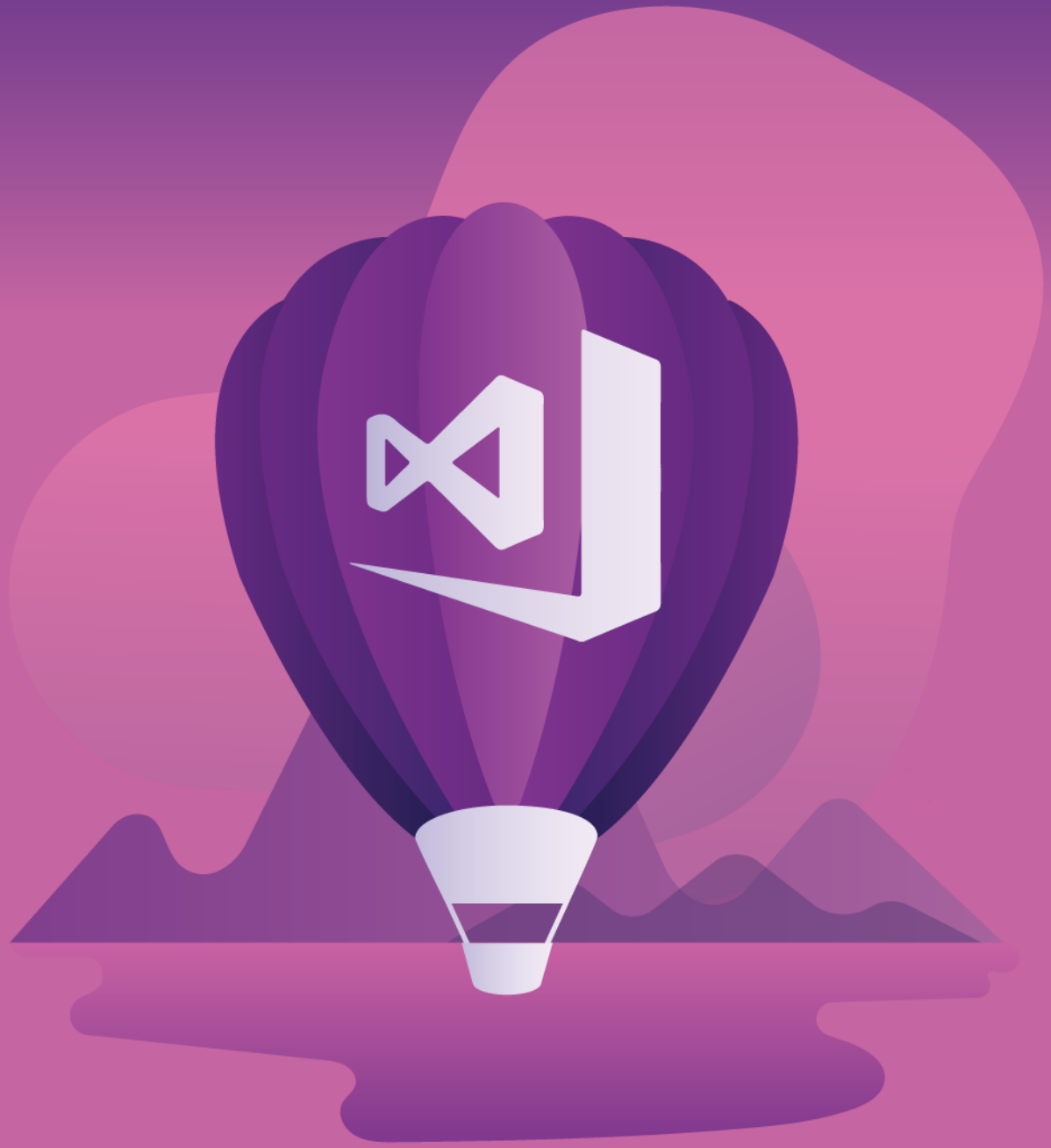


BIENVENIDOS
AL CURSO:

Especialización ASP.NET Core 5 Developer

SESIÓN 02

GALAXY
TRAINING

ÍNDICE

ASP.NET Core 5

ASP.NET Core 5 es un marco multiplataforma y de código abierto destinado a la compilación de modernas aplicaciones web basadas en la nube en Windows, Mac o Linux.

https://docs.microsoft.com/es-es/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0

Helpers

Qué son los Helper’s

Los Helper’s permiten que el código de servidor participe en la creación y la representación de elementos HTML en archivos de Razor. Por ejemplo, el TagHelper **ImageTagHelper** puede agregar un número de versión al nombre de imagen. Cada vez que la imagen cambia, el servidor genera una nueva versión única para la imagen, lo que garantiza que los clientes puedan obtener la imagen actual (en lugar de una imagen obsoleta almacenada en caché). Hay muchos asistentes de etiquetas integradas para tareas comunes (como la creación de formularios, vínculos, carga de activos, etc.) y existen muchos más a disposición en repositorios públicos de GitHub y como paquetes NuGet. Los Helper’s se crean en C# y tienen como destino elementos HTML en función del nombre de elemento, el nombre de atributo o la etiqueta principal. Por ejemplo, la aplicación auxiliar **LabelTagHelper** puede tener como destino el elemento HTML <label> cuando se aplican atributos LabelTagHelper.

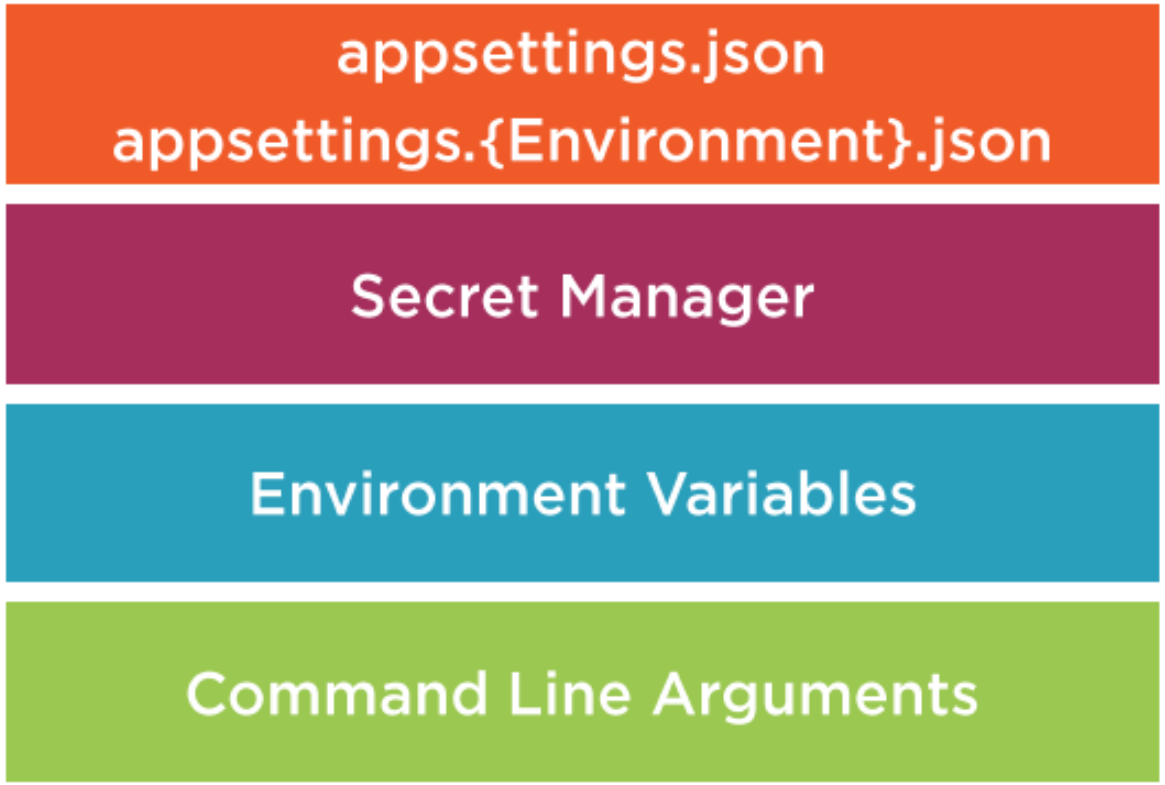
Entendiendo Tag Helpers

HTML Helpers	Tag Helpers
@HTML.EditorFor(m=>Name)	<input asp-for="Name"></input>
Codigo C# en las vistas	HTML amigable
Código poco elegante	Solo se escribe HTML
Puedes crear nuevas extensiones	Puedes crear tus propios Tag Helpers

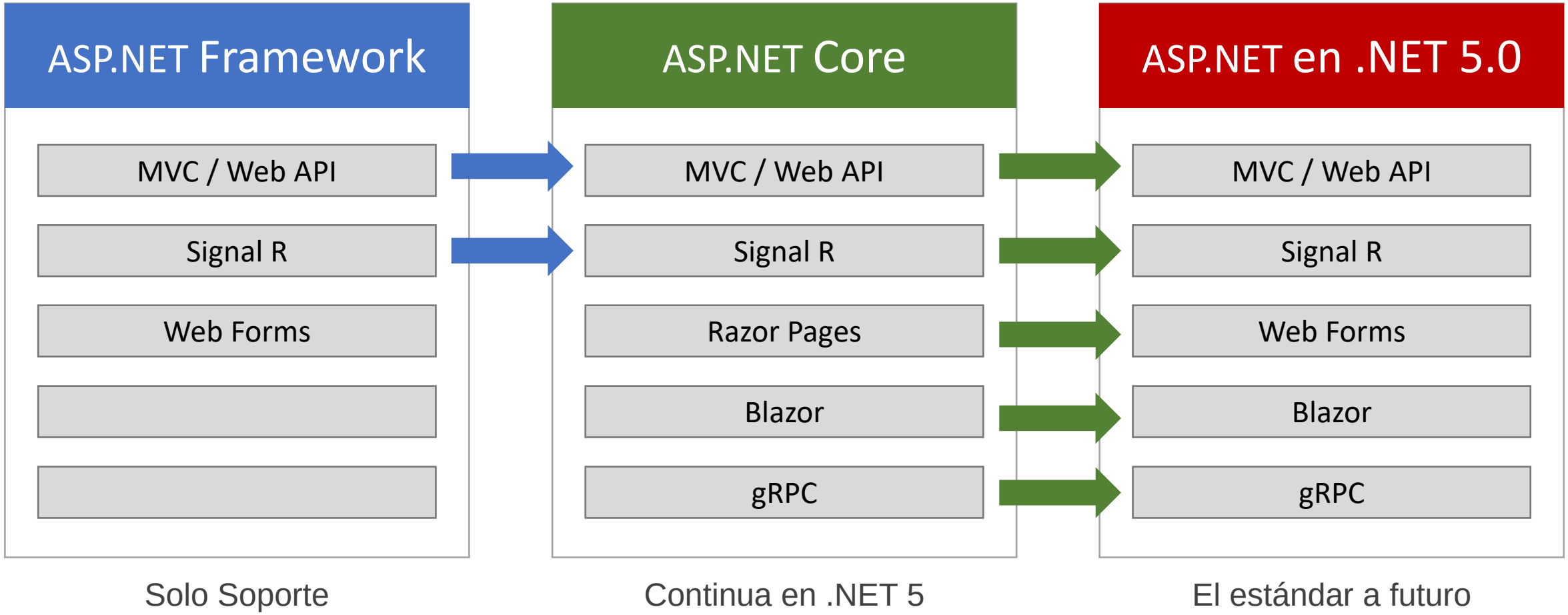
Sistema de configuración

ASP.NET Clasico	ASP.NET Core 5
Key-Value pairs	Key-Value pairs
Web.config y otros archivos XML	In Memoria, JSON, XML, INI y variables de entorno
Configuraciones acopladas al sistema de configuraciones	Configuraciones desacopladas al sistema de configuraciones
Sin soporte para inyección de dependencias	Optimizado para inyección de dependencias

Configuraciones por defecto

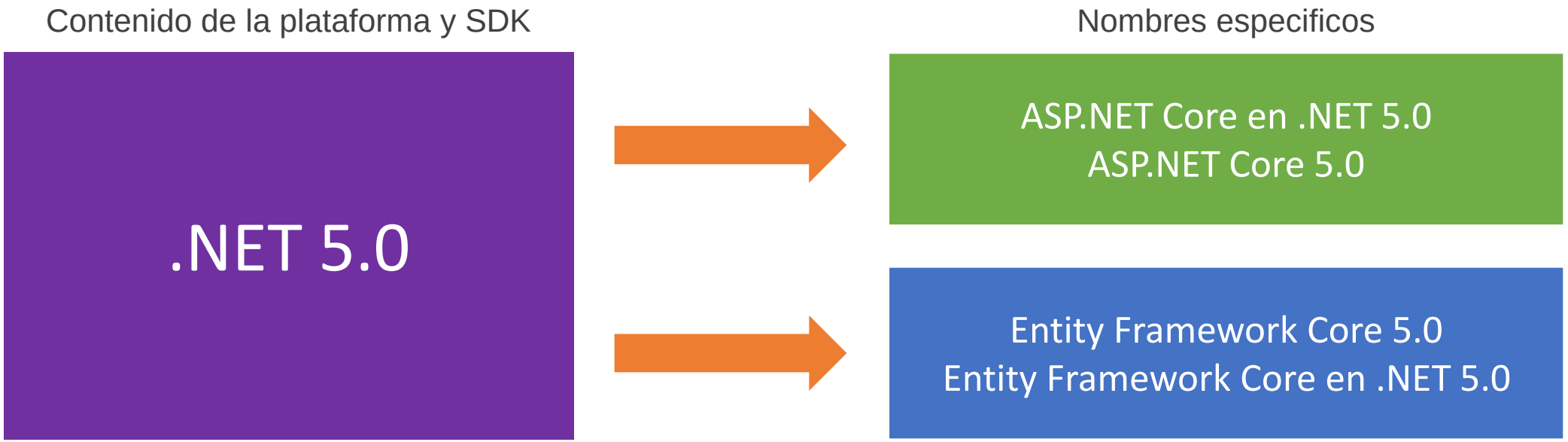


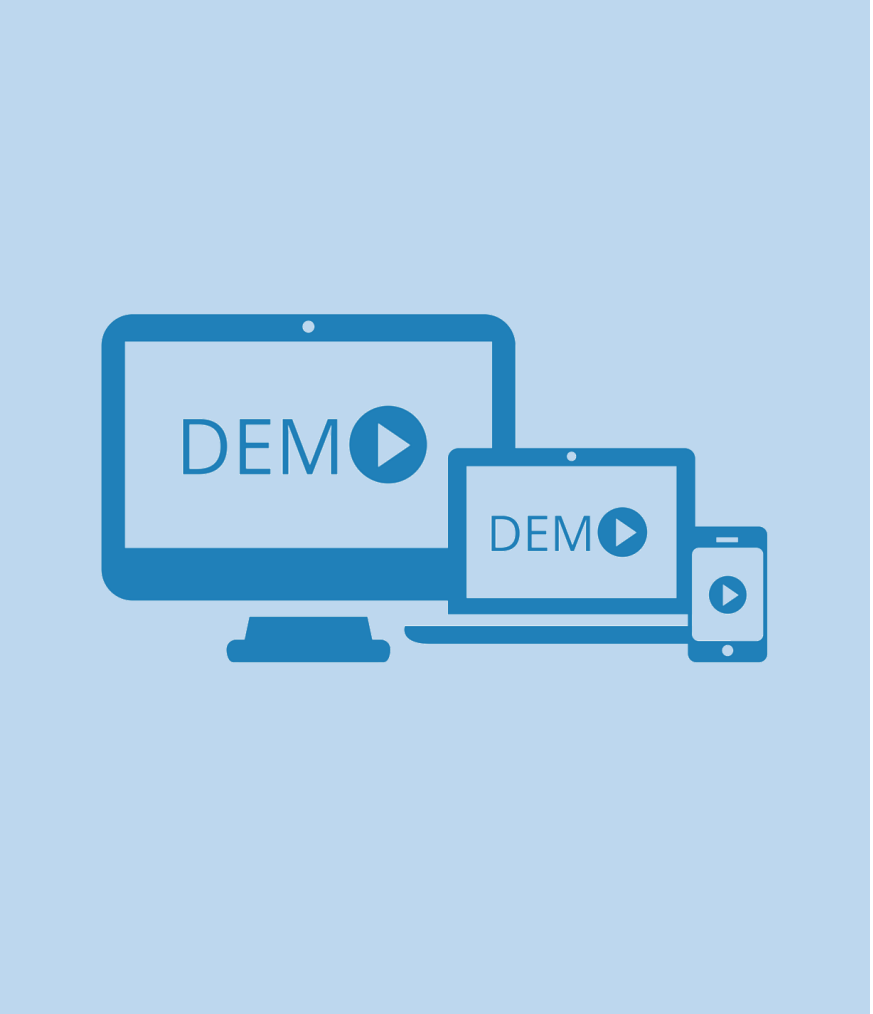
Contenido de ASP.NET Core 5



ASP.NET Core Web Application (Web Apps)

Consideraciones de nomenclatura





ASP.NET Core 5 Web Application (Web Apps)

ASP.NET Core Web Application (Web Apps)

Razor

Razor es una sintaxis de marcado para insertar código basado en servidor en páginas web.

La sintaxis de Razor combina marcado de Razor, C# y HTML.

Los archivos que contienen sintaxis de Razor suelen tener la extensión de archivo .cshtml.

Razor también se encuentra en los archivos de los componentes de Razor (.razor).

Expresiones de Razor implícitas

Las expresiones de Razor implícitas comienzan por @, seguido de código C#.

```
<p>@DateTime.Now</p>
<p>@DateTime.IsLeapYear(2016)</p>
```

Con la excepción de la palabra clave de C# await, las expresiones implícitas no deben contener espacios. Si la instrucción de C# tiene un final claro, se pueden entremezclar espacios

```
<p>@await DoSomething("hello", "world")</p>
```

Expresiones de Razor explícitas

Las expresiones explícitas de Razor constan de un símbolo @ y paréntesis de apertura y de cierre. Para representar la hora de la semana pasada, se usaría el siguiente marcado de Razor

```
<p>Last week this time: @(DateTime.Now - TimeSpan.FromDays(7))</p>
```

Se pueden usar expresiones explícitas para concatenar texto con un resultado de la expresión

```
@{
    var joe = new Person("Joe", 33);
}

<p>Age@(joe.Age)</p>
```

Bloques de código

Los bloques de código de Razor comienzan por @ y se insertan entre {}. A diferencia de las expresiones, el código de C# dentro de los bloques de código no se representa. Las expresiones y los bloques de código de una vista comparten el mismo ámbito y se definen en orden

```
@{
    var quote = "The future depends on what you do today. - Mahatma Gandhi";
}

<p>@quote</p>

@{
    quote = "Hate cannot drive out hate, only love can do that. - Martin Luther King, Jr
}

<p>@quote</p>
```

Bloques de código

En los bloques de código, declare las funciones locales con marcado para utilizarlas como métodos en la creación de plantillas

```
@{
    void RenderName(string name)
    {
        <p>Name: <strong>@name</strong></p>
    }

    RenderName("Mahatma Gandhi");
    RenderName("Martin Luther King, Jr.");
}
```

Estructuras de Control

Condicionales @if, else if, else y @switch

@if controla cuándo se ejecuta el código

```
@if (value % 2 == 0)
{
    <p>The value was even.</p>
}
```

Estructuras de Control

Condicionales @if, else if, else y @switch

else y else if no necesitan el símbolo @:

```
@if (value % 2 == 0)
{
    <p>The value was even.</p>
}
else if (value >= 1337)
{
    <p>The value is large.</p>
}
else
{
    <p>The value is odd and small.</p>
}
```

Estructuras de Control

Condicionales @if, else if, else y @switch

En el siguiente marcado se muestra cómo usar una instrucción switch

```
@switch (value)
{
    case 1:
        <p>The value is 1!</p>
        break;
    case 1337:
        <p>Your number is 1337!</p>
        break;
    default:
        <p>Your number wasn't 1 or 1337.</p>
        break;
}
```

Estructuras de Control

Bucles @for, @foreach, @while y @do while

@for

```
@for (var i = 0; i < people.Length; i++)
{
    var person = people[i];
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>
}
```

Estructuras de Control

Bucles @for, @foreach, @while y @do while

@foreach

```
@foreach (var person in people)
{
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>
}
```

Estructuras de Control

Bucles @for, @foreach, @while y @do while

@while

```
@{ var i = 0; }
@while (i < people.Length)
{
    var person = people[i];
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>

    i++;
}
```

Estructuras de Control

Bucles @for, @foreach, @while y @do while

@do while

```
@{ var i = 0; }
@do
{
    var person = people[i];
    <p>Name: @person.Name</p>
    <p>Age: @person.Age</p>

    i++;
} while (i < people.Length);
```

Directivas

Las directivas de Razor se representan en las expresiones implícitas con palabras clave reservadas seguidas del símbolo @. Normalmente, una directiva cambia la forma en que una vista se analiza, o bien habilita una funcionalidad diferente.

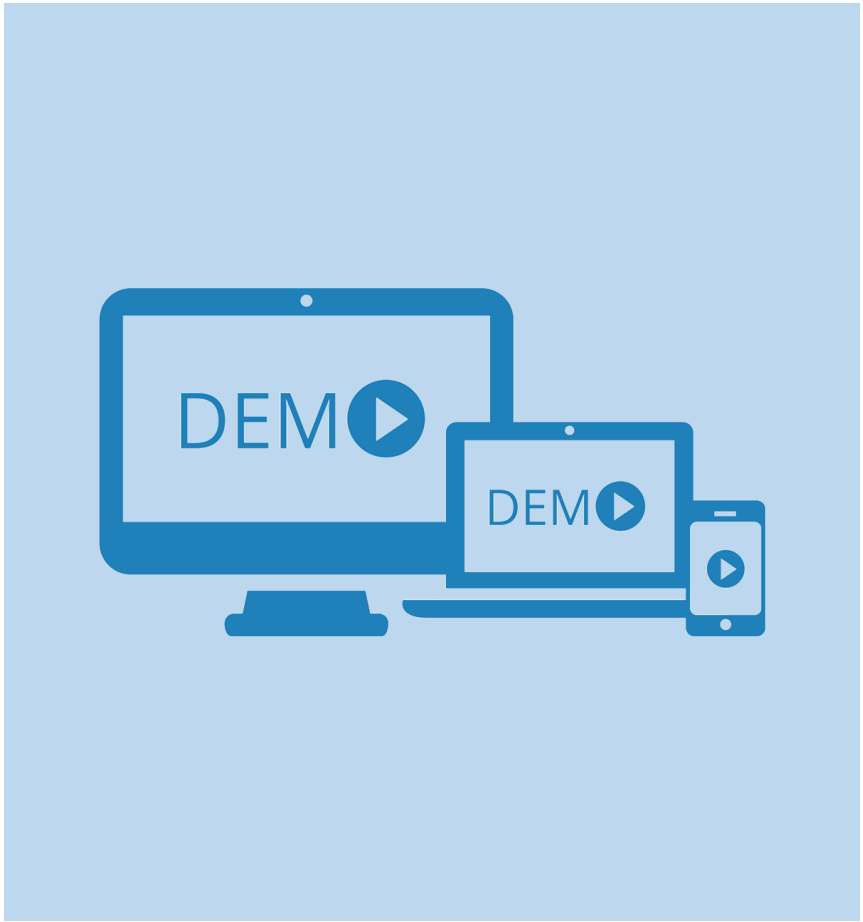
Conocer el modo en que Razor genera el código de una vista hace que sea más fácil comprender cómo funcionan las directivas.

Directivas

@model

Este escenario solo se aplica a las vistas de MVC y Razor Pages (.cshtml).
La directiva @model especifica el tipo del modelo que se pasa a una vista o página

```
@model TypeNameOfModel
```



Razor Pages (expresiones, encoding, bloques, estructuras de control y directivas).

Controlador (Controller)

Los controladores se usan para definir y agrupar un conjunto de acciones. Una acción (o método de acción) es un método en un controlador que controla las solicitudes. Los controladores agrupan lógicamente acciones similares. Esta agregación de acciones permite aplicar de forma colectiva conjuntos comunes de reglas, como el enrutamiento, el almacenamiento en caché y la autorización. Las solicitudes se asignan a acciones mediante el enrutamiento.

Controlador (Controller)

Por convención, las clases de controlador:

- Residen en la carpeta Controllers del nivel de raíz del proyecto.
- Heredan de Microsoft.AspNetCore.Mvc.Controller.

Un controlador es una clase instanciable en la que se cumple al menos una de las siguientes condiciones:

- El nombre de clase tiene el sufijo Controller.
- La clase se hereda de una clase cuyo nombre tiene el sufijo Controller.
- El atributo [Controller] se aplica a la clase.

Controller (Methods: HttpGet, HttpPost y ActionResult: IActionResult y Task<IActionResult>)

Vistas (View)

En el patrón de controlador de vista de modelos (MVC), la vista se encarga de la presentación de los datos y de la interacción del usuario.

Una vista es una plantilla HTML con marcado de Razor incrustado.

El marcado de Razor es código que interactúa con el formato HTML para generar una página web que se envía al cliente.

Vistas (View)

Ventajas:

Las vistas separan el marcado de la interfaz de usuario de otras partes de la aplicación con el fin de ayudar a establecer una separación de responsabilidades dentro de una aplicación MVC.

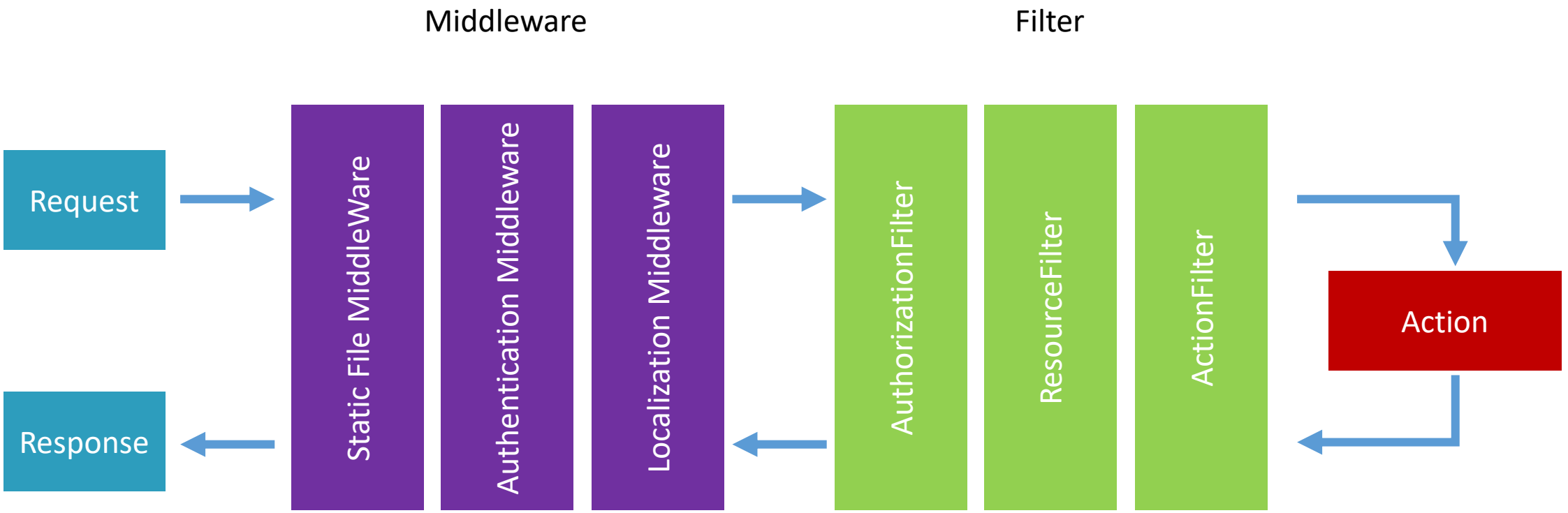
- La aplicación es más fácil de mantener, ya que está mejor organizada. Las vistas generalmente se agrupan por característica de la aplicación. Esto facilita la búsqueda de vistas relacionadas cuando se trabaja en una característica.
- Las partes de la aplicación están acopladas de forma ligera. Las vistas de la aplicación se pueden compilar y actualizar por separado de los componentes de la lógica de negocios y el acceso a datos. Las vistas de la aplicación se pueden modificar sin necesidad de tener que actualizar otras partes de la aplicación.
- Es más fácil probar los elementos de la interfaz de usuario de la aplicación, ya que las vistas son unidades independientes.
- Dada la mejora en la organización, es menos probable que se repitan secciones de la interfaz de usuario de forma accidental.

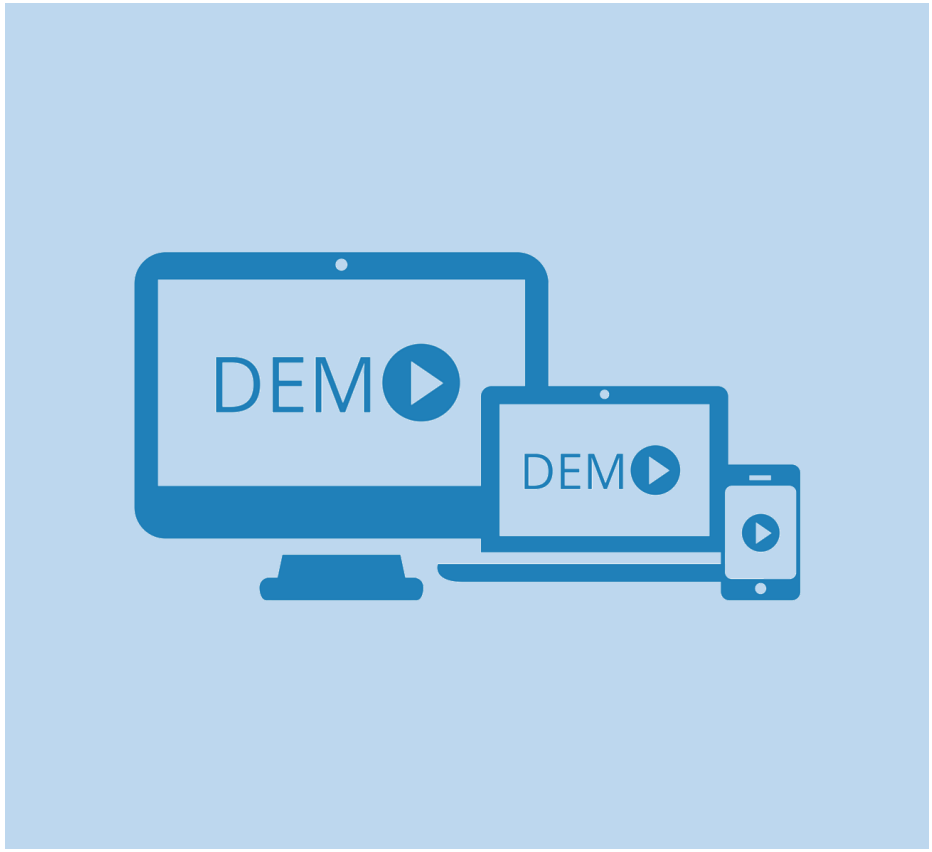
Enrutamiento (Rounting)

ASP.NET Core MVC utiliza el middleware de enrutamiento para buscar las direcciones URL de las solicitudes entrantes y asignarlas a acciones. Las rutas se definen en el código de inicio o los atributos. Las rutas describen cómo se deben asociar las rutas de dirección URL a las acciones. Las rutas también se usan para generar direcciones URL (para vínculos) enviadas en las respuestas.

Las acciones se enrutan bien mediante convención o bien mediante atributos. Colocar una ruta en el controlador o la acción hace que se enrute mediante atributos. Consulte Enrutamiento mixto para obtener más información.

Middleware


Middleware
Filter
Request
Response
Static File MiddleWare
Authentication Middleware
Localization Middleware
AuthorizationFilter
ResourceFilter
ActionFilter
Action



View, Routing e Integración CRUD

View, Routing e Integración CRUD

GRACIAS

POR SU PREFERENCIA

GALAXY
TRAINING