tipo POST que serán generadas desde este formulario. Introduce un objeto Friend como parámetro de entrada de la acción. Establece en ella un breakpoint y observa cómo el binder está creando la instancia y poblándola de forma automática.

4. Crea una nueva entidad de datos para representar la dirección de tus amigos, y llámala Address. Crea en la entidad Friend una propiedad de tipo Address que permita almacenar esta información.

```
public class Address
{
    [StringLength(50), Required] public
    string Street { get; set; }

    [StringLength(5)]
    public string ZipCode { get; set; }
}
```

5. Actualiza el formulario de edición para mostrar cuadros de texto que permitan editar esta nueva información. Comprueba cómo el binder está siendo capaz de instanciar y poblar también los tipos complejos contenidos en la entidad principal.

Validaciones

1. En la acción donde se reciben los datos del formulario introduce código que te permita determinar si los datos recibidos son válidos o no, según las restricciones definidas sobre las entidades. En caso afirmativo, retorna al usuario el mensaje "OK", y el mensaje "ERROR" en caso contrario.
2. Establece de nuevo un breakpoint y navega por el contenido de la propiedad ModelState. Observa que contiene los campos obtenidos de la petición, y por cada uno de ellos es posible consultar información interesante, como si su valor es correcto en función de las restricciones definidas para el mismo, o el valor recibido "en crudo" (antes de ser transformado).
3. Intenta obtener la lista de campos erróneos cuando los datos no son válidos, y mostrarlos al usuario por pantalla. Es decir, cuando ModelState.IsValid sea false, el usuario deberá obtener por pantalla la lista de campos incorrectos, que puedes obtener, por ejemplo, así:

```
   var errorFields = string.Join(",", ModelState.Keys.Where(key =>
!ModelState.IsValidField(key)));
```

Mantenimiento de estado entre peticiones

Cuando un usuario envía al servidor un formulario que contiene datos incorrectos, lo habitual es que el sistema le vuelva a mostrar el mismo formulario con los datos que ha introducido y le indique cuáles de ellos debe corregir.

Veremos una primera aproximación para conseguirlo:

1. Haz que la vista que genera el formulario sea capaz de recibir datos tipados desde el controlador. Esto lo estudiaremos dentro de poco, aunque ya habíamos adelantado en temas anteriores que era algo que podíamos conseguir introduciendo en ella la siguiente directiva Razor:

```
@model MyProject.Models.Friend
```

   A continuación, establece los valores por defecto para cada uno de los controles de edición utilizando la propiedad Model disponible en la vista. Recordarás que la propiedad Model contiene un objeto del tipo que hayamos especificado en la directiva anterior, y que es suministrado por el controlador.

```
<input type="text" name="name" value="@Model.Name" /><br>
<input type="text" name="age" value="@Model.Age" /><br>
<input type="text" name="address.street"
value="@Model.Address.Street" /><br>
...
```

2. Retoca el código de la acción desde la que retornas el formulario de edición, y suminístrale a la vista un objeto Friend inicializado, por ejemplo, así:

```
public ActionResult Create()
{
    var newFriend = new Friend()
    {
        Address = new Address()
    };
    return View(newFriend); // Send this object to the view
}
```

3. Por último, introduce el siguiente código en la acción donde se reciben los datos del formulario:

```
[HttpPost]
public ActionResult Create(Friend friend)
{
   if(!ModelState.IsValid)
      return View(friend);

   ...
}
```

4. Inicia el proyecto, y comprueba que el valor de los campos del formulario está persistiendo entre distintas llamadas. Si el estado del modelo es inválido, se retorna la vista suministrándole el mismo objeto que estamos recibiendo, y dado que esos valores son usados desde la vista como valores iniciales de los <input>, hemos conseguido que éstos no se pierdan entre las llamadas cuando los datos son incorrectos.

5. Aunque los estudiaremos en el módulo dedicado a la capa Vista, para mejorar aún más el resultado de tu aplicación, puedes utilizar los siguientes helpers (o métodos de ayuda) sobre el formulario de datos:
   a. **`@Html.ValidationMessage("fieldName")`**, muestra los mensajes de error en validación para el campo especificado.
   b. **`@Html.ValidationSummary()`**, muestra todos los mensajes de error de las validaciones que no han sido satisfactorias.

```
@Html.ValidationSummary()
<form method="post" action="friends/create">
   ...
   <input type="text" name="name" value="@Model.Name" />
@Html.ValidationMessage("name") <br/>
```