Parte 2. Adición de un controlador a una aplicación de ASP.NET Core MVC

21/09/2021Tiempo de lectura: 6 minutos

En este artículo

Por Rick Anderson

El patrón de arquitectura de Modelo-Vista-Controlador (MVC) separa una aplicación en tres componentes principales: **M** odelo, **v** ista y **c** ontrolador. El patrón de MVC ayuda a crear aplicaciones que son más fáciles de actualizar y probar que las tradicionales aplicaciones monolíticas.

Las aplicaciones basadas en MVC contienen:

- **M** odelos: clases que representan los datos de la aplicación. Las clases de modelo usan lógica de validación para aplicar las reglas de negocio para esos datos. Normalmente, los objetos de modelo recuperan y almacenan el estado del modelo en una base de datos. En este tutorial, un modelo `Movie` recupera datos de películas de una base de datos, los proporciona a la vista o los actualiza. Los datos actualizados se escriben en una base de datos.
- **V** istas: Las vistas son los componentes que muestran la interfaz de usuario (IU) de la aplicación. Por lo general, esta interfaz de usuario muestra los datos del modelo.
- Los **c** ontroladores son clases que:
  - Controlan las solicitudes del explorador.
  - Recuperan datos del modelo.
  - Llaman a plantillas de vista que devuelven una respuesta.

En una aplicación MVC, la vista solo muestra información. El controlador controla la entrada y la interacción del usuario y responde a estas. Por ejemplo, el controlador controla los segmentos de URL y los valores de cadena de consulta y pasa estos valores al modelo. El modelo puede usar estos valores para consultar la base de datos. Por ejemplo:

- `https://localhost:5001/Home/Privacy`: especifica el controlador `Home` y la acción `Privacy`.
- `https://localhost:5001/Movies/Edit/5`: es una solicitud para editar la película con ID=5 mediante el controlador `Movies` y la acción `Edit`, que se detallan más adelante en el tutorial.

Los datos de ruta se explican más adelante en el tutorial.

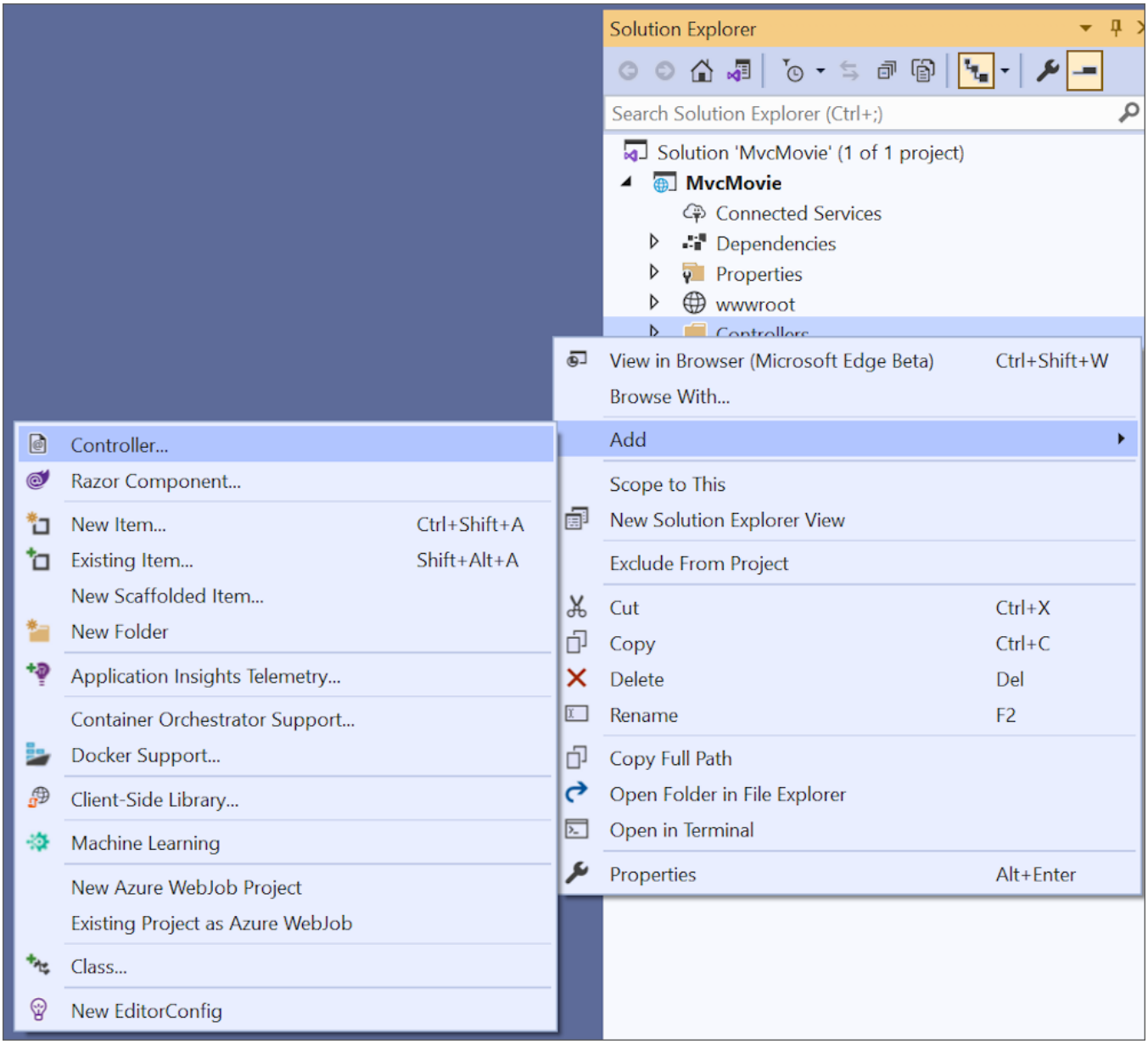
El patrón de arquitectura de MVC separa una aplicación en tres grupos de componentes principales: modelos, vistas y controladores. Este patrón permite lograr la separación de intereses, puesto que la lógica de la interfaz de usuario pertenece a la vista. La lógica de entrada pertenece al controlador. La lógica de negocios pertenece al modelo. Esta separación ayuda a administrar la complejidad al compilar una aplicación, ya que permite trabajar en un aspecto de la implementación de cada vez, sin influir en el código de otro. Por ejemplo, puede trabajar en el código de vista sin depender del código de lógica de negocios.

Estos conceptos se presentan y se muestran en esta serie de tutoriales mientras se compila una aplicación de películas. El proyecto de MVC contiene carpetas para *controladores* y *vistas*.

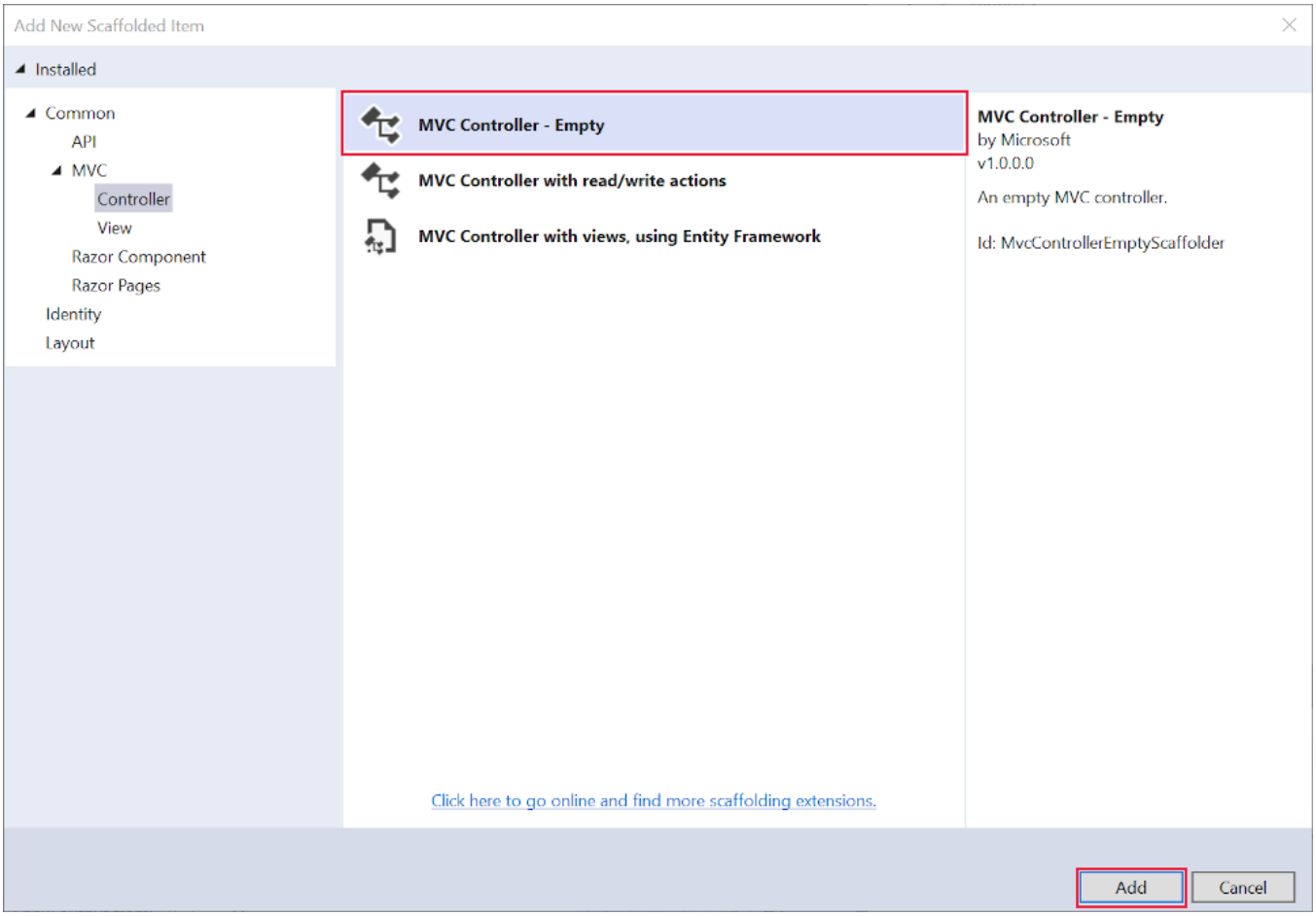
Incorporación de un controlador

Visual Studio | Visual Studio Code | Visual Studio para Mac

En el **Explorador de soluciones**, haga clic con el botón derecho en **Controladores** > **Agregar** > **Controlador**.



En el cuadro de diálogo **Agregar scaffold**, seleccione **Controlador de MVC: en blanco**.



En el **cuadro de diálogo Agregar nuevo elemento: MvcMovie**, escriba **HelloWorldController.cs** y seleccione **Agregar**.

Reemplace el contenido de *Controllers/HelloWorldController.cs* con lo siguiente:

C#

Copiar

```
using Microsoft.AspNetCore.Mvc;
using System.Text.Encodings.Web;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/

        public string Index()
        {
            return "This is my default action...";
        }

        //
        // GET: /HelloWorld/Welcome/

        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
```

```
    }
}
```

Cada método `public` en un controlador puede ser invocado como un punto de conexión HTTP. En el ejemplo anterior, ambos métodos devuelven una cadena. Observe los comentarios delante de cada método.

Un punto de conexión HTTP:

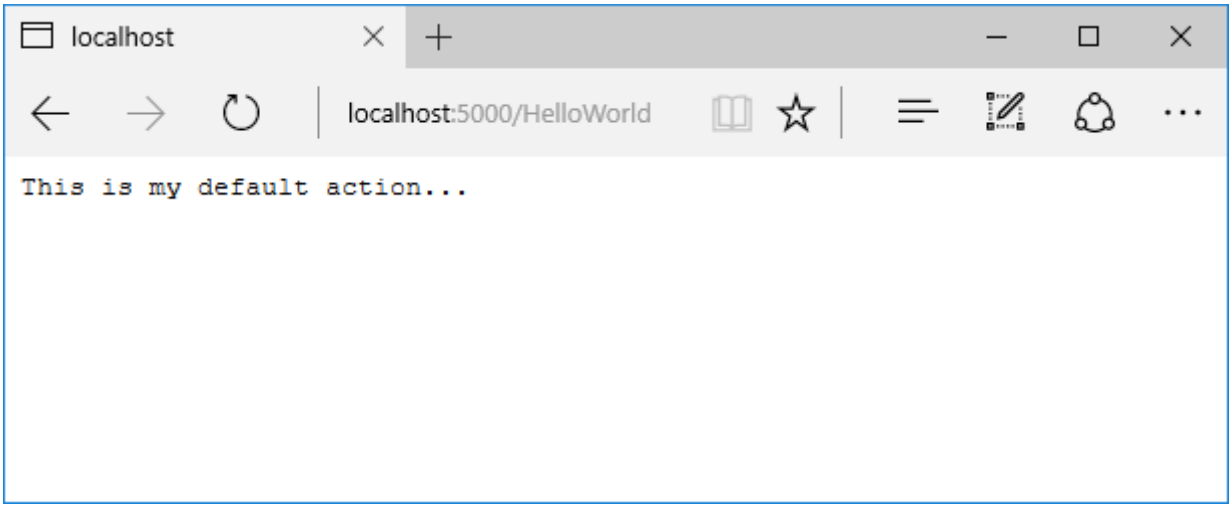
- Es una dirección URL que se puede establecer como destino en la aplicación web, por ejemplo, `https://localhost:5001/HelloWorld`.
- Combina:
  - El protocolo usado: `HTTPS`.
  - La ubicación de red del servidor web, incluido el puerto TCP: `localhost:5001`.
  - El URI de destino: `HelloWorld`.

El primer comentario indica que se trata de un método HTTP GET que se invoca anexando `/HelloWorld/` a la dirección URL base.

El segundo comentario especifica un método HTTP GET que se invoca anexando `/HelloWorld/Welcome/` a la dirección URL. Más adelante en el tutorial se usa el motor de scaffolding para generar métodos `HTTP POST` que actualizan los datos.

Ejecute la aplicación sin el depurador.

Anexe "HelloWorld" a la ruta de acceso en la barra de direcciones. El método `Index` devuelve una cadena.



MVC invoca las clases del controlador y los métodos de acción que contienen, en función de la URL entrante. La lógica de enrutamiento de URL predeterminada que usa MVC emplea un formato similar al siguiente para determinar qué código se debe invocar:

```
/[Controller]/[ActionName]/[Parameters]
```

El formato de enrutamiento se establece en el método `Configure` del archivo *Startup.cs*.

C#

Copiar

```
app.UseEndpoints(endpoints =>
{
```

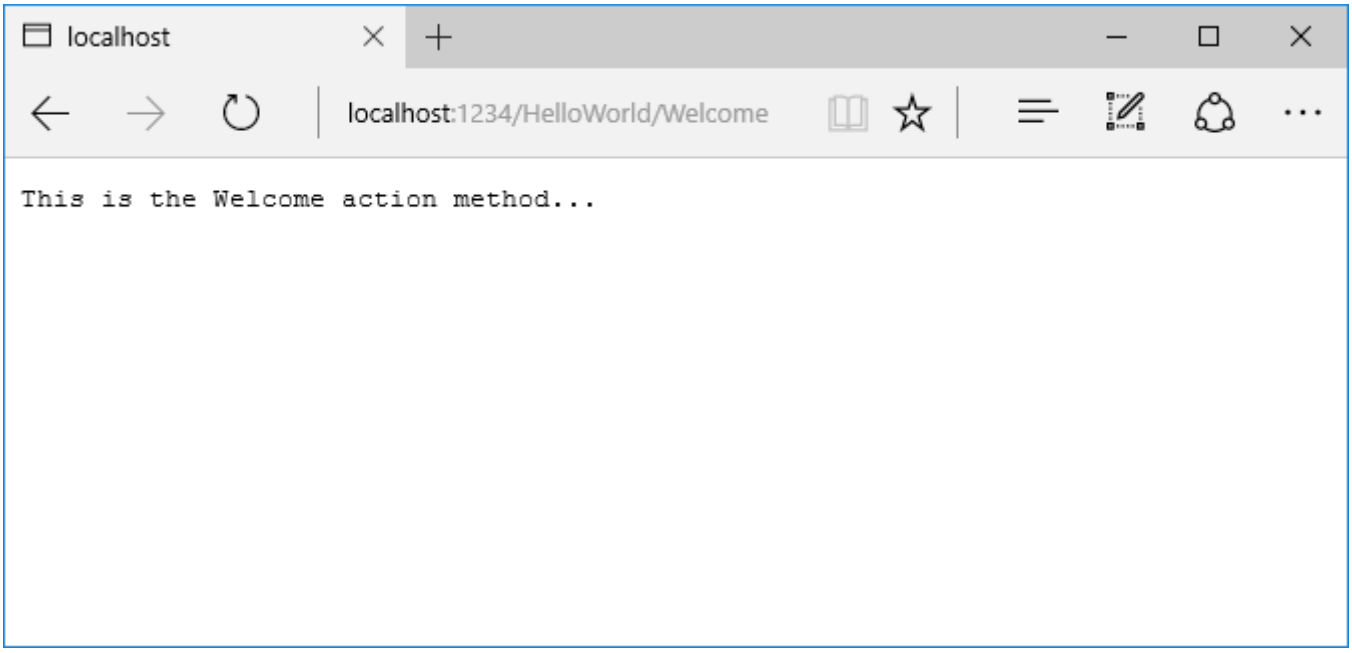
```
);
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");

});
```

Cuando se navega a la aplicación y no se suministra ningún segmento de dirección URL, de manera predeterminada se usan el controlador "Home" y el método "Index" especificados en la línea de plantilla resaltada arriba. En los segmentos de dirección URL anteriores:

- El primer segmento de dirección URL determina la clase de controlador que se va a ejecutar. Por tanto, `localhost:5001/HelloWorld` se asigna a la clase **HelloWorld**Controller.
- La segunda parte del segmento de dirección URL determina el método de acción en la clase. Así pues, `localhost:5001/HelloWorld/Index` provoca que se ejecute el método `Index` de la clase `HelloWorldController`. Tenga en cuenta que solo es necesario navegar a `localhost:5001/HelloWorld` para que se llame al método `Index` de manera predeterminada. `Index` es el método predeterminado al que se llamará en un controlador si no se especifica explícitamente un nombre de método.
- La tercera parte del segmento de dirección URL (`id`) es para los datos de ruta. Los datos de ruta se explican más adelante en el tutorial.

Vaya a `https://localhost:{PORT}/HelloWorld/Welcome`. Reemplace `{PORT}` por el número de puerto.

El método `Welcome` se ejecuta y devuelve la cadena `This is the Welcome action method...`. Para esta dirección URL, el controlador es `HelloWorld` y `Welcome` es el método de acción. Todavía no ha usado el elemento `[Parameters]` de la dirección URL.



Modifique el código para pasar cierta información del parámetro desde la dirección URL al controlador. Por ejemplo: `/HelloWorld/Welcome?name=Rick&numtimes=4`.

Cambie el método `Welcome` para que incluya dos parámetros, como se muestra en el código siguiente.

C#

Copiar

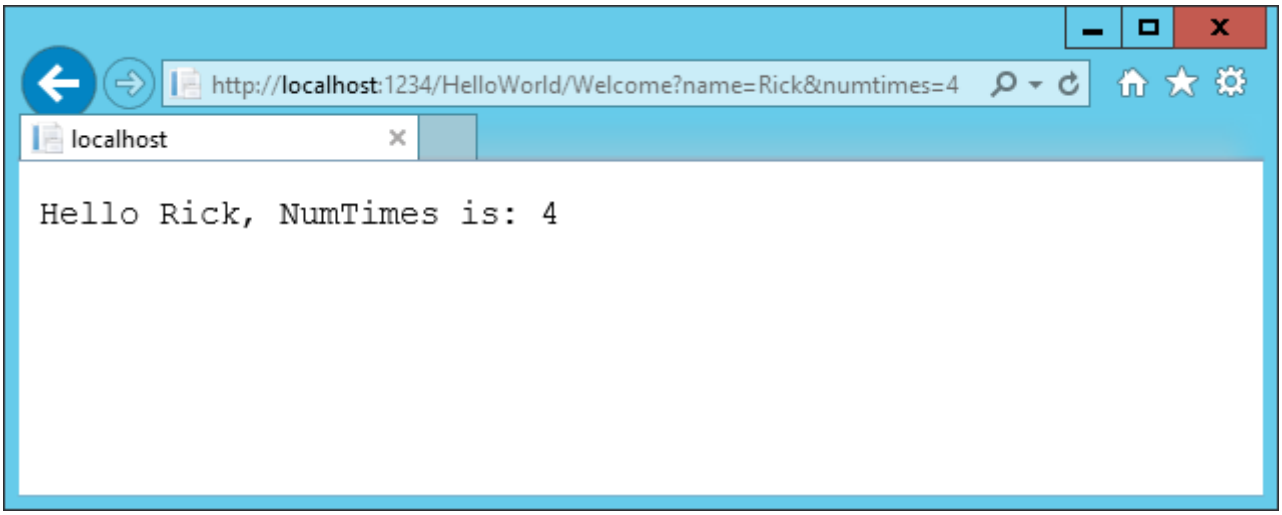
```
// GET: /HelloWorld/Welcome/
// Requires using System.Text.Encodings.Web;
public string Welcome(string name, int numTimes = 1)
{
    return HtmlEncoder.Default.Encode($"Hello {name}, NumTimes is: {numTimes}");
}
```

El código anterior:

- Usa la característica de parámetro opcional de C# para indicar que el parámetro `numTimes` tiene el valor predeterminado 1 si no se pasa ningún valor para ese parámetro.
- Usa `HtmlEncoder.Default.Encode` para proteger la aplicación de entradas malintencionadas, por ejemplo, mediante JavaScript.
- Usa cadenas interpoladas en `$"Hello {name}, NumTimes is: {numTimes}"`.

Ejecute la aplicación y vaya a `https://localhost:{PORT}/HelloWorld/Welcome?name=Rick&numtimes=4`. Reemplace `{PORT}` por el número de puerto.

Pruebe distintos valores para `name` y `numtimes` en la dirección URL. El sistema de enlace de modelos de MVC asigna automáticamente los parámetros con nombre de la cadena de consulta a los parámetros del método. Vea Model Binding (Enlace de modelos) para más información.



En la imagen anterior:

- No se usa el segmento de dirección URL `Parameters`.
- Se pasan los parámetros `name` y `numTimes` en la cadena de consulta.
- El `?` (signo de interrogación) en la dirección URL anterior es un separador y le sigue la cadena de consulta.
- El carácter `&` separa los pares campo-valor.

Reemplace el método `Welcome` con el código siguiente:

C#

Copiar

```
public string Welcome(string name, int ID = 1)
{
```

```
    return HtmlEncoder.Default.Encode($"Hello {name}, ID: {ID}");
}
```

Ejecute la aplicación y escriba la siguiente dirección URL: `https://localhost:{PORT}/HelloWorld/Welcome/3?name=Rick`

En la dirección URL anterior:

- El tercer segmento de dirección URL coincide con el parámetro de ruta `id`.
- El método `Welcome` contiene un parámetro `id` que coincide con la plantilla de dirección URL en el método `MapControllerRoute`.
- El elemento `?` final inicia la cadena de consulta.

C#

Copiar

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

});
```

En el ejemplo anterior:

- El tercer segmento de dirección URL coincide con el parámetro de ruta `id`.
- El método `Welcome` contiene un parámetro `id` que coincide con la plantilla de dirección URL en el método `MapControllerRoute`.
- El elemento `?` final (en `id?`) indica que el parámetro `id` es opcional.