Arquitectura de Aplicaciones

- Crear un proyecto MVC en ASP
- Anatomía de un proyecto MVC
- Recordando conceptos
- Request http
- Vistas en asp .net
- El motor Razor
- Layout ASP

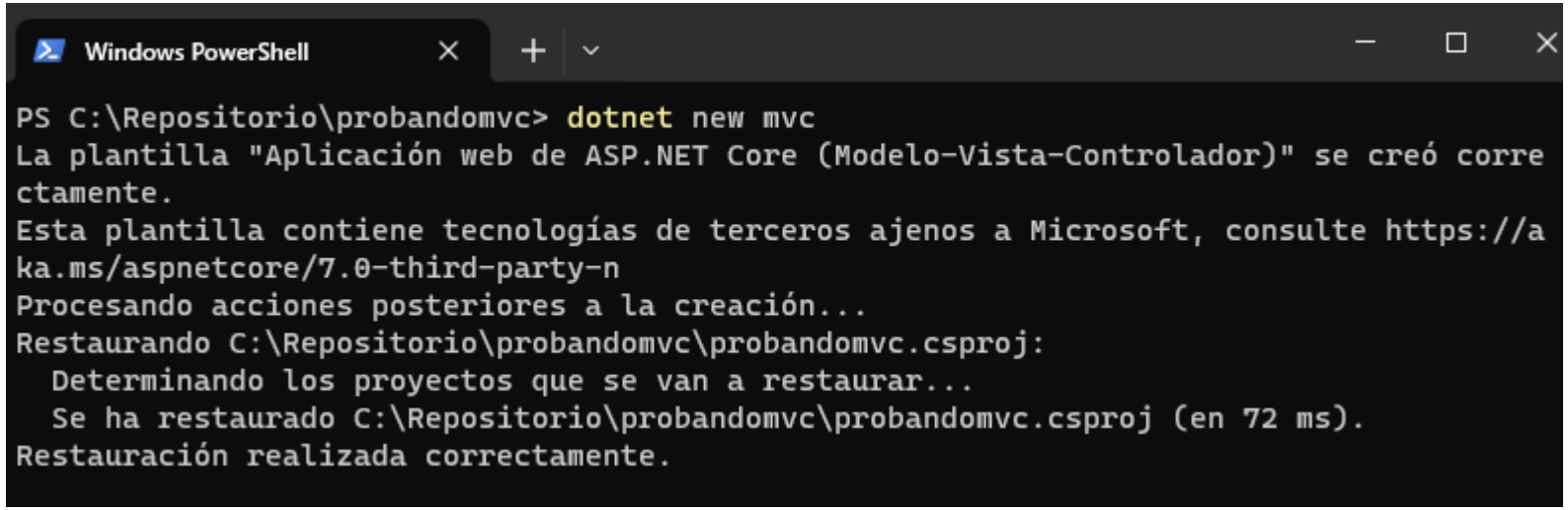
Front End Vs Back End

Web MVC con ASP Net Core

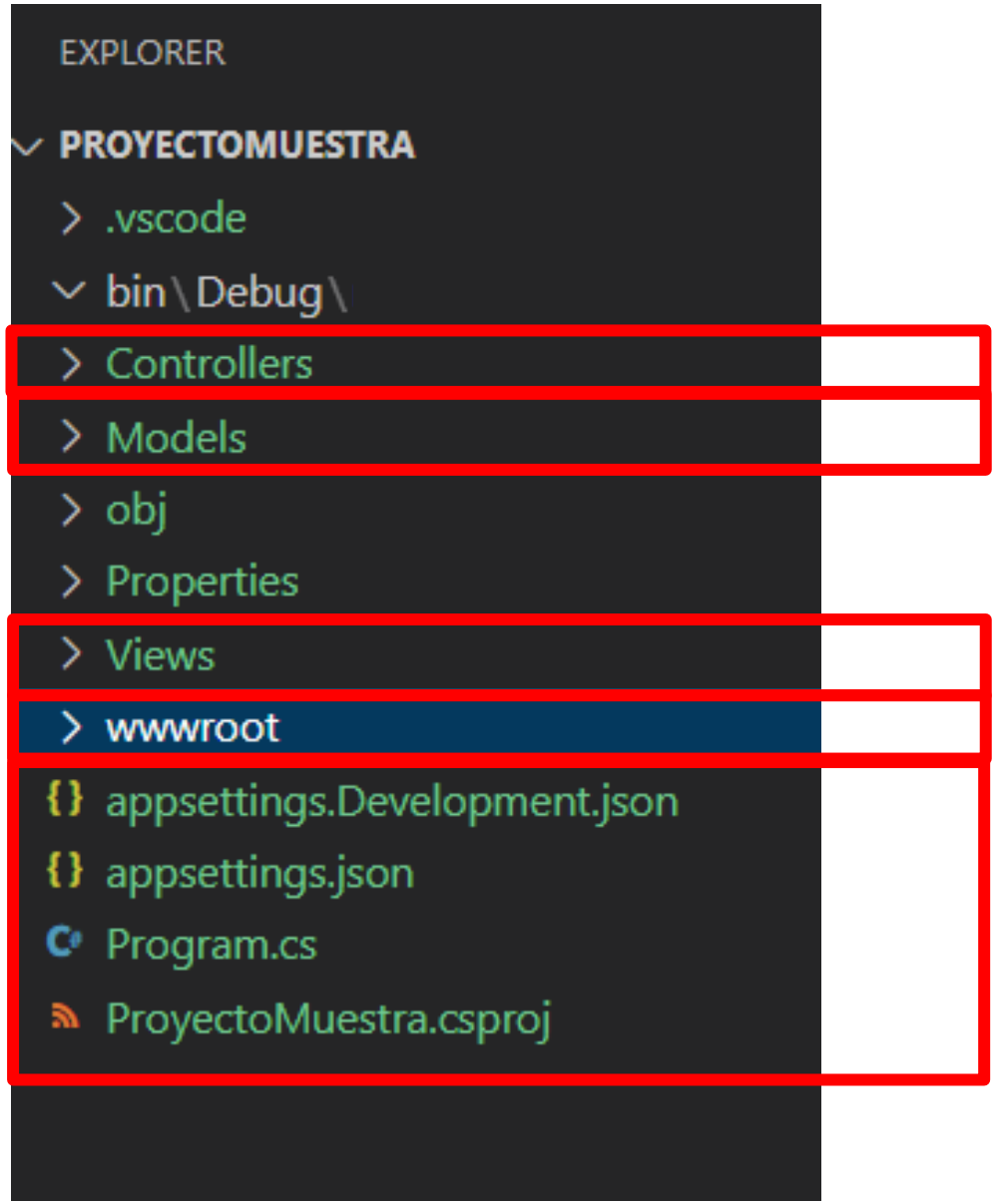
Crear una nuevo proyecto webAPI

Comandos para crear y abrir un proyecto WebAPi basado en controladores:

- dotnet new mvc -n MiWebApp ← Crea un nuevo proyecto MVC
- code -r MiWebApp ← Abre Vs code desde la terminal con el proyecto



Anatomía de un proyecto MVC 6.0



→ Controladores

→ Modelos

→ Plantillas de Vistas

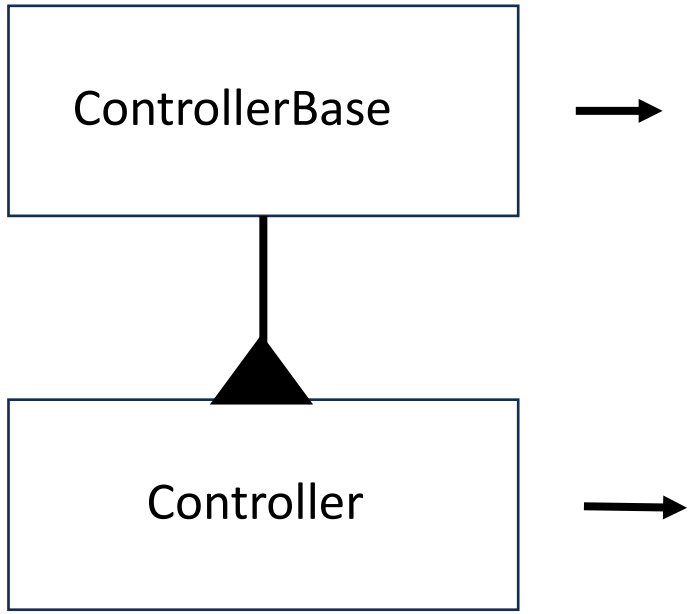
→ Recursos Front-End (Cliente)

→ Clases y archivos de configuración del proyecto

Recordando

Clase Controller

Una API web basada en controladores consta de una o más clases de controladores que derivan de **ControllerBase**.



Clase de la que debemos derivar para implementar un controlador en web API. El response será un **Json**

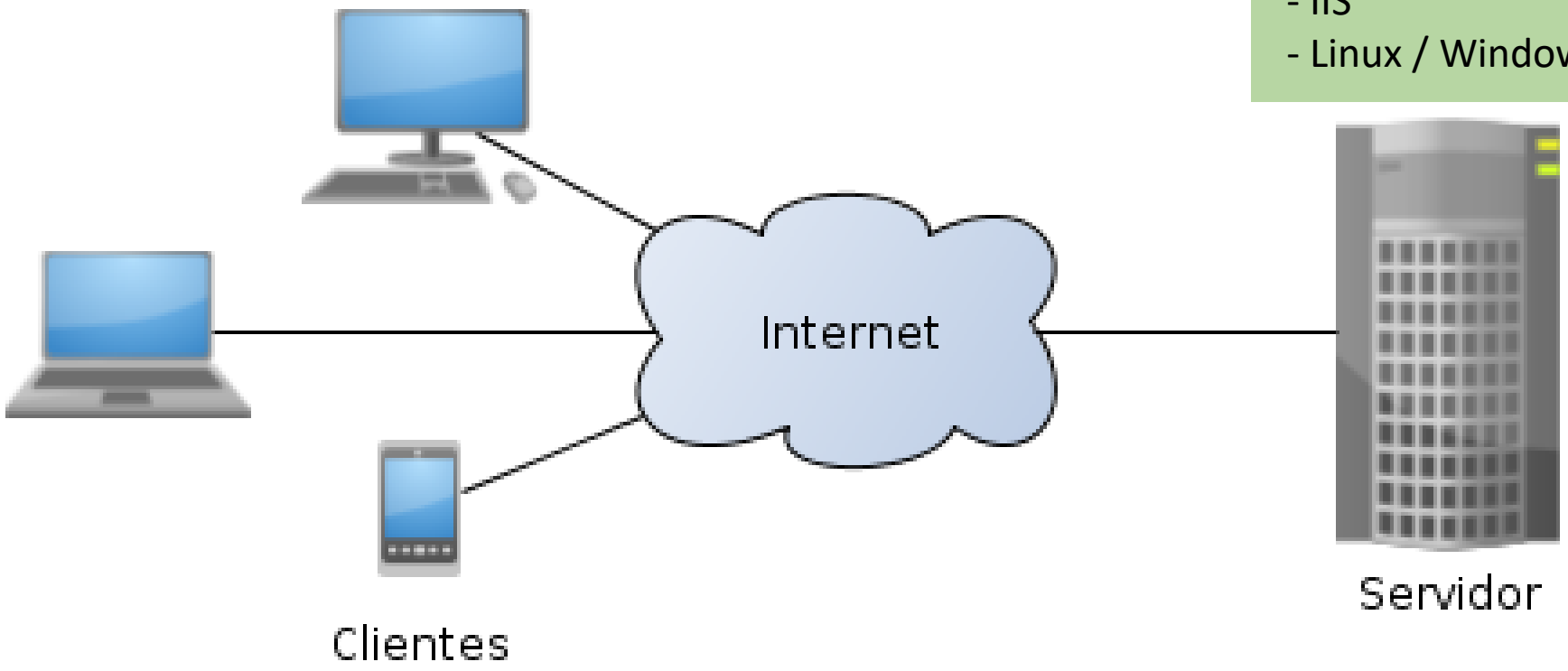
Clase de la que debemos derivar para implementar un controlador en páginas WEB. Agrega soportes para manejar **vistas**.

Notas:
Si el mismo controlador debe admitir vistas y API web, entonces debe derivar de Controller.

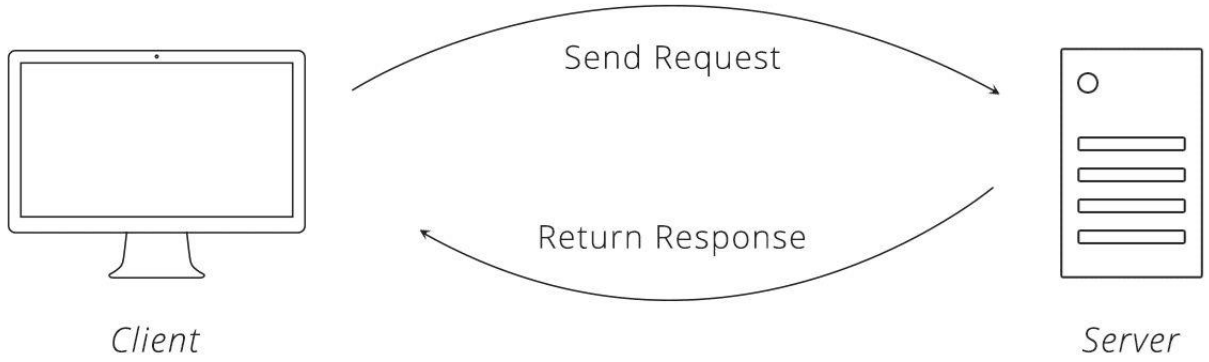
Recordando

Solicitud a un servidor

- ASP
- NET
- IIS
- Linux / Windows S.



Recordando

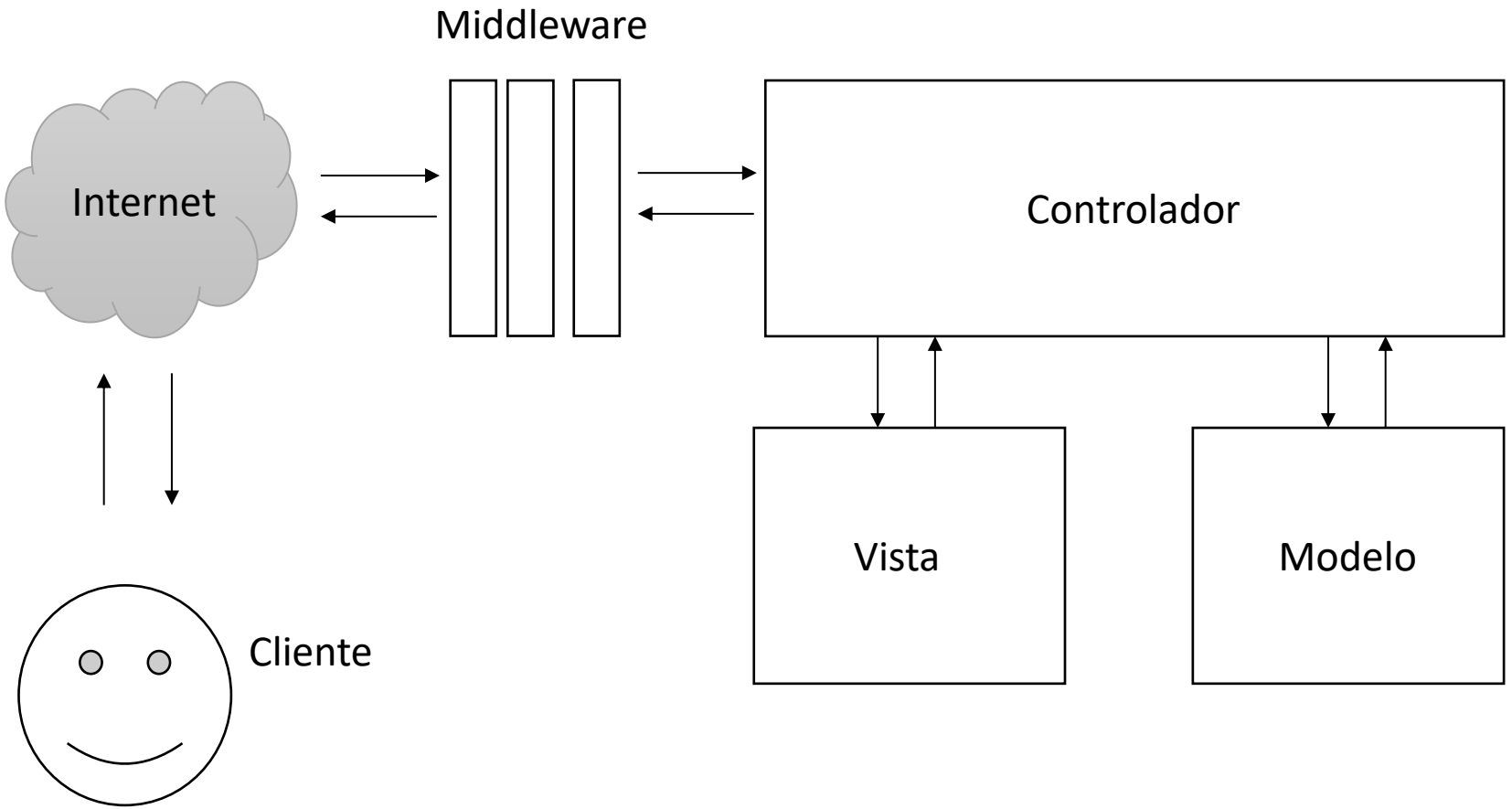


Hyper-Text Transfer Protocol, major conocido como: HTTP

La comunicación en HTTP se centra en un concepto llamado ciclo de solicitud-respuesta. El cliente envía al servidor una solicitud para hacer algo. El servidor, a su vez, envía al cliente una respuesta diciendo si el servidor puede o no hacer lo que el cliente pidió.

Recordando

Funcionamiento – Request / Response



Recordando

La ruta establecida por defecto para los controladores es conocida como *conventional route*. Es llamado así porque es la convención para rutas URL:

Por defecto la ruta se puede mapear desde el archivo program.cs en el bloque de código siguiente:

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

- El primer parámetro del path, {controller=Home}, Mapea el controlador.
- El segundo parámetro de path, {action=Index}, mapea el nombre de la acción vinculada.
- El tercer parámetro, {id?} es usado como parámetro opcional id. El ? en {id?} hace que sea opcional. id es usado para mapear al model de entidad.

Request Http

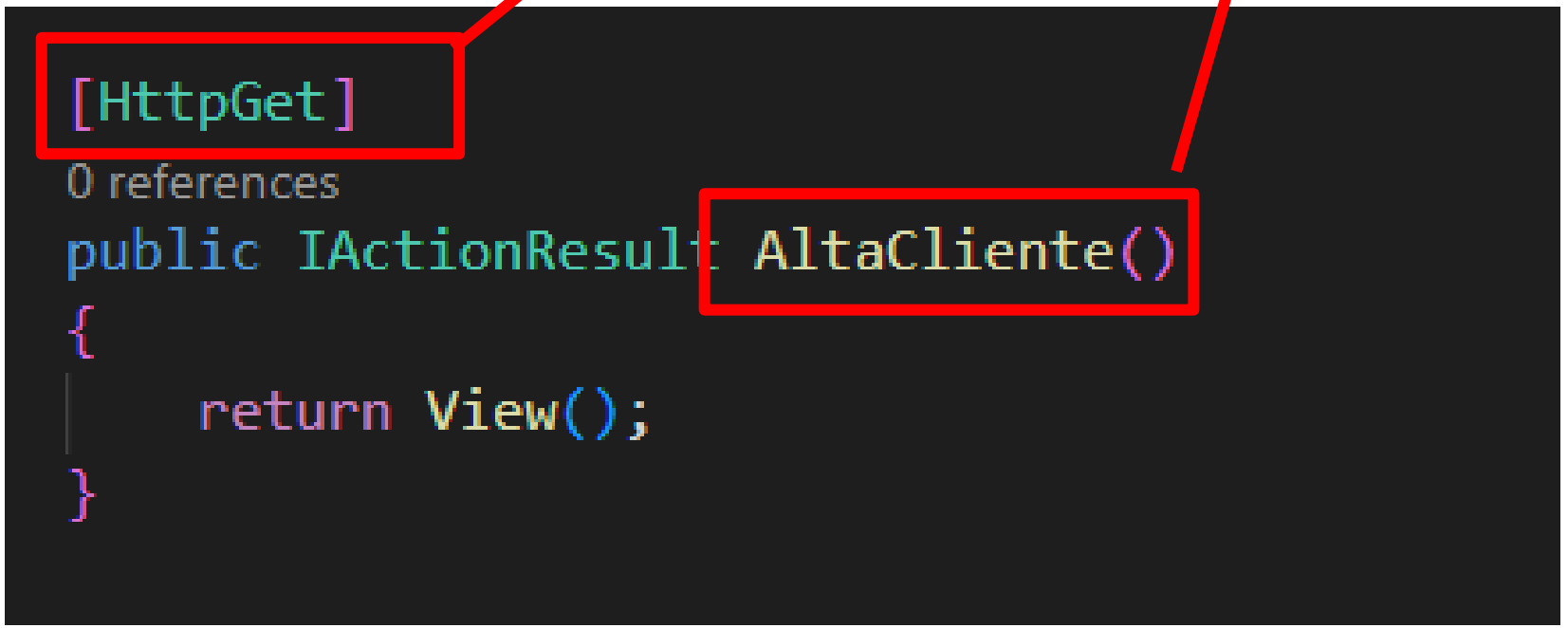
Método Get sin parámetros

Controlador

https://localhost:{PORT}/Cliente/AltaCliente

Atributo Get

Acción



Request Http

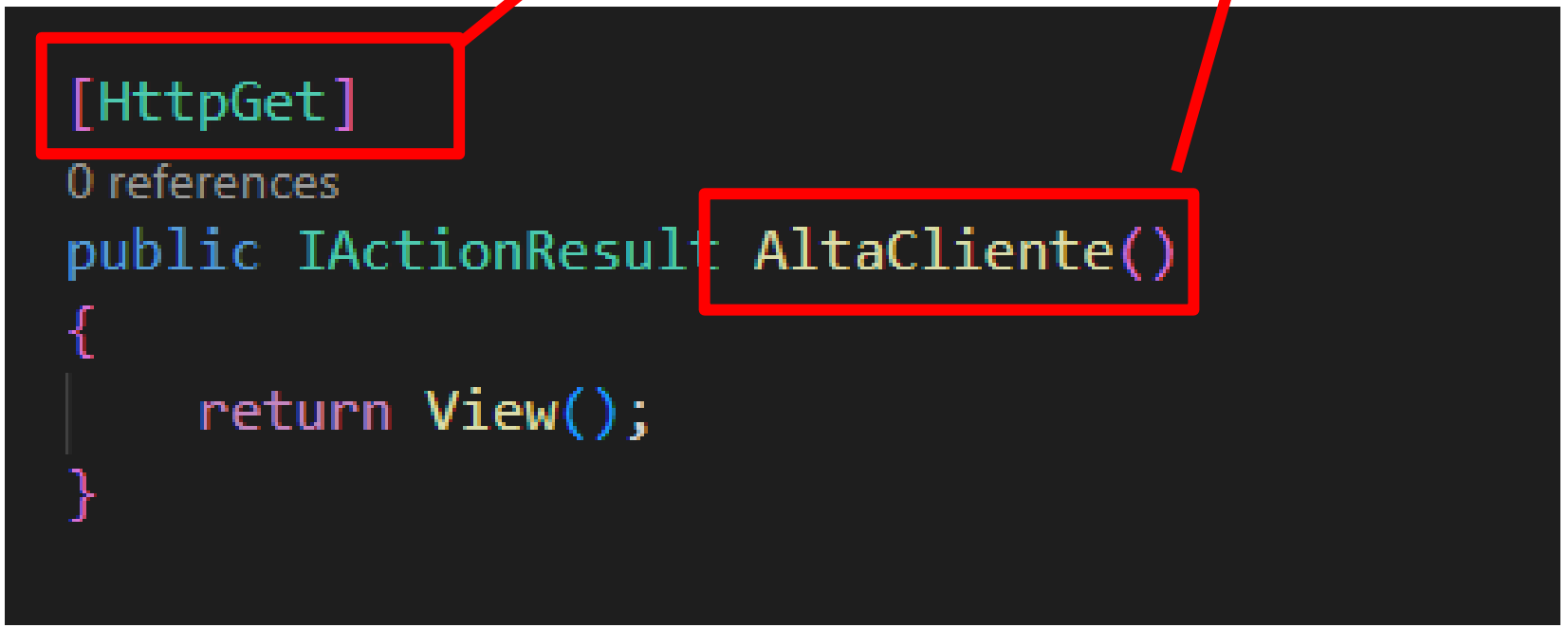
Método Get sin parámetros

Controlador

https://localhost:{PORT}/Cliente/AltaCliente

Atributo Get

Acción



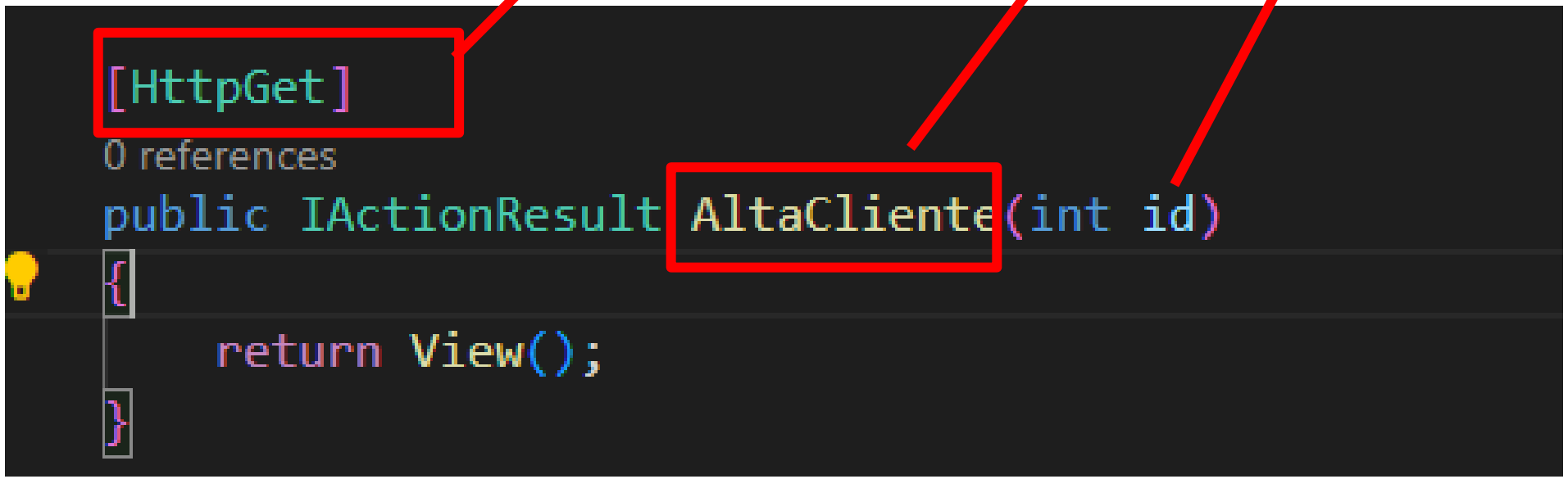
Request Http

Método Get con un parámetro

Controlador

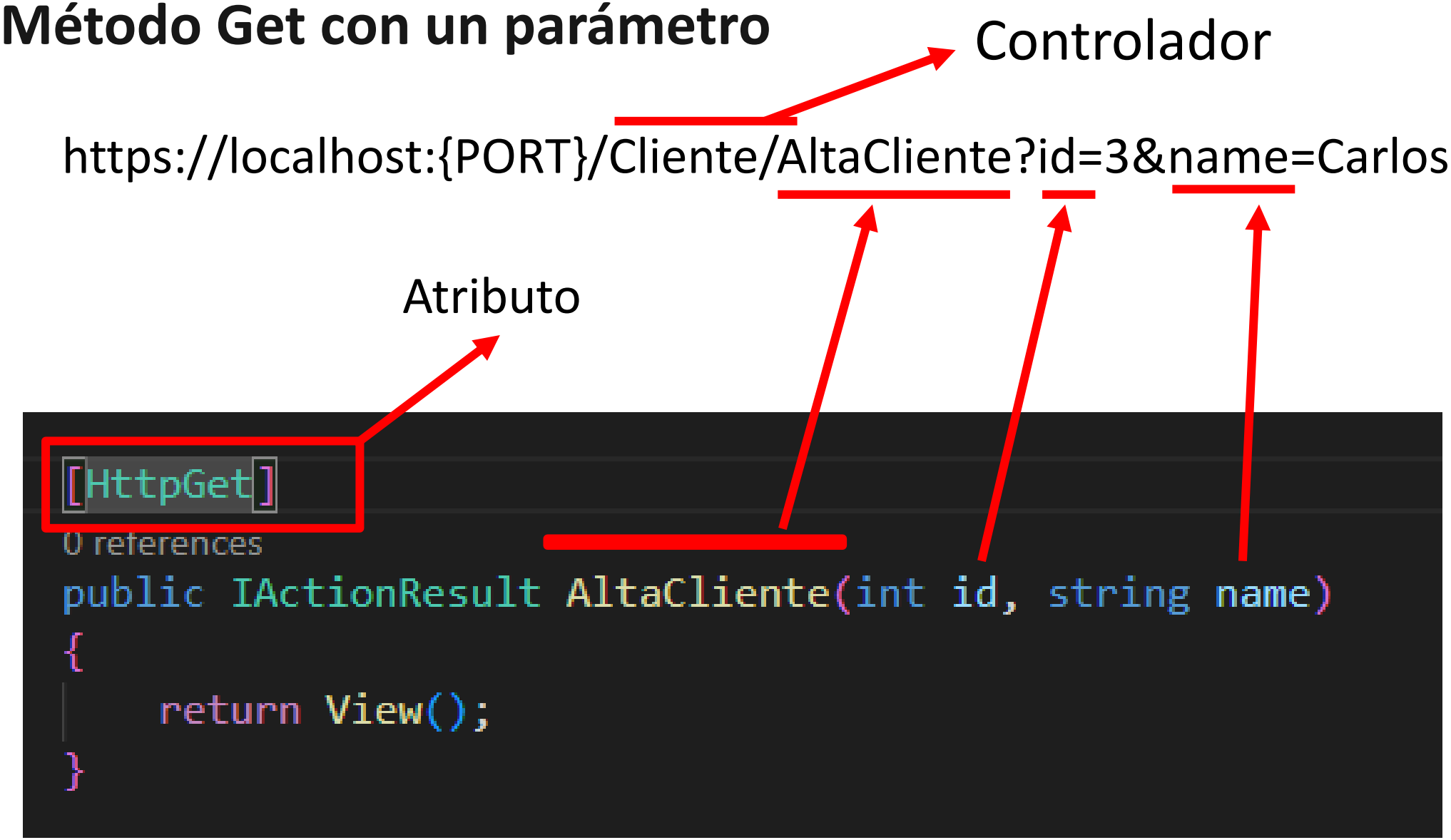
https://localhost:{PORT}/Cliente/AltaCliente/3

Atributo Get



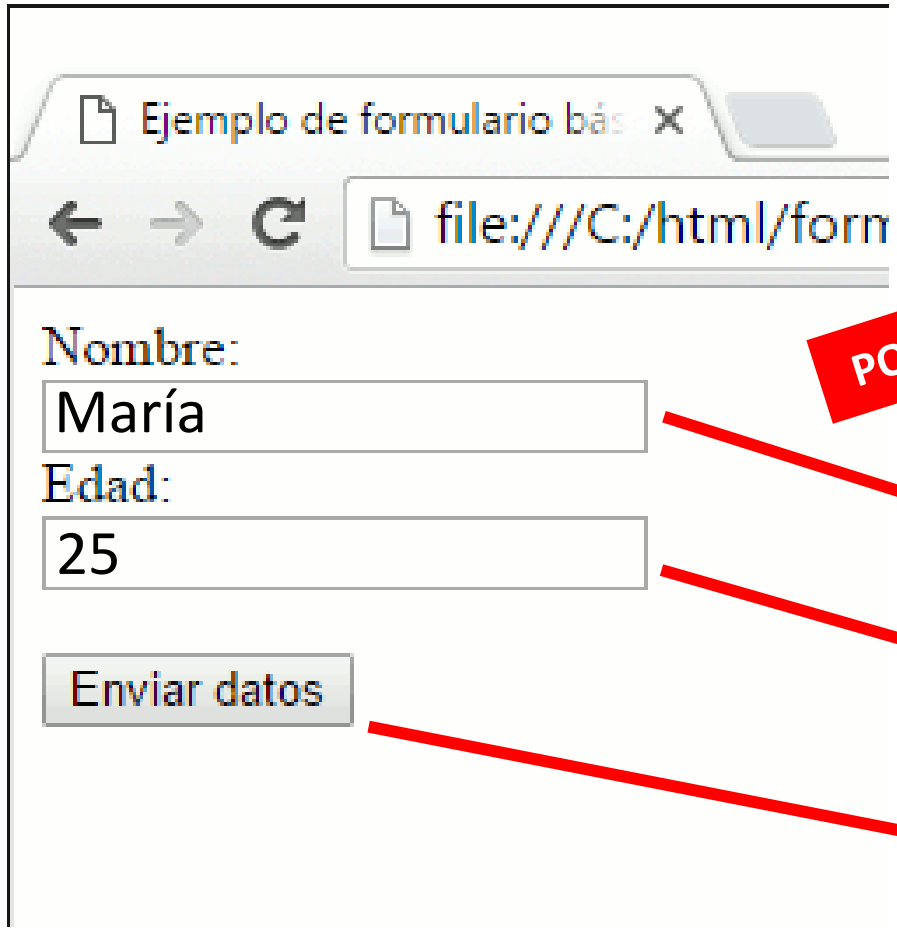
Request Http

Método Get con un parámetro



Request Http

Método Post con un formulario



POST

```
[HttpPost]
0 references
public IActionResult AltaCliente(Cliente cliente)
{
    return View();
}
```

Cliente.Nombre = “María”

Cliente.Edad = 25

Submit

(envía los datos en el cuerpo del request)

Source Img: https://www.abrirllave.com/html/formularios.php

Request Http

Método Post con un formulario

```
Class Cliente
{
   Nombre = “María”
   Edad = 25
}
```

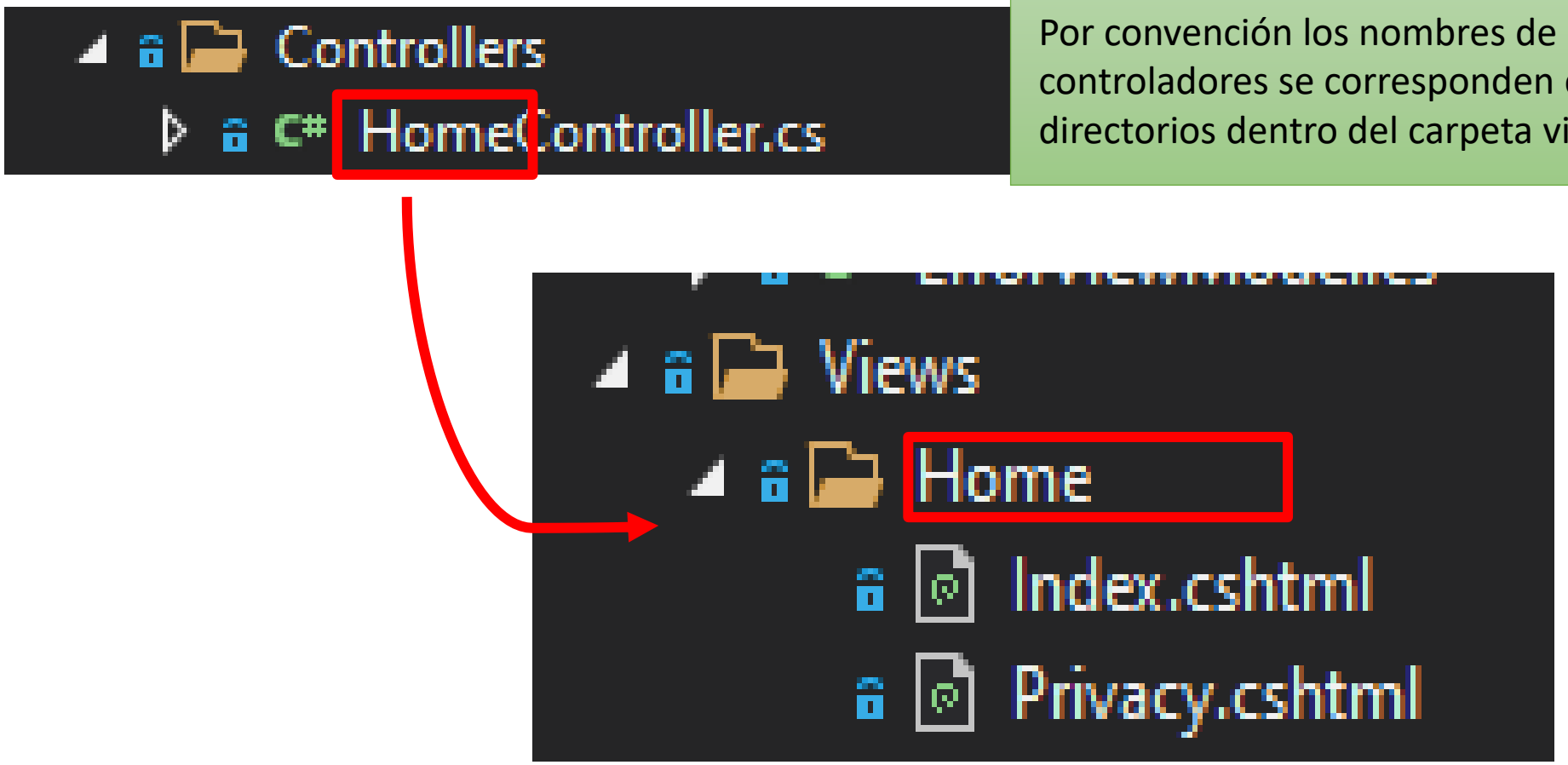
Atributo Post

```
[HttpPost]
0 references
public IActionResult AltaCliente(Cliente cliente)
{
    return View();
}
```

Source Img: https://www.abrirllave.com/html/formularios.php

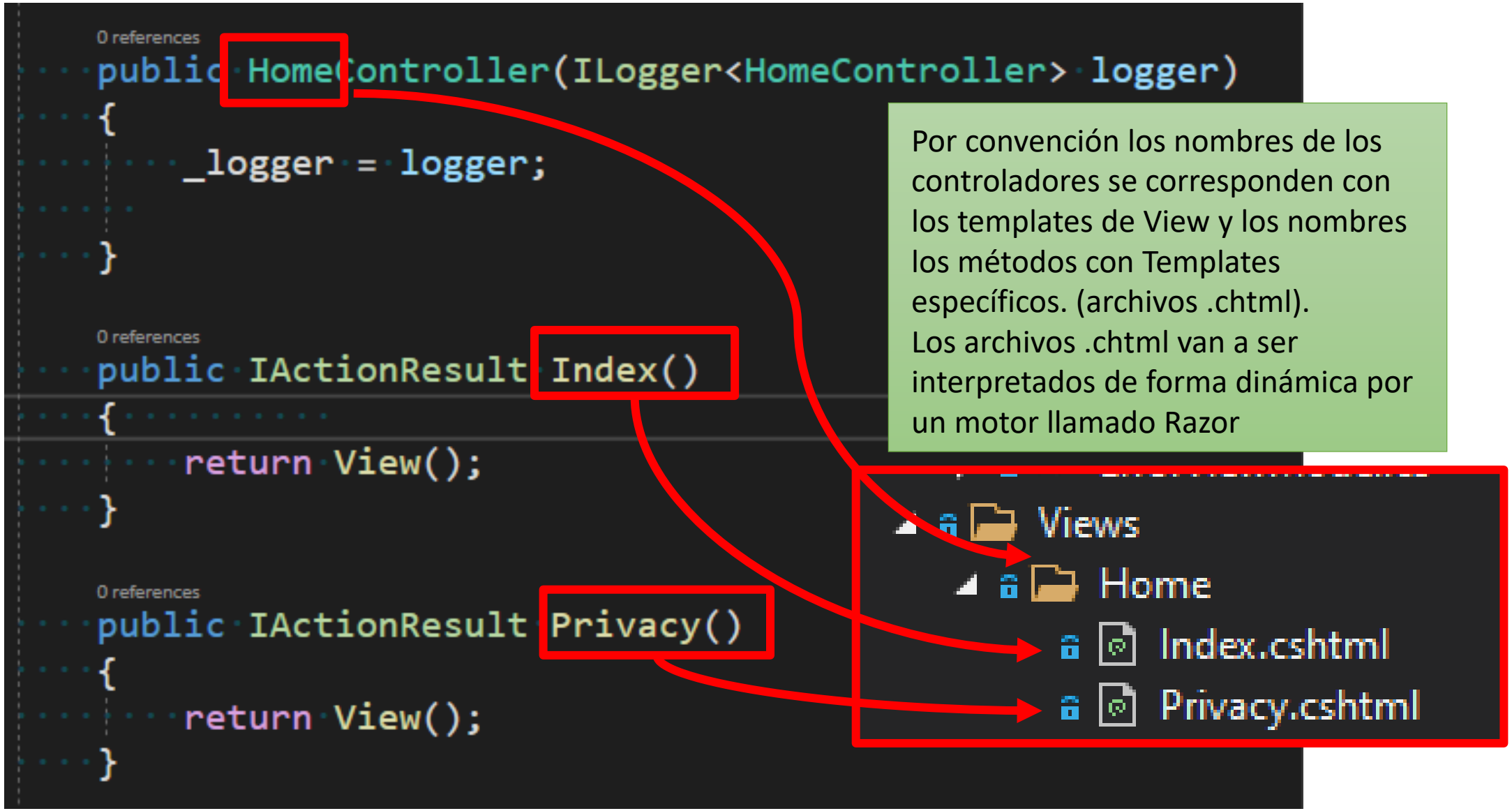
Vistas ASP-NET

Vistas en ASP



Por convención los nombres de los controladores se corresponden con los directorios dentro del carpeta view.

Vistas en ASP



Por convención los nombres de los controladores se corresponden con los templates de View y los nombres los métodos con Templates específicos. (archivos .chtml).
Los archivos .chtml van a ser interpretados de forma dinámica por un motor llamado Razor

Vistas en ASP

```
0 references
public HomeController(ILogger<HomeController> logger)
{
    _logger = logger;

}

0 references
public IActionResult Index()
{
    return View();
}

0 references
public IActionResult Privacy()
{
    return View();
}
```

Los métodos de un controlador pueden retornar vistas haciendo uso del método View. Y pueden pasar objetos como parámetro en este método para que la vista pueda usar esta información y modificarse dinámicamente.

Vistas en ASP

Un controlador puede devolver varios tipos de objetos como resultado de una acción. Algunos de los tipos de objetos que un controlador puede devolver son **ViewResult, PartialViewResult, JsonResult, ContentResult, FileResult y RedirectResult**. Todos estos tipos de objetos heredan de la clase abstracta ActionResult.

- **ViewResult:** Este objeto se utiliza para devolver una vista al explorador. Una vista es una plantilla que se utiliza para generar HTML que se envía al explorador.
- **PartialViewResult:** Se utiliza para devolver una vista parcial al explorador. Una vista parcial es una vista que se utiliza para generar una parte de una página web.
- **JsonResult:** se utiliza para devolver datos en formato JSON al explorador
- **ContentResult:** Se utiliza para devolver contenido sin formato al explorador. Por ejemplo un archivo de texto.
- **FileResult:** Se utiliza para devolver un archivo al explorador.
- **RedirectResult:** Este objeto se utiliza para redirigir al explorador a otra página web.

Vistas en ASP

Los métodos auxiliares en el controlador (por ejemplo, Json(), Content(), View(), …) devuelven diferentes clases concretas que heredan de **ActionResult**, incluyendo JsonResult. Al declarar los métodos de acción para que devuelven ActionResult, damos libertad de devolver cualquier clase de resultado concreto.

```
0 references
public HomeController(ILogger<HomeController> logger)
{
    _logger = logger;

}

0 references
public IActionResult Index()
{
    return View();
}

0 references
public IActionResult Privacy()
{
    return View();
}
```

Los métodos de un controlador pueden retornar vistas haciendo uso del método View. Y pueden pasar objetos como parámetro en este método para que la vista pueda usar esta información y modificarse dinámicamente.

Vistas en ASP

Formas de utilizar el método Helper “View()“

Retornando una vista explícitamente:

```
return View(“Productos"); // va a la vista productos
```

Pasando un modelo a las vistas como parámetro:

```
Producto MiProducto= new Producto();
MiProducto.Nombre = “Galletas”;
MiProducto.Precio = 13;

return View(Producto); // pasa un objeto de tipo Producto
```

Pasando una vistas y un objeto como parámetro:

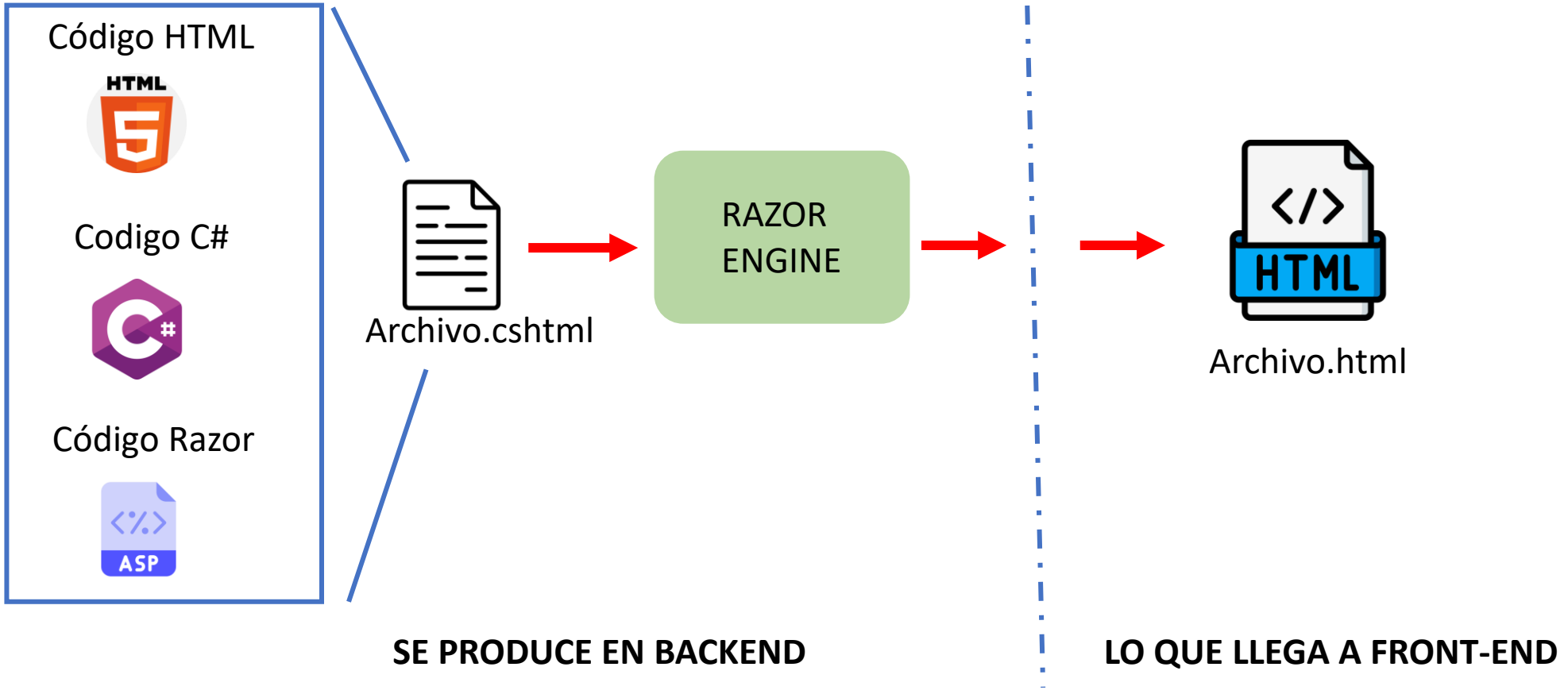
```
Producto MiProducto= new Producto();
MiProducto.Nombre = “Galletas”;
MiProducto.Precio = 13;

return View(“Productos”, Producto); // pasa un objeto de tipo Producto a la vista Productos
```

Razor

¿Qué es Razor Engine?

Para construir vistas dinámicas en ASP hacemos uso de un motor de render llamado **RAZOR**

Razor es una sintaxis de marcado para insertar código basado en .NET en páginas web. La sintaxis de Razor consta de marcado de Razor, C# y HTML. Los archivos que contienen Razor suelen tener la extensión de archivo .cshtml.



Razor

Enviando un modelo a una vista

```
public IActionResult MostrarProducto()
{
        Producto MiProducto= new Producto();
        MiProducto.Nombre = “Galletas”;
        MiProducto.Precio = 13;
        return View(MiProducto);
}
```

Código en el controlador (controller)

Método Mostrar Producto declarado en algún controlador de un proyecto

```
@model Producto
```

Se declara en la cabecera del archivo .chtml

@model es una directiva Razor que se utiliza para especificar el modelo que se utilizará en una **vista** en particular.

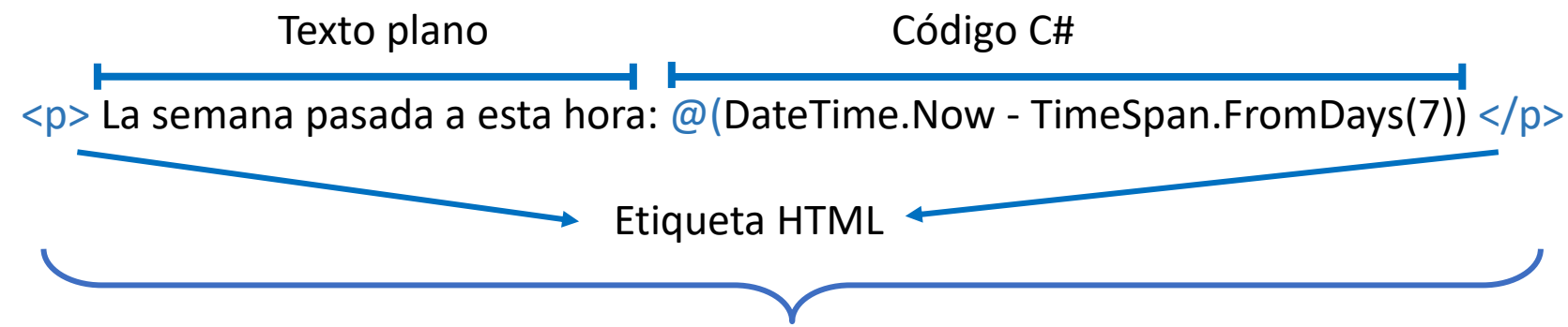
Define el tipo de datos que se espera en la vista.En el ejemplo: la vista espera un objeto de tipo **Producto**

```
<p> @Model.Precio </p>
```

Se utiliza desde el HTML con @Model

Razor

El Engine Razor admite C# y usa el **@** símbolo para pasar de HTML a C#. Razor evalúa las expresiones de C# y las representa en la salida HTML.



Razor engine procesa este código de forma completa y obtiene una salida html

Salida en HTML → La semana pasada a esta hora: 7/7/2016 4:39:52 PM

Vistas

Expresiones explícitas

```
@{
   var carlos = new Empleado(“Carlos", 33);
}

<p>Edad : @(carlos.Edad)</p>
```

Salida en HTML →

```
<p> Edad : 33</p>
```

```
@{
   var Frase = "El odio no puede expulsar al odio, sólo el amor puede hacerlo. - Martin Luther King, Jr.";
}
<p>@Frase</p>
```

Salida en HTML →

```
<p> El odio no puede expulsar al odio, sólo el amor puede hacerlo.
- Martin Luther King, Jr
</p>
```

Razor

Uso de Iteradores en Razor

```
@for (var i = 0; i < empleados.Count(); i++)
{
        var emplado = empleado[i];
        <li>Name: @person.Name</Li>
}
```

Salida en HTML →

```
<ul>
  <li>Pablo</li>
  <li>Carlos</li>
  <li>Juan</li>
</ul>
```

```
@model List<DiaDeSemana>()
<ul>
  @foreach (var dia in Model)
  {
    <li>@dia</li>
  }
</ul>
```

Salida en HTML →

```
<ul>
  <li>Lunes</li>
  <li>Martes</li>
  <li>Miercoles</li>
  <li>Jueves</li>
  <li>Sabado</li>
  <li>Domingo</li>
</ul>
```

Para información oficial del uso de Razor

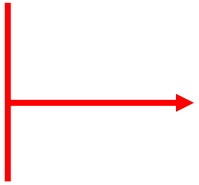
https://docs.microsoft.com/es-es/aspnet/core/mvc/views/razor?view=aspnetcore-3.1

Razor

Ejemplo de un archivo Razor

```
@model List<string>  // Define el modelo para esta vista

<!DOCTYPE html>
<html>
<head>
   <title>Lista de Elementos</title>
</head>
<body>
   <h1>Lista de Elementos</h1>
   <ul>
      @foreach (var elemento in Model)
      {
         <li>@elemento</li>
      }
   </ul>
</body>
</html>
```

Aquí se definen los datos/objetos que llegarán a la vista



Utilizando el objeto Model definido anteriormente

Para información oficial del uso de Razor

https://docs.microsoft.com/es-es/aspnet/core/mvc/views/razor?view=aspnetcore-3.1

Razor

Etiquetas definidas en razor

Son Herramientas que simplifican la generación de código HTML y la interacción con etiquetas HTML en vistas Razor. Se ven como parámetros en etiquetas HTML o incluso en algunos casos como etiquetas nuevas y permiten facilitar tareas de presentación de datos sin perder la estructura visual y de sintaxis de un documento HTML.

Ventajas:

- Aumentan la productividad de desarrollo.
- Reducen errores tipográficos.
- Simplifican la generación de enlaces y URLs.
- Ayudan en la validación de formularios con atributos de validación.
- Mantienen la coherencia en el diseño de páginas web.

Razor

Etiquetas definidas por para anchors

Atributo	Descripción
asp-controller	El nombre del controlador.
asp-action	El nombre del método de acción.

Etiqueta anchor que dirige a un controlador y luego a un método determinado

```
<a asp-controller=“Empleados" asp-action=“GetAll">Empleados</a>
```

Salida en HTML → `<a href="/GetAll">Empleados</a>`

Razor

Etiquetas definidas por para anchors

Atributo	Descripción
asp-controller	El nombre del controlador.
asp-action	El nombre del método de acción.
asp-route	Agrega una ruta en la url para direccionar el recurso

Etiqueta anchor que dirige a un controlador y luego a un método determinado y pasa un id como parámetro

```
<a asp-controller=“Empleados" asp-action=“Editar" asp-route-id="@Model.SpeakerId">
@Model.Nombre</a>
```

Salida en HTML → `<a href="/Empeados/Editar/15">Juan Perez</a>`

Razor

Etiquetas definidas por para anchors

Atributo	Descripción
asp-route-”..”	Agrega una ruta en la url para direccionar el recurso. También se puede parametrizar

Etiqueta anchor que dirige a un controlador y luego a un método determinado y pasa un id como parámetro y especifica un nombre para el parámetro enviado

```
<a asp-controller=“Empleados" asp-action=“Editar" asp-route-Empleadoid="@Model.EmpleadoId">@Model.Nombre</a>
```

Salida en HTML →

```
<a href="/Empeados/Editar?Empleado-Id=15">Juan Perez</a>
```

Razor

Etiquetas definidas por para Formularios

Genera el valor del atributo HTML <FORM>action para una acción de controlador MVC o una ruta con nombre

Genera un token comprobación de solicitudes oculto que impide que se falsifiquen solicitudes entre sitios (cuando se usa con el atributo [ValidateAntiForgeryToken] en el método de acción HTTP Post).

Proporciona el atributo asp-route-<Parameter Name>, donde <Parameter Name> se agrega a los valores de ruta. Los parámetros routeValues de Html.BeginForm y Html.BeginRouteForm proporcionan una funcionalidad similar.

Tiene Html.BeginForm y Html.BeginRouteForm como alternativa del asistente de HTML.

Razor

Etiquetas definidas para Formularios

Podemos usar otros atributos para construir formularios en las vistas

```
<form asp-controller=“Empleados" asp-action=“CrearEmpleado" method="post">
        <!– Elementos Input y Submit del formulario-->
</form>
```

Salida en HTML


```
<form method="post" action="/ Empleados/CrearEmpleado ">
   <!-- Input and Submit elements -->
</form>
```

Razor

Etiquetas definidas para Formularios

El atributo asp-for en ASP.NET Core MVC se utiliza para vincular un elemento HTML a una propiedad específica del modelo. Al hacerlo, especifica el atributo html type basado en el tipo de dato de la propiedad que se vincula.

Etiqueta input utilizando el atributo asp-for

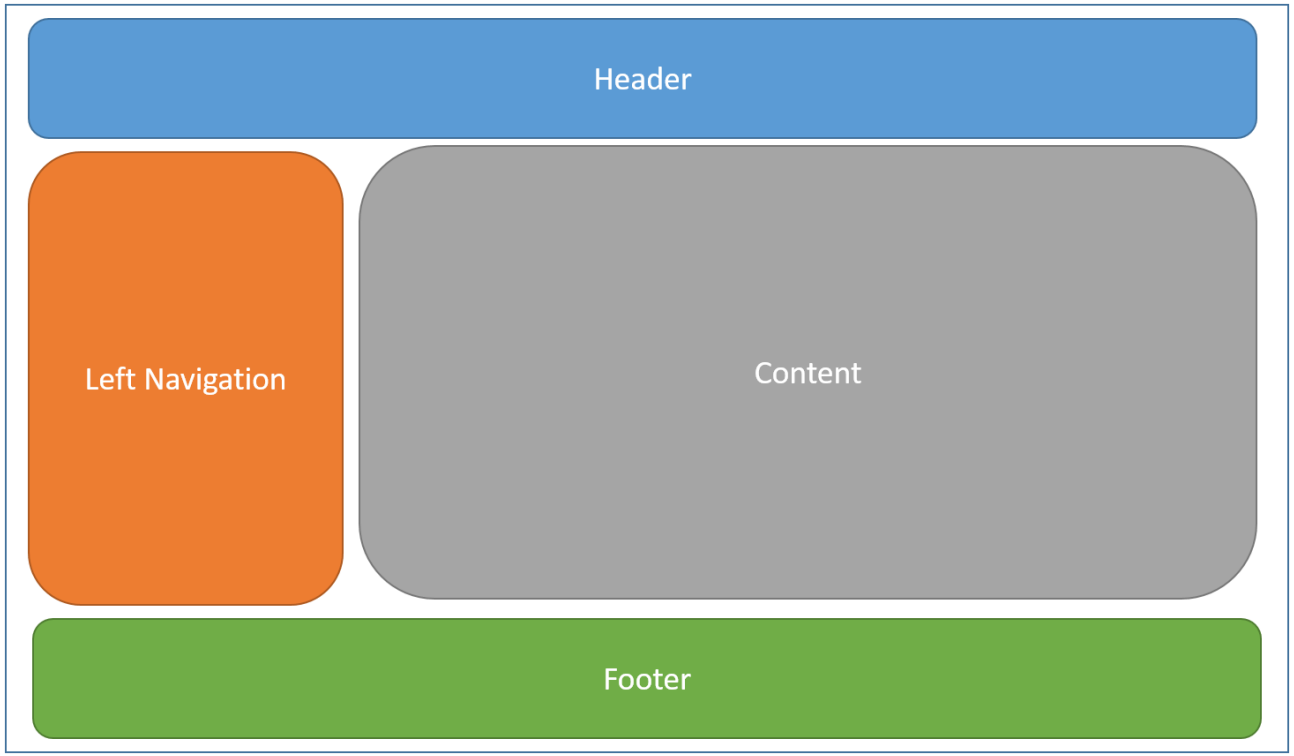
<input asp-for="<Expression Name>">

Tipo de .NET	**Tipo de salida HTML**
Bool	type="checkbox"
String	type="text"
DateTime	type="datetime-local"
Byte	type="number"
Int	type="number"
Single, Double	type="number"

View Layout de MCV Template

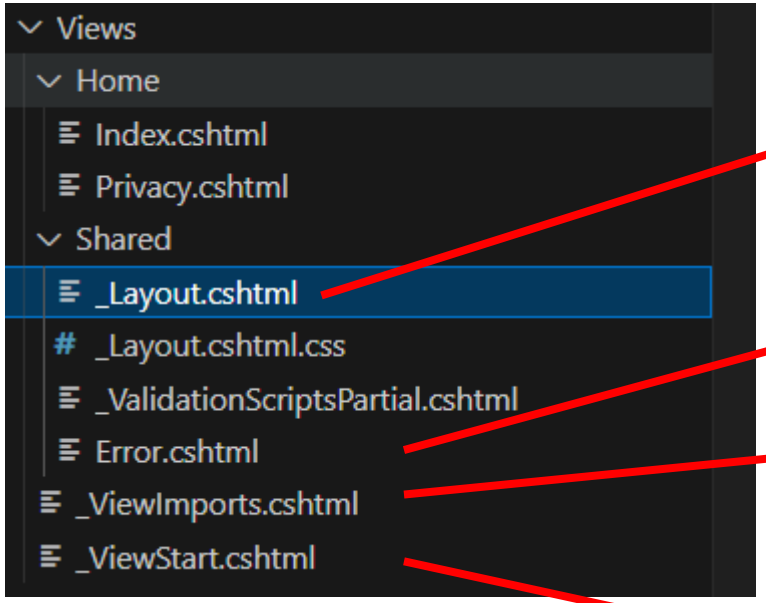
El layout es una vista Razor que define la estructura común de las páginas de un sitio web, como la estructura HTML básica, las secciones comunes (cabecera, pie de página, barra de navegación, etc.), y áreas donde se insertará el contenido específico de cada página.

Las vistas individuales pueden heredar el layout y reempalazar las secciones específicas con su propio contenido. Esto permite mantener una apariencia y estructura coherente en todo el sitio web.



View Layout de MCV Template

En ASP.NET MVC, la carpeta Shared permite almacenar vistas que se comparten entre varios controladores. Por defecto incluye los archivos que se ven a continuación.



- **_Layout.cshtml**: También conocida como “plantilla maestra”. Esta vista define la estructura general de la aplicación web, como la barra de navegación, el encabezado y el pie de página.
- **Error.cshtml:** Esta vista se muestra cuando se produce un error en la aplicación web.
- **_ViewImports.cshtml:** Esta vista se utiliza para importar espacios de nombres y otros elementos comunes en todas las vistas de la aplicación.
- **_ViewStart.cshtml:** Esta vista se ejecuta antes de cualquier otra vista y se utiliza para establecer valores predeterminados para la aplicación web, como el diseño predeterminado o el idioma.

View Layout de MCV Template

_Layout.cshtml define la estructura de la página web, como la cabecera, el menú de navegación, el pie de página, etc.

```
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>
```

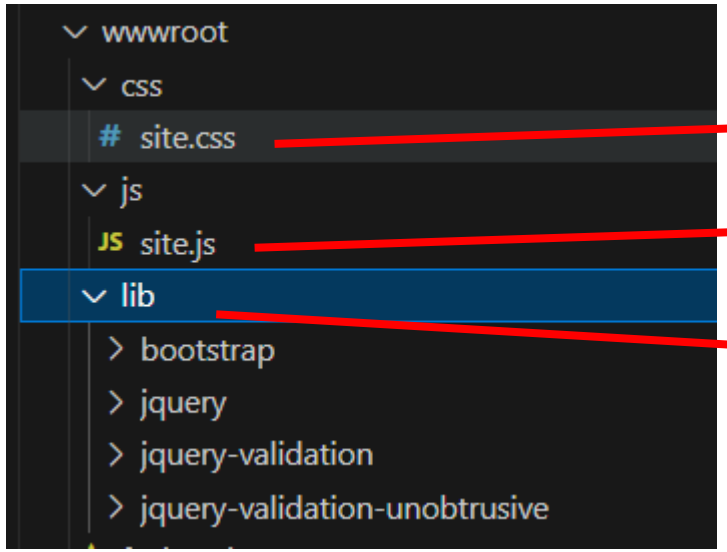
La etiqueta @RenderBody() es una directiva de Razor que se utiliza para renderizar el contenido de una vista en la plantilla Layout.

```
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

@RenderSection("scripts", required: false): Esta es otra directiva de Razor que se utiliza para renderizar la sección de scripts de la vista que se está utilizando.

View Layout de MCV Template

La carpeta wwwroot es una carpeta especial en un proyecto ASP.NET MVC que se utiliza para almacenar archivos estáticos, como archivos HTML, CSS, JavaScript, imágenes, etc



Archivos css del proyecto

Archivos javaScript del proyecto

Se utiliza para almacenar bibliotecas de terceros, como jQuery, Bootstrap, etc.

Bibliografía

https://learn.microsoft.com/es-mx/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-7.0#the-form-action-tag-helper

https://learn.microsoft.com/es-mx/aspnet/core/mvc/views/tag-helpers/built-in/?view=aspnetcore-7.0

Vistas

Ejemplo de Formulario de login

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

```
@model RegisterViewModel
<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email:  <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>
```