educación IT
CODE YOUR FUTURE

Programación Web .NET Core

Módulo 5

Desarrollo de controladores

Introducción a controladores

El controlador es responsable de procesar una solicitud web, interactuar con el modelo y luego pasar los resultados a la vista.

Un controlador es una clase que contiene varios métodos que se denominan *acciones.*

Cuando la aplicación de **MVC** recibe una solicitud, encuentra qué **controlador** y acción deben manejar esa soli- citud. Determina esto mediante el enrutamiento de **URL**. **El enrutamiento de URL** es otro concep- to muy importante y necesario para desarrollar aplicaciones **MVC**.

Programar controladores y acciones

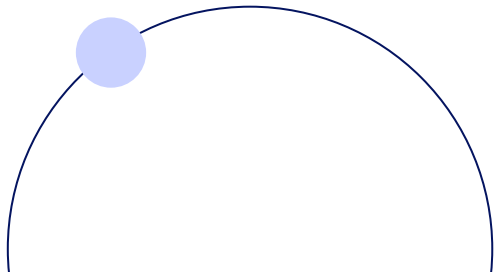
- Responder a las solicitudes de los usuarios.
- Escribir acciones del controlador.
- Usar parámetros.
- Uso de **`ViewBag`** y **`ViewData`** para pasar información a las vistas.
- Ejercicio: Agregar controladores y acciones a una aplicación **MVC**.

Responder a las solicitudes de los usuarios

Cuando una aplicación web **MVC** recibe una solicitud de un navegador, crea un objeto instancia del controlador para responder. Luego determina el método de acción que debe ser llamado en el objeto **`Controller`**.

La aplicación utiliza una carpeta de ***Models*** para determinar los valores a pasar a la acción como parámetros. A menudo, la acción crea una nueva instancia de una clase modelo.

Las plantillas de proyecto de Microsoft Visual Studio incluyen una carpeta denominada ***Controllers***. Los programadores deben crear sus controladores en esta carpeta o en sus subcarpetas.



El siguiente código muestra un controlador que crea un modelo y lo pasa a una vista:

Controller

```
public class HomeController : Controller
{
    public IactrionResult Index()
    {
        SimpleModel model = new SimpleModel() { Value = ¨My Value¨ };
        return View(model);
    }
}
```

Model

```
public class SimpleModel
{
    public string Value { get; set; }
}
```

View

```
@model MyWebApplication.Models.SimpleModel

@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <title>Document</title>
</head>
<body>
    @Model.Value
</body>
</html>
```

Escribir métodos de acción en el controlador

- El método de acción debe ser **público** en la clase **`controlador`**.
- Cada uno de los métodos públicos en una clase **`Controller`** es considerado una **`Action`**.
- Las acciones pueden devolver objetos que implementan la interfaz **`IActionResult`**.

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

Las acciones o llamadas que realiza el usuario se hacen mediante una **Url**, donde la estructura es **Dominio/Controlador/Acción**. Por ejemplo: al **controlador `Home`** y la **acción `Index`** se accede mediante la **Url localhost/Home/Index**.

Una ***Acción*** posee diferentes funciones según cuál sea el método de llamada **HTTP**, por ejemplo, **GET** o **POST**. Por defecto es **GET**, por eso, no hace falta especificar. Los demás métodos se especifican por medio de atributos, antes de declarar el método de acción.

Cabe mencionar que en el lenguaje **C#** no se pueden declarar dos funciones en una misma clase con el mismo nombre y tipos de parámetros sino que deben tener diferentes tipos para reconocer la sobrecarga.

Si la función **Index** sin parámetros existe, se debe crear la siguiente función **Index** con algún parámetro, como en el ejemplo de la derecha:

```
// GET: Home
public ActionResultIndex()
{
    return View();
}

[HttpPost]
public ActionResultIndex(int Id)
{
    return View();
}
```

Puede pasar que exista una acción con dos métodos de llamada **HTTP** distintos y requieran el mismo tipo de parámetro. Por ejemplo en una acción ***Eliminar***, podemos tener una función para el método **GET** que se utiliza para confirmar la eliminación y recibe como parámetro el Id del registro que va a eliminar. Además podemos tener una función para el método **POST** que realiza la eliminación y también recibe como parámetro el **Id** del registro que va a eliminar.

Para solucionar el problema de las sobrecargas de funciones se puede cambiar el nombre de la función para el método **POST** y vincularlo a la acción ***Eliminar***, como por ejemplo **`ConfirmarEliminar`**.

```
public ActionResultEliminar(int Id)
{
    ...
}

[HttpPost, ActionName("Eliminar")]
public ActionResultConfirmarEliminar(int Id)
{
    ...
}
```

Si se accede a **Dominio/Controlador/Eliminar** mediante el método **POST** (por medio de un formulario html), la función que se va a ejecutar es **`ConfirmarEliminar`**.

Comunicación con la Vista

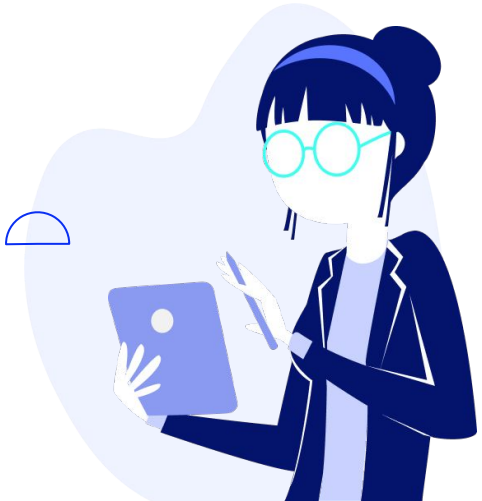
Cada Controlador posee una carpeta en **Views** con su mismo nombre para alojar las vistas de las acciones que lo requieran. Estas vistas deben llevar el mismo nombre que la acción a la que pertenecen para que queden vinculadas automáticamente. En la acción del controlador con el código **`return View();`** busca la vista con su mismo nombre. Sin embargo, se puede cambiar la vista ingresando el nombre como parámetro de la función **`View()`**.

```
public ActionResultIndex()
{
    return View("About");
}
```

Incluso se puede vincular una vista que no se encuentra en la carpeta del controlador en ***Views***. Se debe especificar la ruta completa donde se encuentra, como por ejemplo: ***~/Views/Common/List.cshtml***.

De esta manera, se pueden tener vistas genéricas que comparten varios controladores.

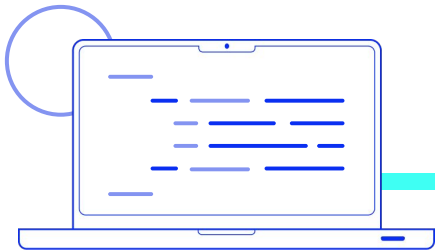
```
public ActionResultIndex()
{
    return View("~/Views/Common/List.cshtml");
}
```



Tipos de resultados del método de acción

`ActionResults`: es una clase base para todos los resultados de la acción.

`IActionResults`: es una interfaz. Eso define el contrato que representa el resultado de un método de acción.



Lista de resultados de acciones

`ViewResult`	Retorna una vista.
`ContentResult`	Retorna una cadena.
`RedirectToResult`	Redirige a la URL específica.
`RedirectToRouteResult`	Redirige a un método de acción diferente o un método de acción de controlador diferente.
`PartialViewResult`	Retorna una vista parcial.
`ViewComponentResult`	Retorna un componente de vista.
`FileResult`	Retorna un archivo.
`EmptyResult`	No retorna nada.

Lista de resultados de acciones

`JsonResult`	Retorna datos en formato JSON.
`HttpNotFoundResult`	Retorna la página de error 404 no encontrada.
`HttpStatusCodeResult`	Retorna un código de estado de respuesta HTTP y una descripción específicos.
`HttpUnauthorizedResult`	Retorna el código de estado HTTP 403.
`JavascriptResult`	Retorna un script para su ejecución.
`FileStreamResult`	Retorna el contenido del archivo.
`FilePathResult`	Retorna el contenido del archivo.
`FileContentResult`	Retorna el contenido del archivo.

Algunos ejemplos de IActionResult:

ContentResult

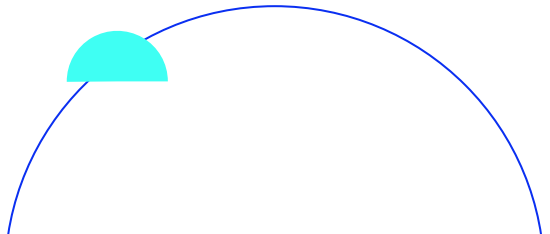
```
Public ContentResult AnotherAction()
{
     return Content("some text");
}
```

RedirectToRouteResult

```
public RedirectToRouteResult Index()
{
     return RedirectToRoute(new
{
controller = "Another",
action = "AnotherAction"
});
}
```

RedirectToAction

```
[HttpPost]
public ActionResult Edit(Student std)
{
     return RedirectToAction("Index");
}
```



```
PartialViewResult
public IActionResultPartialAction ()
{
     List<string>ListofString = new
List<string>();
ListofString.Add("One");
ListofString.Add("Two");
     return PartialView("_DemoPartial",
ListofString);
}
```

```
ViewComponentResult
public IActionResultViewComponentResponse()
 {
     return ViewComponent("DisplayPrice");
 }
```

```
StatusCodeResult
public StatusCodeResultIndex()
{
    return new StatusCodeResult(404);
}
```

Usar parámetros

Los parámetros son enviados mediante la solicitud del usuario a los métodos de acción.

Hay varias formas de recuperar parámetros, que incluyen:

- La propiedad **`Request`** (se verán en el módulo de View).
- El objeto **`FormCollection`** (se verán en el módulo de View).
- **Enrutamiento**.

Todos los métodos de acción pueden tener parámetros de entrada como los métodos normales. Pueden ser un tipo de dato primitivo o parámetros de tipo complejo, como se muestra a continuación.

Todos los métodos de acción pueden tener parámetros de entrada como los métodos normales. Pueden ser un tipo de dato primitivo o parámetros de tipo complejo, como se muestra debajo.

```
//parametro con un tipo de dato complejo (modelo)
“Student std”

[HttpPost]
public ActionResult Edit(Student std)
{
    // update student to the database

    return RedirectToAction("Index");
}
```

```
//Parametro con un tipo de dato primitive “int id”

[HttpDelete]
public ActionResult Delete(int? id)
{
    // delete student from the database whose id
matches with specified id

    return RedirectToAction("Index");
```

Se debe tener en cuenta que el parámetro del método de acción puede ser de tipo **`nullable.`**

El siguiente código muestra la ruta que aparece en el archivo ***Startup.cs:***

```
Ejemplo de ruta
public void Configure(IApplicationBuilder app)
{
app.UseMvc(routes =>
{
routes.MapRoute(
   name: "default",
   template: "{controller}/{action}/{id?}",
   defaults: new { controller = "Home", action =
"Index" });
});
}
```

El siguiente código muestra como el enlace del modelo ubica los parámetros en los valores de enrutamiento:

```
La clase Controller
public class HomeController : Controller
{
public IActionResult Index(string id)
{
   return Content(id);
}
}
```

Cuando el usuario escribe la **URL http://host/Home/Index/1**, el método de acción **`Index`** del **`HomeController`** recibe el parámetro **`id=1`**. Luego de obtener el valor del parámetro, el **`Id`** se muestra como salida en el navegador.

La propiedad RouteData

El enlace del modelo usa la propiedad **RouteData** que encapsula la información sobre la ruta. Contiene una propiedad denominada ***Values***, que se puede utilizar para obtener el valor del parámetro.

El siguiente código muestra cómo el enlace del modelo localiza parámetros en **"Values"** del enrutamiento mediante la **Propiedad RouteData**. Tiene el mismo resultado que el código anterior:

La clase Controller

```
public class HomeController : Controller
{
      public IActionResultIndex()
      {
    string id =
(string)RouteData.Values["id"];
          return Content(id);
      }
}
```

Cuando el usuario escribe la URL **http://host/Home/Index?title=someTitle**, la solicitud invoca el método de acción **Index** de la clase **HomeController** y envía el parámetro **title** con el valor **someTitle**. Luego el navegador muestra un título como la salida.

El siguiente código muestra como el enlace del modelo ubica los parámetros en los valores de enrutamiento:

```
La clase Controller
public class HomeController : Controller
{
public IActionResult Index(string id)
{
   return Content(id);
}
}
```

Cuando el usuario escribe la URL **http://host/Home/Index?title=someTitle**, la solicitud invoca el método de acción **Index** de la clase **HomeController** y envía el parámetro **title** con el valor **someTitle**. Luego el navegador muestra un título como la salida.

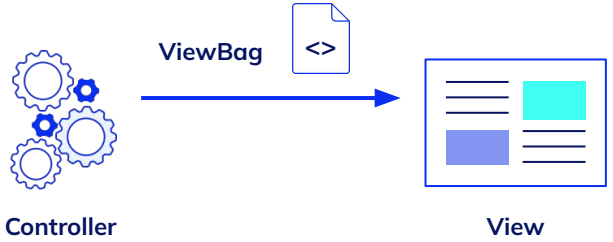
El siguiente código muestra cómo el enlace del modelo ubica los parámetros en una cadena de consulta:

La clase Controller

```
public class HomeController : Controller
{
      public IActionResultIndex(string title)
      {
          return Content(title);
      }
}
```

Uso de ViewBag y ViewData para pasar información a las vistas

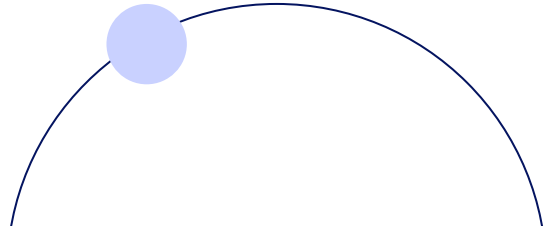
- **Los modelos son la mejor manera de pasar datos de los controladores a las vistas.**
- En algunos casos, es posible que desees aumentar el modelo con datos adicionales sin modificarlo.



Puedes pasar un objeto modelo de la acción a la vista mediante el método **`View()`**. Este método se utiliza con frecuencia para pasar información de una acción del controlador a una vista.

Se recomienda este enfoque porque se adhiere estrechamente al patrón **MVC**, en el que cada vista muestra las propiedades que se encuentran en la clase de modelo que recibe del controlador. Debes utilizar este enfoque siempre que sea posible.

Algunas vistas no utilizan un modelo. La página de inicio de un sitio web, por ejemplo, a menudo no tiene una clase de modelo específica. Para proporcionar información desde el servidor a la vista se pueden utilizar las propiedades **`ViewBag`** y **`ViewData`**.



Usar la propiedad ViewBag

La propiedad **ViewBag** es un objeto contenedor dinámico que forma parte de la clase base Controller. Es un objeto dinámico, puedes agregarle valores de cualquier tipo en el método de acción. En la vista, puede utilizar la propiedad **ViewBag** para obtener los valores agregados en el método de acción.

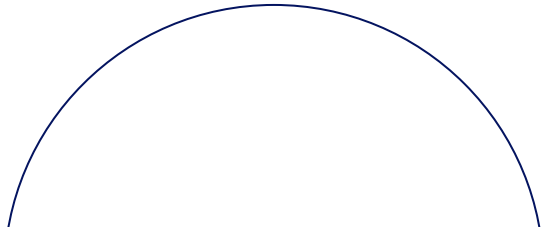
El siguiente código muestra cómo agregar la propiedad **ViewBag**:

```
public class HomeController : Controller
{
    public IActionResultIndex()
    {
ViewBag.Message = "some text";
ViewBag.ServerTime = DateTime.Now;
        return View();
    }}
```

El siguiente código muestra cómo obtener y utilizar las mismas propiedades en una vista mediante **Razor**:

Usar las propiedades de ViewBag

```
<p>
Message is:@ViewBag.Message
</p>
<p>
Server time is: @ViewBag.ServerTime.ToString()
</p>
```



Utilización de la propiedad ViewData

Puedes pasar datos adicionales a las vistas mediante la propiedad **ViewData**. Esta función está disponible para desarrolladores que prefieren utilizar un diccionario de objetos. De hecho, **ViewBag** es un contenedor dinámico y **ViewData** es un diccionario. Esto significa que puedes asignar un valor en una acción del controlador utilizando **ViewBag** y leer el mismo valor en la vista utilizando **ViewData**.

```
<p>
Message is:@ViewData[“Message”]
</p>
```

¡Sigamos trabajando!