educación IT
CODE YOUR FUTURE

Programación Web .NET Core

Módulo 5

Configurar rutas

Configurar rutas

El motor de enrutamiento de **ASP.NET Core**.

- Discusión: ¿Por qué agregar rutas?
- ¿Qué es la optimización de un motor de búsqueda (Search Engine Optimization)?
- Configuración de rutas mediante enrutamiento basado en convenciones.
- Uso de rutas para pasar parámetros.
- Configurar rutas mediante atributos.
- Demostración: cómo agregar rutas.

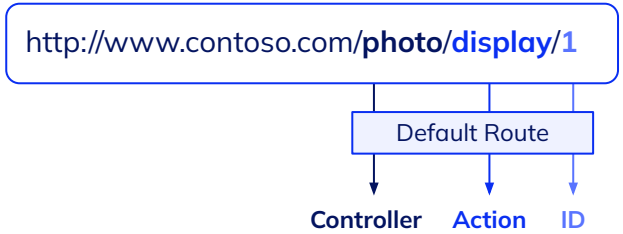
Puedes usar **ASP.NET Core** para controlar las **URL** que usa una aplicación web. También configurar rutas para vincular las **URL** con el contenido. Una ruta es una regla. Como las rutas son configurables, puedes vincular una **URL** y su contenido de forma más eficaz. **ASP.NET Core** usa rutas para analizar una **URL** solicitada para determinar el controlador y acción a la que debe remitirse la solicitud.

El motor de enrutamiento de ASP.NET Core

- **El enrutamiento determina qué controlador y acción deben llamarse** para manejar una solicitud.
- El enrutamiento se puede **configurar** de forma centralizada en el archivo ***Startup.cs*** y en forma local mediante el **uso de atributos**.

Los siguientes pasos ocurren cuando se recibe una solicitud desde un navegador web:

1. Se crea una **instancia de un controlador** para **responder a una solicitud.**
2. **Se examina la URL de la solicitud.** Se utiliza el **enrutamiento** para determinar la acción que debe realizarse.
3. El **enlace de modelo** se utiliza para **determinar los valores** que deben pasarse a la acción como **parámetros.**
4. A continuación, se **invoca la acción** y se crea una **nueva instancia de una clase modelo.** Este objeto modelo **se pasa a una vista** para representar los resultados y enviar una respuesta al navegador.



Discusión: ¿Por qué agregar rutas?

- Para facilitar la comprensión de las URL a los visitantes del sitio.
- Para mejorar la clasificación en los motores de búsqueda.

En ASP.NET Core las rutas se utilizan para dos propósitos:

1. Analizar las **URL solicitadas** por los navegadores. Este análisis garantiza que las solicitudes se envíen a los controladores y acciones correctas. Se denominan ***URL entrantes.***
2. **Formular URL** en enlaces de páginas web y otros elementos. Cuando se usa *Helpers* como ***Html.ActionLink*** en las vistas **MVC**, los *Helpers* construyen una **URL** de acuerdo con las rutas en la tabla de enrutamiento. Se denominan ***URL salientes.***

ASP.NET Core MVC nos permite configurar rutas de dos formas diferentes:

1. Configurar rutas mediante **enrutamiento basado en convenciones.** En este caso, las rutas se configuran en el ***Startup.cs*** y tienen un impacto en toda la aplicación.
2. Configurar rutas **usando atributos**. Como su nombre lo indica, usa atributos para definir las rutas.

Puedes combinar enrutamiento basado en convenciones y enrutamiento basado en atributos en la misma aplicación MVC.

La ruta predeterminada

Se puede configurar una aplicación web **ASP.NET Core** para usar una ruta predeterminada. La ruta debe ser ubicada en el archivo ***Startup.cs*** con el método **`Use.MvcWithDefaultRoute`**.

Este método agrega MVC a la canalización de ejecución de solicitudes con **`IApplicationBuilder{controller=Home}/{action=Index}/{id?}`**.

El siguiente código muestra cómo se implementa la ruta predeterminada en el archivo ***Startup.cs***:

La ruta predeterminada

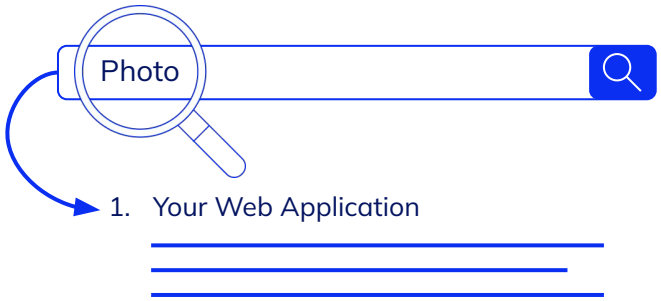
```
public class Startup
{
  // Code is omitted
  public void Configure(IApplicationBuilder app)
  {
     // Code is omitted
     // Add MVC to the request pipeline.
app.UseMvcWithDefaultRoute();
  }
}
```

¿Qué es la optimización de un motor de búsqueda?

La mayoría de los usuarios encuentran aplicaciones web mediante motores de búsqueda. Por eso, es importante asegurarse de que la aplicación web aparezca en la parte superior de los resultados del motor de búsqueda.

Los usuarios tienden a visitar los enlaces que aparecen en la parte superior de la primera página de resultados del motor de búsqueda. Por este motivo, los administradores de sitios web y los desarrolladores intentan asegurarse de que su aplicación web aparezca en ese lugar.

Para esto se valen de un proceso conocido como ***optimización de motor de búsqueda*** o **SEO** (del inglés, ***Search Engine Optimization)***.



Configuración de rutas basada en convenciones

- Las rutas basadas en convenciones pueden contener las siguientes propiedades: **nombre (`name`), plantilla (`template`), valores predeterminados (`defaults`), restricciones (`constraints`) y `dataTokens`**.
- Se pueden agregar rutas personalizadas como se muestra en el ejemplo debajo.

```
app.UseMvc(routes =>
{  routes.MapRoute(
name: "default",
template: "{controller}/{action}/{param}",
defaults: new{ controller = "Some", action = "ShowParam" },
constraints: new{ param = "[0-9]+" });
  });
```

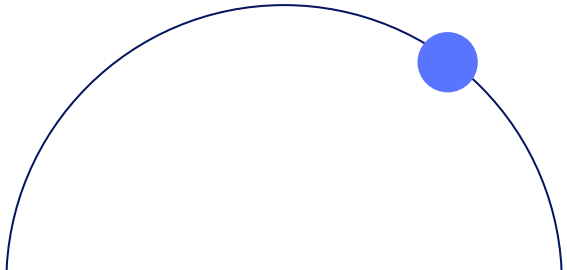
Nota: las rutas deben agregarse en un orden apropiado.

Puedes utilizar el enrutamiento basado en convenciones en la clase ***Startud.cs*** de tu proyecto.

El siguiente código muestra la clase ***Startud.cs*** configurada para usar **MVC**.

En la clase ***Startud.cs*** el método **`ConfigureServices`** agrega el servicio **MVC** y lo utiliza en el método **`Configure`**.

```
public class Startup
{
   public void
ConfigureServices(IServiceCollectionservices)
   {
services.AddMvc();
   }
  public void
Configure(IApplicationBuilderapp)
  {
app.UseMvc();
  }
}
```

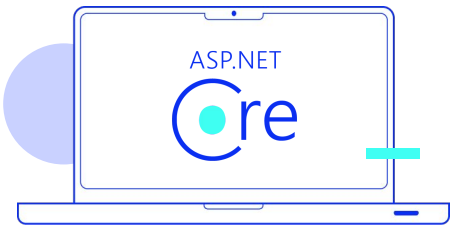


El método **Configure** agrega la canalización de solicitudes **MVC**.

El siguiente código agrega un controlador:

```
public class SomeController : Controller
{
   public IActionResultShow()
   {
      return Content("Reached the action”);
   }
}
```

Si un usuario solicita la **URL** relativa **/**, la solicitud fallará.



Propiedades de una ruta

Las propiedades de una ruta incluyen **`name`** y **`template`**, que son propiedades del tipo cadena.

La propiedad `name` asigna un nombre a una ruta. No participa en la búsqueda de coincidencias ni en el reenvío de solicitudes.

La propiedad **`template`** es un patrón de URL que se compara con una **solicitud URL** para **determinar si se debe utilizar la ruta.**

Por ejemplo, si configura la plantilla de una ruta en **`{controlador}/{acción}`** y luego un usuario solicita la URL relativa **/Some/Show**, la solicitud se realizará correctamente. La variable **`{controlador}`** se asigna a **`SomeController`** y la variable **`{action}`** se asigna a **`Show`**.

Si un usuario solicita la URL relativa **/**, la solicitud fallará porque no se establecieron valores predeterminados.

El siguiente código muestra el método **Configure** en la clase ***Startup*** con una ruta:

El método Configure

```
public void Configure(IApplicationBuilderapp)
{
app.UseMvc(routes =>
{
routes.MapRoute(
            name: "default",
            template:
"{controller}/{action}");
});}
```

```
public void Configure(IApplicationBuilder app)
{
app.UseMvc(routes =>
{
routes.MapRoute(
name: "myRoute",
template: "{action}/{controller}");
});
}
```

Si usas la plantilla **{controller}/{action}**, solo las URL con dos segmentos se considerarán legales. Por ejemplo, si un usuario solicita la URL relativa**/Some/Display/1**, la solicitud fallará.

Agregar template

Para admitir segmentos adicionales, debes actualizar la plantilla en consecuencia.

Por ejemplo: Cambia el valor de la plantilla a **{controlador}/{acción}/{param}**. En este caso, si un usuario solicita la **URL** relativa **/some/display**, la solicitud fallará y si el usuario solicita la **URL** relativa **/some/display/1**, tendrá éxito.

El código muestra cómo se puede agregar un segmento adicional llamado **param** a una ruta:

El método Configure

```
public void Configure(IApplicationBuilderapp)
{
app.UseMvc(routes =>
   {
routes.MapRoute(
      name: "default",
      template:
"{controller}/{action}/{param}");
   });
}
```

Por ejemplo: una acción llamada **ShowParam** obtiene un parámetro llamado **param** y envía su valor al navegador.

En este caso, si un usuario solicita la URL relativa **/Some/ShowParam/hello**, el parámetro `param` obtiene el valor `hola`, que a su vez se envía al navegador.

El siguiente código muestra una acción que obtiene un parámetro llamado **param** y envía su valor al navegador:

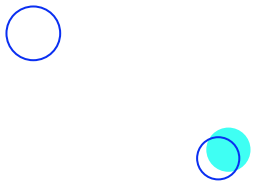
La acción ShowParam

```
public class SomeController : Controller
{
public IActionResultShowParam(stringparam)
{
   return Content(param);
}
}
```

La propiedad predeterminada

La propiedad **`defaults`** es otra propiedad importante de una ruta. Esta propiedad puede **asignar valores predeterminados a las variables de segmento en el patrón de URL.** Los valores predeterminados se utilizan para las variables de segmento cuando la solicitud no los especifica.

Por ejemplo, el siguiente código muestra una ruta con la propiedad `defaults`:



Ruta con propiedad por defecto

```
app.UseMvc(routes =>
{
routes.MapRoute(
   name: "default",
   template: "{controller}/{action}/{param}",
   defaults: new { controller = "Some", action
= "ShowParam", param = "val" });
});
```

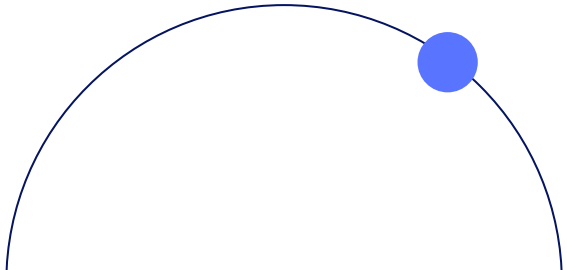
Si usas la ruta anterior, se invocará la acción **`ShowParam`** del controlador **`SomeController`** cuando un usuario solicite las URL relativas **/**, **/Some** y **/Some/ShowParam**. En todos estos casos, el valor de **`param`** será **`val`**.

Los valores predeterminados de una ruta se pueden establecer directamente en la propiedad de la plantilla.

Por ejemplo, es posible reemplazar la plantilla y las propiedades predeterminadas del ejemplo anterior con el siguiente valor **`{controller=Some}/{action=ShowParam}/{param=val}`**.

Estos dos ejemplos crean rutas equivalentes:

```
app.UseMvc(routes =>
{
routes.MapRoute(
   name: "default",
template:
"{controller=Some}/{action=ShowParam}/
{param=val}");
});
```



La propiedad de las restricciones

A veces, es necesario establecer r**estricciones adicionales** en la ruta para asegurarse de que **solo coincida con solicitudes específicas.**

La propiedad de restricciones (**`constraints`**) permite **especificar una expresión regular para cada variable de segmento.** La ruta coincidirá con una solicitud sólo si todas las variables de segmento coinciden con las expresiones regulares que especifique.

Por ejemplo, para asegurarse de que el segmento de parámetro incluya solo dígitos, se puede establecer la ruta con la propiedad de restricciones, como se muestra a la derecha.

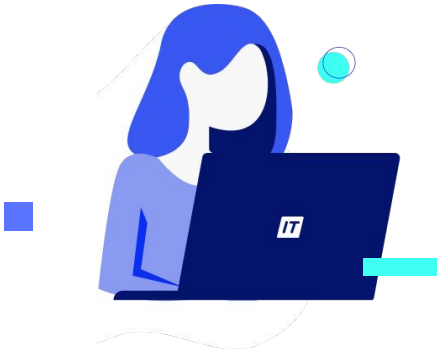
Una ruta con la propiedad de restricciones

```
app.UseMvc(routes =>
{
routes.MapRoute(
   name: "default",
   template: "{controller}/{action}/{param}",
   defaults: new { controller = "Some",
action = "ShowParam" },
constraints: new { param = "[0-9]+" });
});
```

Si un usuario solicita la URL relativa **/Some/ShowParam/1**, la solicitud se realizará correctamente, pero si un usuario solicita la URL relativa **/Some/ShowParam/**, la solicitud fallará.

Es posible especificar restricciones de ruta directamente en la propiedad de la plantilla.

Por ejemplo, para asegurar que el parámetro de valor de ruta sea convertible a un entero, se puede configurar la siguiente ruta:



Una ruta con restricciones en la propiedad de la plantilla

```
app.UseMvc(routes =>
{
routes.MapRoute(
   name: "default",
template:
"{controller=Some}/{action=ShowParam}/
{param:int}}");
});
```

La propiedad dataTokens

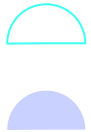
Otra propiedad que se puede establecer en una ruta es la propiedad **dataTokens**, que **habilita los tokens de datos para la ruta.** Cada token de datos contiene un nombre y un valor.

El siguiente código muestra una ruta con la propiedad **dataTokens**:

```
app.UseMvc(routes =>
{
routes.MapRoute(
   name: "default",
   template: "{controller}/{action}/{param}",
   defaults: new { controller = "Some", action =
"ShowParam" },
   constraints: new { param = "[0-9]+" },
dataTokens: new { locale = "en-US" });
});
```

Orden de las rutas

Las rutas se evalúan en el orden en que se agregan. Si una ruta coincide con una solicitud de URL, se utiliza. Si una ruta no coincide, se ignora y se evalúa la siguiente. Por esta razón, debes agregar rutas en el orden apropiado. Primero las más específicas, como las rutas que no tienen variables de segmento y sin valores predeterminados.



Agregar dos rutas

```
public void Configure(IApplicationBuilderapp)
{
app.UseMvc(routes =>
   {
routes.MapRoute(
       name: "firstRoute",
       template: "{controller}/{action}/{param}",
       defaults: new { controller = "Some", action = "ShowParam" },
       constraints: new { param = "[0-9]+" });

routes.MapRoute(
       name: "secondRoute",
       template: "{controller}/{action}/{id?}",
       defaults: new { controller = "Home", action = "Index" });
});
}
```

¡Sigamos trabajando!