

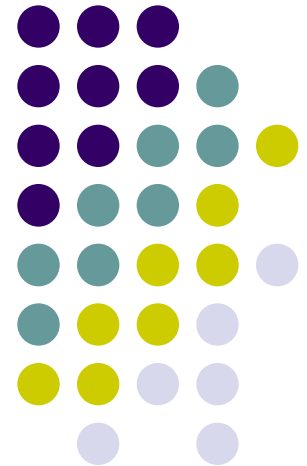
Desarrollo de Aplicaciones Web

Desarrollo Web en Entornos de Servidor



Tema 7

Desarrollo de Aplicaciones Web MVC (II)



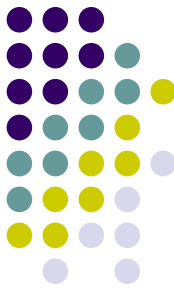
Vicente J. Aracil Miralles

vj.aracilmiralles@edu.gva.es

25/11/2024

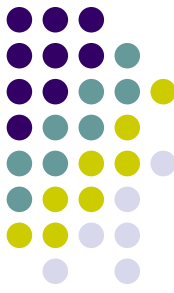
Tema 7

Desarrollo de Aplicaciones Web MVC



Objetivos

- Conocer las operaciones de acceso a datos que se realizan en las acciones del controlador
- Especificar código que permita implementar los procesamientos básicos para recuperar información mediante operaciones de búsqueda y consulta, así como para actualizar información mediante operaciones de inserción, eliminación y modificación
- Identificar las operaciones con la información almacenada, así como los procesamientos de operaciones combinadas de tipo CRUD (*Create, Read, Update and Delete*)
- Comprender el mecanismo de paso de información del modelo a una vista desde el controlador, a través de la directiva `@model`
- Comprender la forma cómo se realiza el enlace entre los datos y la interfaz de usuario que se realiza en las vistas



Tema 7

Desarrollo de Aplicaciones Web MVC

Contenidos

- 7.1 Operaciones sobre los datos
- 7.2 Operaciones para recuperar datos
- 7.3 Operaciones para actualizar datos
- 7.4 Enlace entre los datos y la interfaz Web de usuario



Generalidades

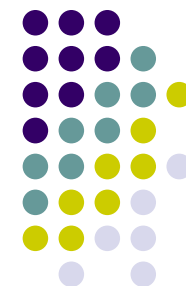
- En las aplicaciones Web basadas en ASP.NET Core MVC, se utiliza la técnica denominada **ORM (*Object Relation Mapper*)**
 - La herramienta ORM que se va a utilizar se denomina ***Entity Framework Core (EF Core)***
 - Proporciona una capa interpuesta entre los proveedores de datos y la aplicación Web. Esta capa de software facilita a los desarrolladores trabajar con un modelo conceptual de datos de alto nivel, que representa a los datos que maneja la aplicación Web mediante un conjunto de objetos
 - EF Core permite establecer la **vinculación entre la información almacenada en la base de datos y el modelo** de la aplicación Web
 - Una vez realizada esa vinculación mediante *EF Core*, existirá una base de datos enlazada con la aplicación Web a través:
 - Una clase del contexto de datos
 - Varias clases de datos, que representan a las entidades de datos de la aplicación
 - De esta manera, **el modelo es una representación en objetos de la información que maneja la aplicación Web**. Y, por tanto:
 - Se podrán realizar **operaciones de manipulación de datos sobre el modelo** para acceder a la información almacenada y así, poder desarrollar los procesamientos adecuados



Operaciones de manipulación de datos

- **Las operaciones de manipulación de datos básicas**, que suelen ejecutarse en las aplicaciones Web, se pueden clasificar en dos grupos:
 - **Operaciones para recuperar los datos**
 - Consisten en **recuperar una fila o un conjunto de filas** de una tabla, mediante operaciones de búsqueda o **consulta**. Estas operaciones **permiten obtener y explotar la información almacenada**, atendiendo a diversos criterios de búsqueda o consulta
 - **Operaciones para actualizar los datos**
 - Consisten en **añadir una nueva fila, modificar una fila existente o eliminar una fila existente** en una tabla. Estas operaciones permiten **mantener actualizada la información almacenada**, de modo que la información almacenada pueda representar la realidad del momento
- **En las aplicaciones Web basadas en ASP.NET Core MVC, la responsabilidad del acceso y el enlace a los datos recae en el código**
 - Por regla general, se acepta que:
 - **Las operaciones de manipulación de datos se realizan en las acciones del controlador**
 - **El enlace entre los datos almacenados y la interfaz Web de usuario se realiza en las vistas**
 - De manera que, las vistas generan dinámicamente en el servidor Web, las páginas HTML de respuesta que se envían al cliente como resultado del procesamiento

7.1 Operaciones sobre los datos



Operaciones de manipulación combinadas sobre una tabla

- **Procesamiento de tipo CRUD (*Create, Read, Update and Delete*)**

- Las operaciones de manipulación de datos, que se aplican sobre una tabla, suelen agruparse para expresar un procesamiento típico y habitual de las aplicaciones informáticas denominado CRUD
- Permite **mantener la actualidad de los datos** almacenados

Los proyectos de ASP.NET Core MVC disponen de un asistente que permite crear de forma automática los controladores y las vistas asociadas que implementan un procesamiento de tipo CRUD para cada tabla de la base de datos

MvcBiblioteca Home Privacy Libros Autores Generos					Register Login
Libros					
Create New					
Título	Fecha de Edición	Núm.Páginas	Autor	Genero	
Los pilares de la tierra	21/03/1990	1403	Ken Follet	Novela histórica	Edit Details Delete
El cisne negro	17/06/2001	534	Nassim Nicholas Taleb	Ensayo	Edit Details Delete
El profesor	01/10/2006	291	Frank McCourt	Novela	Edit Details Delete
Un mundo sin fin	23/05/1991	503	Ken Follet	Novela	Edit Details Delete
Churchill	28/02/1985		Alan Moorehead	Biografía	Edit Details Delete
Eclipse	01/06/2007	865	Stephenie Meyer	Novela juvenil	Edit Details Delete
Canto General			Pablo Neruda	Poesía	Edit Details Delete
© 2022 - MvcBiblioteca - Privacy					

7.1 Operaciones sobre los datos



Métodos de manipulación de datos de la API de EF Core

- **Permiten expresar las operaciones de manipulación de datos en las aplicaciones Web basadas en ASP.NET Core MVC**
 - La API (*Application Programming Interface*) de *EF Core* ofrece métodos que pueden ser utilizados por otro software como una capa de abstracción
 - Estos métodos se aplican **sobre los objetos de tipo DbSet** creados en la clase del contexto de datos que **representan a las tablas** de la base de datos

En los dos siguientes apartados, se estudian algunos de los métodos de la API de EF Core más utilizados para recuperar datos y para actualizar datos, respectivamente. Para los ejemplos que aparecen, se utiliza la siguiente declaración de clase del contexto de datos:

```
public class MvcBibliotecaContexto : DbContext
{
    public MvcBibliotecaContexto(DbContextOptions<MvcBibliotecaContexto> options) : base(options)
    {
    }

    public DbSet<Autor>? Autores { get; set; }
    public DbSet<Genero>? Generos { get; set; }
    public DbSet<Libro>? Libros { get; set; }
}
```

7.2 Operaciones para recuperar datos



Métodos de la API de *EF Core* para recuperar datos

- **ToListAsync()**. Obtiene una lista que contiene todas las entidades del objeto DbSet o de la consulta especificada. Por ejemplo:

```
var autores = await _context.Autores.ToListAsync()
```

El método *ToListAsync()*, al igual que otros métodos de la API de EF Core, es un método asíncrono, por lo que su llamada debe ir precedida del operador *await*. Por razones de mejora del rendimiento, las aplicaciones Web basadas en ASP.NET Core MVC utilizan la **programación asíncrona**, basada en la llamada de métodos asíncronos, mediante el modificador *asyn* y el operador *await* de C#

Si el acceso a un recurso Web queda bloqueado en un proceso sincrónico, todo el procesamiento pendiente de ejecución deberá esperar. Mientras que, **en un procesamiento asíncrono, la ejecución de la aplicación Web puede continuar con otra tarea que sea independiente del recurso bloqueado** y que, normalmente, puede estar relacionada con la construcción de la interfaz Web

Por convención, los métodos que se llaman de forma asíncrona incluyen el sufijo *Async* en el nombre del método

- **FindAsync()**. Busca y recupera una entidad cuya clave primaria coincida con el valor del argumento especificado. Por ejemplo:

```
var autor = await _context.Autores.FindAsync(id);
```


7.2 Operaciones para recuperar datos



Métodos de la API de *EF Core* para recuperar datos

- **Include()**. Especifica las entidades relacionadas a incluir en el resultado de una consulta

```
var libros = _context.Libros
    .Include(l => l.Autor).Include(l => l.Genero);
```

- **FirstOrDefaultAsync()**. Devuelve el primer elemento de una secuencia de entidades que satisface la condición especificada o un valor predeterminado (*null*) si no se encuentra ningún elemento

```
var libro = await _context.Libros
    .Include(l => l.Autor)
    .Include(l => l.Genero)
    .FirstOrDefaultAsync(m => m.Id == id);
```

- **FromSqlRaw()**. Ejecuta una consulta de SQL, que se facilita como argumento en una cadena de caracteres sin formato, devolviendo el conjunto de entidades resultante. El método *FromSqlRaw()* solo se puede usar en la raíz de la consulta, es decir, directamente sobre el objeto de tipo *DbSet*.

```
var libros = _context.Libros
    .FromSqlRaw("SELECT * FROM dbo.Libros")
    .Include(l => l.Autor)
    .Include(l => l.Genero);
```

7.2 Operaciones para recuperar datos



Consultas de *LINQ to Entities* para recuperar datos

- Además de los métodos de la API de *EF Core*, también se pueden emplear las consultas de *LINQ to Entities* para recuperar datos
 - Estas consultas permiten realizar **operaciones complejas de filtrado, ordenación y agrupamiento** sobre un objeto de tipo `DbSet`
 - El conjunto de entidades resultante de una consulta *LINQ to Entities* implementa la interfaz ***IQueryable*** que está definida en el espacio de nombres *System.Linq*
 - Las consultas de *LINQ to Entities* se pueden formular usando dos sintaxis diferentes:
 - **Sintaxis de expresiones de consulta.** Se utiliza una sintaxis declarativa de alto nivel para especificar las consultas. Por ejemplo, para obtener una colección de libros de novela:

```
var libros = from fila in _context.Libros
              where fila.Genero.Descripcion == "Novela"
              select fila;
```

- **Sintaxis de consulta basada en métodos.** Se recomienda utilizar esta sintaxis. Permite establecer una secuencia de llamadas a los métodos de manipulación de datos. Por ejemplo:

```
var libros = _context.Libros
              .Where(x => x.Genero.Descripcion == "Novela");
```

7.3 Operaciones para actualizar datos



Métodos de la API de *EF Core* para actualizar datos

- **Permiten mantener la información almacenada en un estado consistente, de manera que se pueda representar la realidad existente en cada momento**

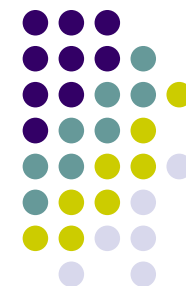
Las operaciones de actualización sobre una tabla o conjunto de entidades son tres: añadir una fila o entidad, modificar una fila o entidad existente y eliminar una fila o entidad existente

- **Añadir una fila.** Para agregar una nueva entidad a una colección se utiliza el método **Add()**. Como parámetro se especifica el objeto que contiene la información relativa a la nueva entidad, por lo que, previamente, habrán asignado los valores correspondientes a las propiedades de la nueva entidad a añadir. Por ejemplo:

```
_context.Add(libro);  
await _context.SaveChangesAsync();
```

Finalmente, se invoca el método **SaveChanges()** para producir reflejo de los cambios realizados en el modelo sobre la base de datos.

7.3 Operaciones para actualizar datos



Métodos de la API de *EF Core* para actualizar datos

- **Modificar una fila.** Para modificar una entidad existente en una colección se utiliza el método `Update()`. Como parámetro se especifica el objeto que contiene los datos ya modificados, por lo que, previamente, se habrán asignado los valores correspondientes a las propiedades de la entidad a modificar. Por ejemplo:

```
_context.Update(libro);  
await _context.SaveChangesAsync();
```

- **Eliminar una fila.** Para eliminar una entidad existente en una colección se utiliza el método `Remove()` sobre el objeto de tipo `DbSet` correspondiente. Como parámetro se especifica el objeto a eliminar, por lo que, previamente, se habrá obtenido el objeto a eliminar mediante el método `FindAsync()`. Por ejemplo:

```
var libro = await _context.Libros.FindAsync(id);  
_context.Libros.Remove(libro);  
await _context.SaveChangesAsync();
```



Generalidades

- El enlace entre los datos y la interfaz Web de usuario se realiza en las vistas
 - En las aplicaciones Web basadas en ASP.NET Core MVC:
 - Las operaciones de manipulación de datos se realizan en las acciones del controlador
 - El enlace entre los datos almacenados y la interfaz Web de usuario se realiza en las vistas

De esta manera, las vistas generan dinámicamente, en el servidor Web, las páginas de respuesta que se envían al cliente

- Es necesario tener en cuenta, que **la responsabilidad del acceso y el enlace a los datos recae en el código**, por lo que es necesario comprender de manera detallada **la utilización del código** para poder completar los procesamientos deseados
- A continuación, se estudian **dos ejemplos** que permiten comprender cómo se realiza la manipulación de datos en el controlador y el enlace entre los datos almacenados y la interfaz Web en la vista
 - El primer ejemplo, muestra un procesamiento para **recuperar información almacenada y presentar visualmente una lista de los datos** almacenados
 - El segundo ejemplo, muestra un procesamiento de actualización para **modificar los datos almacenados**



Procesamiento para recuperar información

- **Para realizar cualquier procesamiento que implique una recuperación de los datos almacenados:**
 - En primer lugar, se realiza una operación de **búsqueda o consulta** de datos sobre el modelo en una acción del controlador. Estas operaciones permiten recuperar los datos que se van a procesar, que podrán ser:
 - Una **entidad**, si se ha recuperado una sola fila
 - Una **colección de entidades**, si se han recuperado un conjunto de filas
 - En segundo lugar, la acción del controlador activa la vista correspondiente. Y, al mismo tiempo, **se pasan hacia la vista los datos que han sido recuperados y que van a ser presentados visualmente** sobre la interfaz Web
 - En tercer lugar, ya en la vista, **se accede a los datos del modelo que se ha pasado a la vista desde el controlador** y que suele denominarse **Modelo de vista (*View Model*)**
 - Finalmente, **la vista genera la página de respuesta** que especifica una presentación visual adecuada de los datos recuperados sobre interfaz Web

7.4 Enlace entre los datos y la interfaz Web de usuario



Ejemplo 1. Procesamiento para recuperar información

- Para conocer con detalle el código que facilita el acceso a los datos almacenados, así como el enlace de los datos con la interfaz generada por las vistas, continuando con el ejemplo, se estudia el código para presentar la lista de todos los libros almacenados
 - Para ello, se incluye el código de la acción *Index()* del controlador *LibrosController.cs* y el código de la vista *Index.cshtml* alojada en la subcarpeta de vistas */Views/Libros*

Acción *Index()* del controlador *LibrosController.cs*

```
// GET: Libros
public async Task<IActionResult> Index()
{
    var libros = _context.Libros.Include(l => l.Autor).Include(l => l.Genero);

    return View(await libros.ToListAsync());
}
```

Se activa la vista correspondiente y se le pasa la lista de todas las entidades de tipo Libro

Se obtiene la variable *libros* que contiene la colección de todas las entidades de tipo Libro y se incluyen los valores de las entidades *Autor* y *Genero* relacionadas

7.4 Enlace entre los datos y la interfaz Web de usuario



Ejemplo 1. Procesamiento para recuperar información

Vista *Index.cshtml*, que está alojada en la subcarpeta de vistas */Views/Libros*

```
@model IEnumerable<MvcBiblioteca.Models.Libro>
```

```
@{  
    ViewData["Title"] = "Index";  
}  
<h1>Libros</h1>
```

La directiva `@model` especifica el tipo de datos del modelo que se pasa a la vista desde el controlador

```
<p>  
    <a asp-action="Create">Create New</a>  
</p>  
<table class="table">  
    <thead>  
        <tr>  
            <th>  
                @Html.DisplayNameFor(model => model.Titulo)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.FechaEdicion)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.NumeroPaginas)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Autor)  
            </th>  
            <th>  
                @Html.DisplayNameFor(model => model.Genero)  
            </th>
```

El asistente de HTML *DisplayNameFor* obtiene el nombre de la propiedad del modelo que se pasa a la vista y que se especifica mediante la expresión *lambda* que se incluye como argumento

El modelo que se pasa a la vista desde el controlador suele denominarse *Modelo de vista (View model)*

7.4 Enlace entre los datos y la interfaz Web de usuario



Ejemplo 1. Procesamiento para recuperar información

```
<th></th>
</tr>
</thead>
<tbody>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Titulo)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.FechaEdicion)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.NumeroPaginas)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Autor.Nombre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Genero.Descripcion)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
            <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
            <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
        </td>
    </tr>
}
</tbody>
</table>
```

La propiedad *Model* facilita el acceso a los valores de la colección de libros del modelo que se pasa a la vista para que los datos almacenados puedan ser enlazados en la interfaz Web

El asistente de HTML *DisplayFor* obtiene el valor de la propiedad del modelo que se pasa a la vista y que se especifica mediante la expresión *lambda* que se incluye como argumento

Las expresiones *lambda* permiten escribir funciones anónimas que pueden pasarse como un argumento en las llamadas a funciones

7.4 Enlace entre los datos y la interfaz Web de usuario



Ejemplo 1. Procesamiento para recuperar información

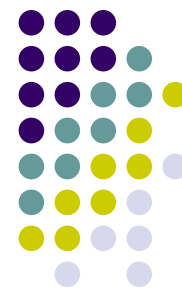
MvcBiblioteca Home Privacy Libros Autores Generos

Register Login

Libros

[Create New](#)

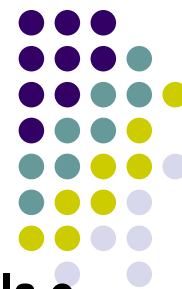
Título	Fecha de Edición	Núm.Páginas	Autor	Genero	
Los pilares de la tierra	21/03/1990	1403	Ken Follet	Novela histórica	Edit Details Delete
El cisne negro	17/06/2001	534	Nassim Nicholas Taleb	Ensayo	Edit Details Delete
El profesor	01/10/2006	291	Frank McCourt	Novela	Edit Details Delete
Un mundo sin fin	23/05/1991	503	Ken Follet	Novela	Edit Details Delete
Churchill	28/02/1985		Alan Moorehead	Biografía	Edit Details Delete
Eclipse	01/06/2007	865	Stephenie Meyer	Novela juvenil	Edit Details Delete
Canto General			Pablo Neruda	Poesía	Edit Details Delete



Procesamiento para actualizar información

- **Permiten mantener la actualidad de la información almacenada en una tabla**
 - El procesamiento que se vaya a realizar varía según el tipo de operación de actualización de información que se vaya a realizar en una tabla. Estas operaciones pueden:
 - **Añadir una nueva entidad**
 - **Modificar los datos de una entidad existente**
 - **Eliminar una entidad existente**
 - Es importante que tener siempre presente que solo interviene una única entidad o fila en cualquiera de las tres operaciones de actualización de datos de una tabla
 - Aunque varía la operación a realizar, este tipo de procesamientos tienen en común que:
 - En primer lugar, se actualizará el modelo ejecutando una de las tres operaciones de actualización posibles, mediante la llamada a uno de los métodos siguientes: **Add()**, **Update()** o **Remove()**
 - Y, en segundo lugar, se realizará la llamada al método **SaveChanges()** para que se produzca reflejo de los cambios realizados en el modelo sobre la base de datos
 - En los procedimientos para añadir o para editar una entidad, la interfaz Web deberá incorporar un formulario para facilitar al usuario las tareas de introducción o de edición de los datos, según sea el caso

7.4 Enlace entre los datos y la interfaz Web de usuario



Ejemplo 2. Procesamiento para actualizar información

- Es importante conocer con detalle el código que permite modificar una fila o entidad, así como el enlace de los datos con la interfaz generada por las vistas. Se estudia el código para modificar la información de un libro existente
 - Para ello, se incluye el código de las acciones GET *Edit()* y POST *Edit()* del controlador *LibrosController.cs* y el código de la vista *Edit.cshtml* alojada en */Views/Libros*

Acciones *Edit()* del controlador *LibrosController.cs*

```
...  
// GET: Libros/Edit/5  
public async Task<IActionResult> Edit(int? id)  
{  
    if (id == null || _context.Libros == null)  
    {  
        return NotFound();  
    }  
  
    var libro = await _context.Libros.FindAsync(id);  
    if (libro == null)  
    {  
        return NotFound();  
    }  
    ViewData["AutorId"] = new SelectList(_context.Autores, "Id", "Nombre", libro.AutorId);  
    ViewData["GeneroId"] = new SelectList(_context.Generos, "Id", "Descripcion", libro.GeneroId);  
    return View(libro);  
}
```


Se realiza una búsqueda por el valor de *Id* para obtener la entidad libro a editar

Se activa la vista correspondiente y se le pasa la entidad *libro* encontrada

7.4 Enlace entre los datos y la interfaz Web de usuario



Ejemplo 2. Procesamiento para actualizar información



```
// POST: Libros/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
    [Bind("Id,Titulo,FechaEdicion,NumeroPaginas,AutorId,GeneroId")] Libro libro)
{
    if (id != libro.Id) {
        return NotFound();
    }
    if (ModelState.IsValid) {
        try {
            _context.Update(libro);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException) {
            if (!LibroExists(libro.Id)) {
                return NotFound();
            }
            else {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["AutorId"] = new SelectList(_context.Autores, "Id", "Nombre", libro.AutorId);
    ViewData["GeneroId"] = new SelectList(_context.Generos, "Id", "Descripcion", libro.GeneroId);
    return View(libro);
}
...
```

Acción POST

Impide ataques de falsificación de solicitud

Compone la entidad *libro* con los valores recibidos a través del formulario

Modifica la entidad libro en el modelo

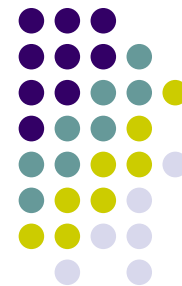
Produce reflejo de los cambios realizados sobre la Base de Datos

Redirecciona a la acción Index()

La propiedad *ViewData* permite pasar datos auxiliares hacia la vista

7.4 Enlace entre los datos y la interfaz Web de usuario

Ejemplo 2. Procesamiento para actualizar información



Vista *Edit.cshtml*, que está alojada en la subcarpeta de vistas */Views/Libros*

```
@model MvcBiblioteca.Models.Libro
```

```
@{  
    ViewData["Title"] = "Edit";  
}
```

La directiva `@model` especifica el tipo de datos del modelo que se pasa a la vista, que es de tipo *Libro*

```
<h1>Edit</h1>
```

```
<h4>Libro</h4>
```

```
<hr />
```

```
<div class="row">
```

```
    <div class="col-md-4">
```

```
        <form asp-action="Edit">
```

```
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
```

```
            <input type="hidden" asp-for="Id" />
```

```
            <div class="form-group">
```

```
                <label asp-for="Titulo" class="control-label"></label>
```

```
                <input asp-for="Titulo" class="form-control" />
```

```
                <span asp-validation-for="Titulo" class="text-danger"></span>
```

```
            </div>
```

```
            <div class="form-group">
```

```
                <label asp-for="FechaEdicion" class="control-label"></label>
```

```
                <input asp-for="FechaEdicion" class="form-control" />
```

```
                <span asp-validation-for="FechaEdicion" class="text-danger"></span>
```

```
            </div>
```

El atributo `asp-action` especifica el nombre de la acción POST que será destino del envío del formulario para que se realice su procesamiento

7.4 Enlace entre los datos y la interfaz de usuario Web



Ejemplo 2. Procesamiento para actualizar información

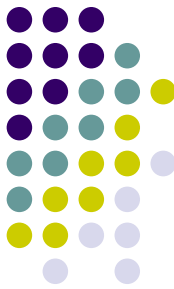
```
<div class="form-group">
  <label asp-for="NumeroPaginas" class="control-label"></label>
  <input asp-for="NumeroPaginas" class="form-control" />
  <span asp-validation-for="NumeroPaginas" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="AutorId" class="control-label"></label>
  <select asp-for="AutorId" class="form-control"
    asp-items="ViewBag.AutorId"></select>
  <span asp-validation-for="AutorId" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="GeneroId" class="control-label"></label>
  <select asp-for="GeneroId" class="form-control"
    asp-items="ViewBag.GeneroId"></select>
  <span asp-validation-for="GeneroId" class="text-danger"></span>
</div>
<div class="form-group">
  <input type="submit" value="Save" class="btn btn-primary" />
</div>
</form>
</div>
</div>
<div>
  <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

El atributo *asp-for* de cada control especifica el nombre de la propiedad del modelo que se pasa a la vista cuyo valor se está editando

Los asistentes de etiquetas (*Tag Helpers*) de los elementos *<label>*, *<input>*, *<select>*, etc. representan los controles del formulario

7.4 Enlace entre los datos y la interfaz de usuario Web

Ejemplo 2. Procesamiento para modificar información




MvcBiblioteca Home Privacy Libros Autores Generos Register Login

Edit

Libro

Título

Fecha de Edición
 

Núm.Páginas

Autor Principal

Género

[Save](#)

[Back to List](#)

© 2022 - MvcBiblioteca - [Privacy](#)