Tema 7: La Vista

Asignatura: Desarrollo web en entorno servidor

CS Desarrollo de Aplicaciones Web

Introducción

- En este capítulo veremos aspectos como:
  - Características de las vistas
  - Tipos de vistas
  - Implementación de vistas
  - Helpers HTML

Introducción

- La Vista se encarga de:
  - la **generación del resultado,** enviado finalmente al cliente
  - **maquetación del interfaz**, con los elementos necesarios para permitir al usuario interactuar con el sistema
  - la **implementación de la lógica de presentación**, cuando sea necesario.

Introducción

- La Vista contiene el **conjunto de componentes responsables de crear una representación del estado del sistema**, o en otras palabras, de su Modelo, que es quien finalmente lo almacena.
- **MUY IMPORTANTE:** Una vista puede contener **lógica exclusivamente destinada a tratar aspectos relativos a la presentación.** (Colores, forma de mostrar los datos…)

Introducción

- Una vista **siempre va a ser invocada desde un controlador**.
- Por ejemplo, este código provocaría el envío al cliente de la vista **Index.cshtml,** ubicada según convención en la carpeta **/Views/Home** del proyecto.

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

Introducción

- Este código, sin embargo, provocaría el envío al cliente de la vista **InicioSecundario.cshtml,** ubicada en la misma carpeta que Index, pero con un nombre diferente.

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View("InicioSecundario");
    }
}
```

FORMACIÓN PROFESIONAL MONTECASTELO

TIPOS DE VISTAS

Tipos de vistas

- Trabajaremos con tres diferentes tipos de vistas:
  - **Layouts o Vistas maestras**
    - Proporcionan la base sobre la que generar una vista de contenido.
  - **Vistas de contenido** de página, o simplemente **Vistas**
    - Encargadas de generar páginas completas
  - **Vistas parciales**
    - Encargadas de maquetar una porción de la página.

Tipos de vistas

Vistas de contenido

- Las utilizamos **para mostrar las acciones realizadas por el controlador**.
- Si tenemos una vista tipada, deberemos incluir la directiva **@model**, para poder acceder al valor obtenido desde la palabra clave **Model**

```
@model Ejemplo.Models.ErrorViewModel;

@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Showing error: @Model.RequestId </p>
</div>
```

Tipos de vistas

Vistas maestras o layouts

- Los Layouts son plantillas con extensión .cshtml y cuyo nombre comienza por un guión bajo ("_").
- La instrucción @RenderBody nos dice que en ese espacio se renderizará el cuerpo principal de la vista que está utilizando este Layout

Tipos de vistas

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Ejemplo</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/Ejemplo.styles.css" asp-append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Ejemplo</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse"
                aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>
```

Tipos de vistas

Vistas parciales

- Las vistas parciales se utilizan como mecanismos de reutilización de código de la Vista, pues permiten generar solamente una porción de la página con los datos que le son suministrados.
- Se implementan igual que una vista de contenido, con ficheros .cshtml
- Suelen usar como prefijo el guión bajo "_", para distinguirlos rápidamente de las vistas de contenido.

Tipos de vistas

Vistas parciales

```
@model MyProject.Models.MessageViewModel

<p>Message: @Model.Message</p>
<p>Delivery date: @Model.Date</p>
```

Tipos de vistas

Código de inicialización

- **Razor permite añadir a la carpeta /Views** del proyecto **el archivo _ViewStart.cshtml para incluir código de inicialización común** a todas las vistas.
- Su ejecución se llevará a cabo justo antes de ejecutar las vistas.
- Se suele usar para inicializar propiedades de todas las vistas en un único punto. Por ejemplo, para establecer el mismo Layout a todas las páginas.

Tipos de vistas

Código de inicialización

```
@{
Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Tipos de vistas

Creando vistas con Razor

- Como bien sabemos, crear vistas con el framework Razor es muy sencillo, ya que trabaja en dos modos:
  - **Modo marcado**, en el que espera encontrar exclusivamente código cliente (HTML, CSS, JS…)
  - **Modo código**, en el que espera encontrar exclusivamente código de servidor (C#)
- Para cambiar entre modos utilizaremos el carácter **@**

Tipos de vistas

Creando vistas con Razor

- Por ejemplo, aquí combinamos los dos modos:

```
<h3>Multiplication table of 3</h3>
<ul>
    @for (int i = 1; i <= 10; i++)
    {
        <li>
            @i x 3 = @(i*3)
        </li>
    }
</ul>
```

FORMACIÓN PROFESIONAL MONTECASTELO

HELPERS

Helpers

- Un **helper** es un método generador de código de la clase **Html**, que nos permitirá generar código cliente desde nuestras vistas de forma más rápida y eficiente.
- Por ejemplo, estos dos códigos son equivalentes:

```
<input type="text" name="ProductName" id="ProductName" value="@Model.ProductName" />
```

```
@Html.TextBox("ProductName")
```

- A continuación, veremos varios tipos de helpers que nos pueden ayudar.

Helpers

Generación de hipervínculos

- Seguro que en nuestra web necesitaremos introducir enlaces (hipervínculos), para guiar al usuario a sitios externos, o hacia las distintas zonas internas del sistema:
  - **Para los enlaces externos**, los enlaces se crearán como siempre, **introduciendo etiquetas <a>**
  - **Para los enlaces internos**, existen varios helpers para generar etiquetas <a> hacia acciones del sistema:
    - **Html.ActionLink()**
    - **Html.RouteLink()**

Helpers

Html.ActionLink()

- **Permite generar de forma sencilla enlaces hacia acciones**.
- Por ejemplo:

```
@Html.ActionLink(
    "Netbooks catalog",          //Texto del link
    "index",                     //Acción
    new { category="netbooks"} // Parámetros de ruta
)
```

Helpers

Html.RouteLink()

- Este helper es muy similar al anterior, pero permite un mayor control sobre la forma en que se generará la dirección URL.
- Por ejemplo, podremos indicar los parámetros de ruta en un objeto anónimo.
- Veremos un ejemplo a continuación.

Helpers

Html.RouteLink()

```
@Html.RouteLink(
    "Netbooks catalog",          //Texto del link
    "index",                     //Acción
    new {                        // Parámetros de ruta
        controller = "Products",
        category="netbooks",
        }
)
```

Helpers

Helper HTML

- Algunos de los helpers HTML más comunes en ASP.NET Core MVC 6 son:
  - **Html.BeginForm**
  - **Html.LabelFor**
  - **Html.TextBoxFor**
  - **Html.CheckBoxFor**
  - **Html.DropDownListFor**

Helpers

Html.BeginForm

- Crea un formulario HTML y establece la acción y el controlador que se ejecutarán cuando se envíe el formulario.
- Veremos un ejemplo a continuación:

Helpers

- En este ejemplo, veremos cómo crea un formulario HTML y establece la acción y el controlador que se ejecutarán cuando se envíe el formulario.
- Vemos como la acción es **Crear** y el controlador es **Empleado**. El parámetro FormMethod.Post indica que el formulario se enviará utilizando el método POST HTTP.

```
@using (Html.BeginForm("Crear", "Empleado", FormMethod.Post))
{
    <div class="form-group">
        <label for="Name">Nombre:</label>
        <input type="text" class="form-control" name="Name" id="Name" />
    </div>

    <div class="form-group">
        <label for="Email">Email:</label>
        <input type="email" class="form-control" name="Email" id="Email" />
    </div>

    <button type="submit" class="btn btn-primary">Guardar</button>
}
```

Helpers

Html.LabelFor y Html.TextBoxFor

- **Html.LabelFor**: Crea una etiqueta label para una propiedad de modelo
- **Html.TextBoxFor:** Crea un elemento input de tipo texto para una propiedad de modelo.
- Veremos un ejemplo a continuación:

Helpers

- En este ejemplo, el helper **Html.LabelFor** crea una etiqueta label para la propiedad Titulo del modelo. El método p => p.Titulo se utiliza para obtener el nombre de la propiedad del modelo.
- El helper **Html.TextBoxFor** crea un elemento input de tipo texto para la propiedad Titulo del modelo.

```
@model Ejemplo.Models.Post

<div class="form-group">
@Html.LabelFor(p => p.Titulo)
@Html.TextBoxFor(p => p.Titulo, new { @class = "form-control" })
</div>
```

Helpers

Html.CheckBoxFor

- Crea un elemento input de tipo casilla de verificación para una propiedad de modelo booleana.
- Veremos un ejemplo a continuación:

Helpers

- En este ejemplo, el helper **Html.CheckBoxFor** crea un elemento input de tipo casilla de verificación para la propiedad booleana IsSubscribed del modelo.
- El método p => p.EsPublicado se utiliza para obtener el nombre de la propiedad del modelo.

```
@model Ejemplo.Models.Post
<div class="form-group">
@Html.LabelFor(p => p.EsPublicado)
@Html.CheckBoxFor(p => p.EsPublicado)
</div>
```

Helpers

Html.DropDownListFor

- Crea un elemento select para una propiedad de modelo y llena el elemento con opciones de una lista de selección.
- Veremos un ejemplo a continuación:

Helpers

- En este ejemplo, el helper **Html.DropDownListFor** crea un elemento select para la propiedad Autor del modelo. El segundo parámetro new SelectList(ViewBag.Autores, "Value", "Text") se utiliza para llenar el elemento select con opciones de una lista de selección.
- El tercer parámetro "-- Selecciona un autor --" se utiliza para agregar una opción predeterminada con un valor vacío al elemento select.
- El cuarto parámetro new { @class = "form-control" } se utiliza para agregar la clase CSS form-control al elemento select.

```
@model Ejemplo.Models.Post
<div class="form-group">
@Html.LabelFor(p => p.Autor)
@Html.DropDownListFor(p => p.Autor,
    new SelectList(ViewBag.Autores, "Value", "Text"), "-- Selecciona un autor --", new { @class = "form-control" })
</div>
```

FORMACIÓN PROFESIONAL MONTECASTELO

Proyecto final

Simplemente, haz el ejercicio 1 del boletín de ejercicios.

? MINUTOS

KEEP CALM IT'S KAHOOT TIME

Tema 7: La Vista

Asignatura: Desarrollo web en entorno servidor

CS Desarrollo de Aplicaciones Web