Cursos de Programación Web .NET

Unidad 1

“Web: MVC .NET parte 1”

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Índice

ASP.NET MVC

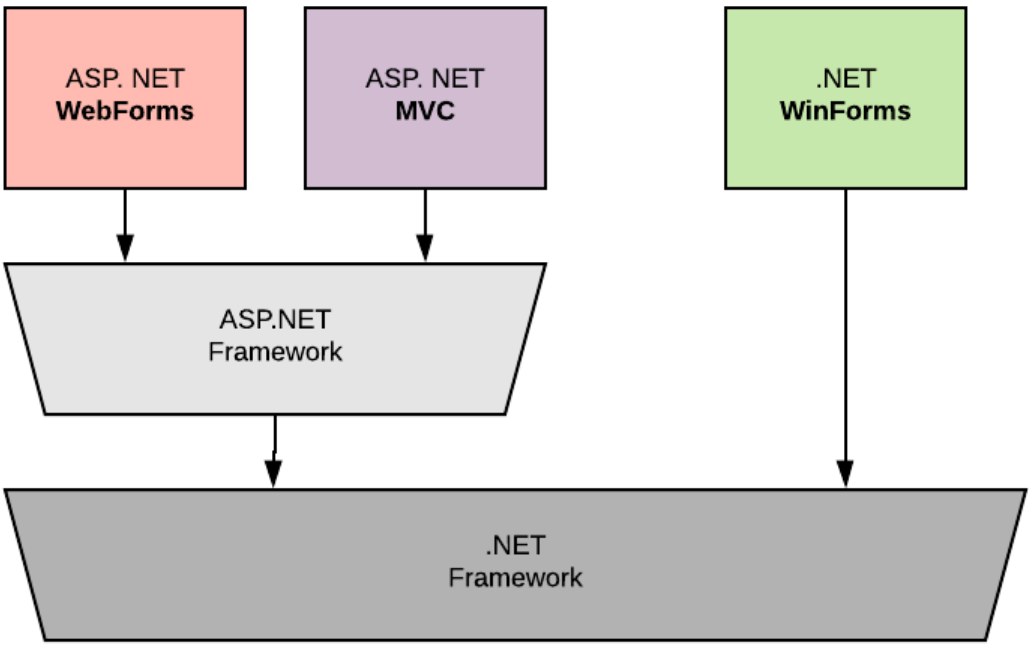
Introducción

MVC es un framework de desarrollo web de Microsoft que está basado en el patrón MVC (modelo-vista-controlador). Este framework por lo tanto nos brinda un modelo desacoplado de desarrollo y a su vez nos permite utilizar los componentes ASP.NET para agilizar la programación.

Entonces MVC no es un Framework hecho desde cero sino que es una alternativa escalable a ASP.NET **WebForms**.

Diagrama ASP.NET

Cuando se distribuyó la primer versión de ASP.NET la 1.0 era muy fácil pensar que ASP.NET y WebForms era lo mismo. Pero esto no es así, una cosa es el Framework Web ASP.NET (**System.Web**) y otra cosa es



WebForms (**System.Web.UI**) que se monta sobre este.

WebForms es la capa que maneja todo lo referido a los controles y la ViewState, básicamente todo lo referido a la visualización. En cambio el Framework se encarga de la capa de la comunicación HTTP y todo lo necesario para que la web funcione.

Un par de años más tarde se anunció y se empezó a distribuir MVC ASP.NET (**System.Web.Mvc**) como una alternativa moderna a la capa de WebForms.

En el diagrama anterior podemos ver que sea cual sea nuestra elección MVC o WebForms siempre vamos a estar sobre el paragüas de ASP.NET. En el caso de los desarrollos de escritorio (WPF o WinForms) estos utilizan el Framework .NET directamente ya los mismos que no son de tipo Web.

Ventajas de MVC

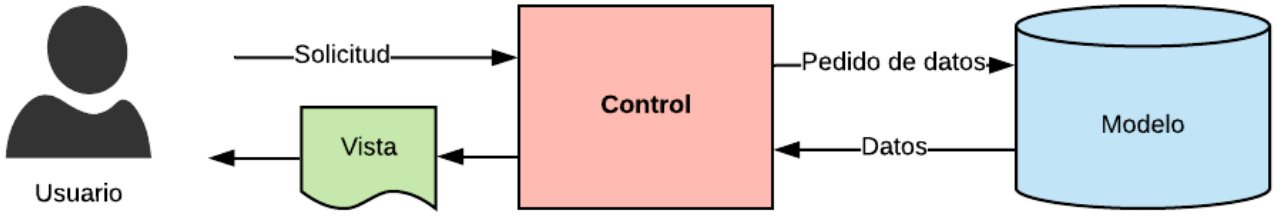
Con MVC lo que se buscó lograr fue simplificar cuestiones arraigadas al diseño de WebForms y la Web HTTP, uno de ellos es que en MVC no hablamos de la ilusión de estados, no existe el page_load ni tampoco tenemos que preocuparnos por el ciclo de vida de la página ni nada por el estilo. Lo que sí tenemos es técnicas para persistir la información por la característica del protocolo HTTP.

- La principal ventaja es que nos permite manejar la complejidad de una aplicación dividiéndola en 3 capas: modelo (datos), vista (UI) y controlador (lógica)
- Nos permite acceder y tener control sobre el código HTML directamente lo que también nos ayuda a estar en acuerdo con estándares de accesibilidad.
- Aplicar estilos y hacer que nuestra página sea más responsiva e interactiva es mucho más fácil en MVC, ya que al controlar el HTML generado puedo aplicar eficientemente este tipo de Frameworks de css y js.
- Componentes separados permiten distribuir el desarrollo del código entre grandes equipos de desarrolladores y diseñadores web, para quienes el control sobre el HTML generado es crucial.

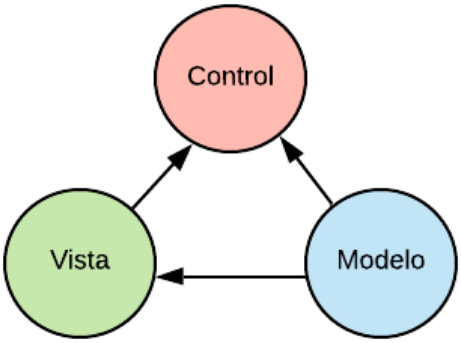
Patrón MVC

El patrón MVC nos permite separar una aplicación en tres grupos principales de componentes: modelos, vistas y controladores lo cual me permite separar los intereses y las responsabilidades dentro de la aplicación.

Este patrón está centrado en el usuario el cual llega con un solicitud que se enruta a un **controlador** el mismo se encarga de operar junto al **modelo** para satisfacer esa solicitud. Una vez que se realizó el pedido del usuario, ya sea realizando las acciones pedidas o recuperando resultados de consultas, se procede a elegir la **vista** para mostrar al usuario el cual le proporcionará cualquier dato del modelo que sea necesario.



La vista y el controlador dependen del modelo. Sin embargo, el modelo no depende de la vista ni del controlador. Esta es una de las ventajas principales de la separación. Esta separación permite compilar y probar el modelo con independencia de la presentación visual. Esto está representado en el siguiente diagrama:



Una ventaja grande que tiene este tipo de modelo es que al separar las responsabilidades es mucho más fácil desarrollar y depurar en un modulo que solo tenga una. Por ejemplo, la lógica de la vista de usuario o de la UI cambia más frecuentemente que la lógica del controlador o el modelo de datos, por lo tanto tiene sentido que en vez de crear un solo objeto con toda la lógica los separemos, ya que si así lo fuera modificar la UI significaría volver a probar tanto la UI como la lógica de negocios.

Modelo

El modelo representa el estado de la aplicación como así también la lógica de negocios u operaciones que deba realizar. La lógica de negocios debe encapsularse en el modelo, junto con cualquier lógica de implementación para conservar el estado de la aplicación. Las vistas fuertemente tipadas normalmente usan tipos ViewModel diseñados para que contengan los datos para mostrar en esa vista. El controlador crea y rellena estas instancias de ViewModel desde el modelo.

Vista

Las vistas se encargan de presentar el contenido a través de la interfaz de usuario. El framework usa un motor de vistas llamado Razor para incrustar código .NET en formato HTML un proceso parecido al que hacia WinForms pero este es mucho más controlable.

Si se necesita una gran cantidad de lógica en los archivos de vistas para mostrar datos en un modelo complejo se debe recurrir a un componente intermedio de vista llamado ViewModel.

Controlador

Los controladores se encargan de controlar la interacción de usuario por lo tanto trabajan con el modelo y si la operación lo amerita seleccionan una vista para representarla. La vista solo muestra información, el controlador controla y responde a la interacción y datos que ingresa el usuario.

El controlador es el punto de entrada inicial que se encarga de seleccionar con que modelos y vistas trabajar. De ahi su nombre, controla el modo en que una solicitud se procesa en el sistema,

Los controladores son los componentes que controlan la interacción del usuario, trabajan con el modelo y, en última instancia, seleccionan una vista para representarla. En una aplicación de MVC, la vista solo muestra información; el controlador controla y responde a la interacción y los datos que introducen los usuarios.

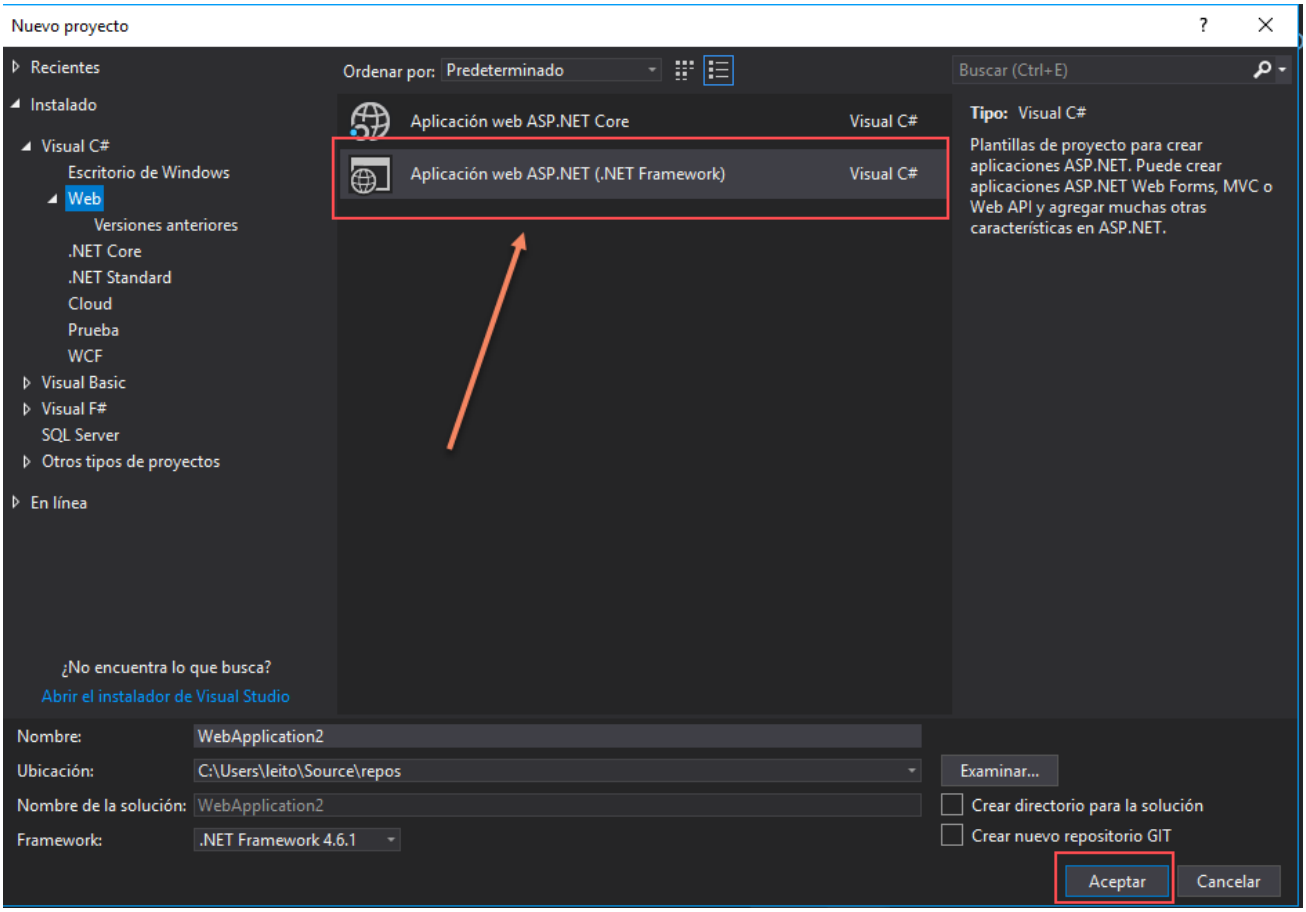
En el patrón de MVC, el controlador es el punto de entrada inicial que se encarga de seleccionar con qué tipos de modelo trabajar y qué vistas representar (de ahí su nombre, ya que controla el modo en que la aplicación responde a una determinada solicitud).

App de ejemplo

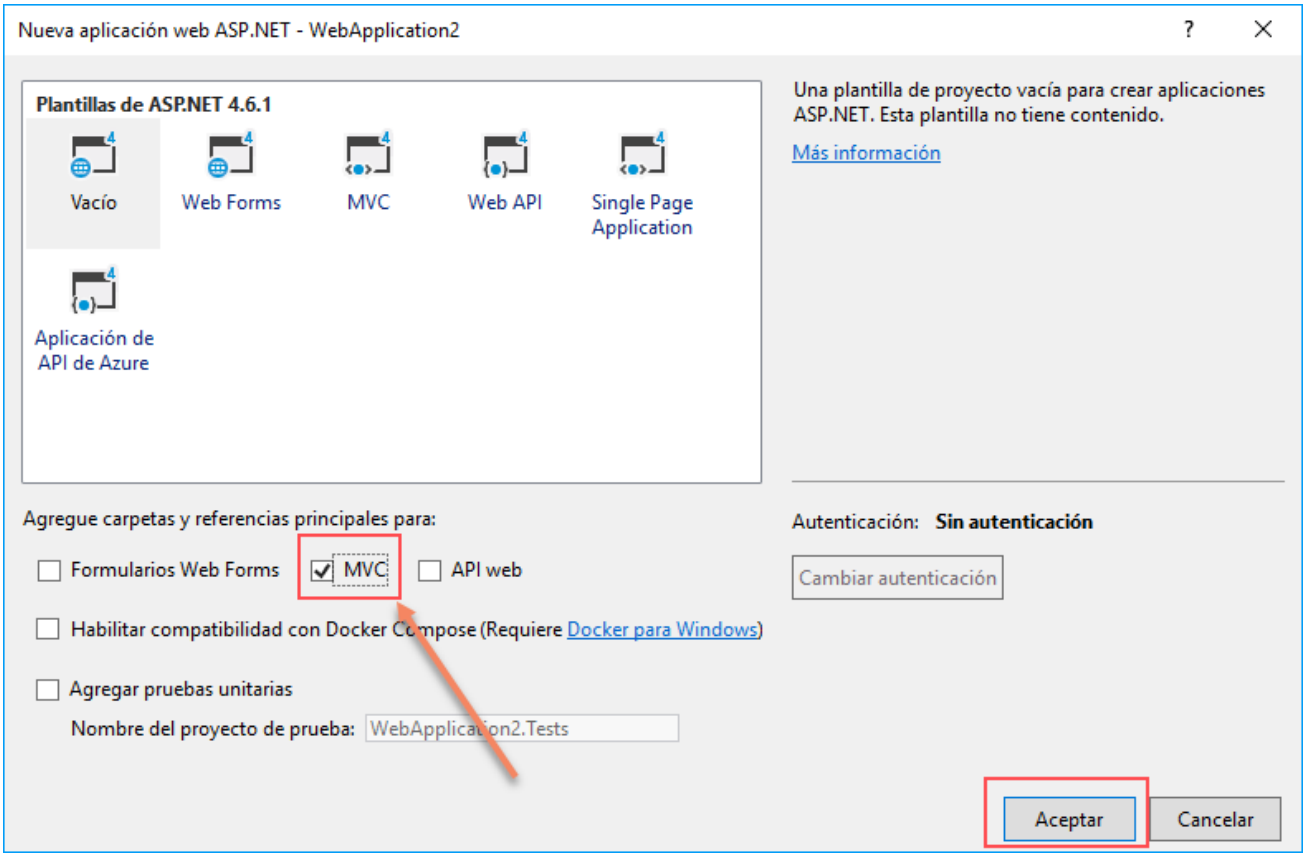
Vamos a hacer un clásico hola mundo con MVC:

Primero tenemos que crear el proyecto MVC que ya viene incluido en el paquete ASP.NET.

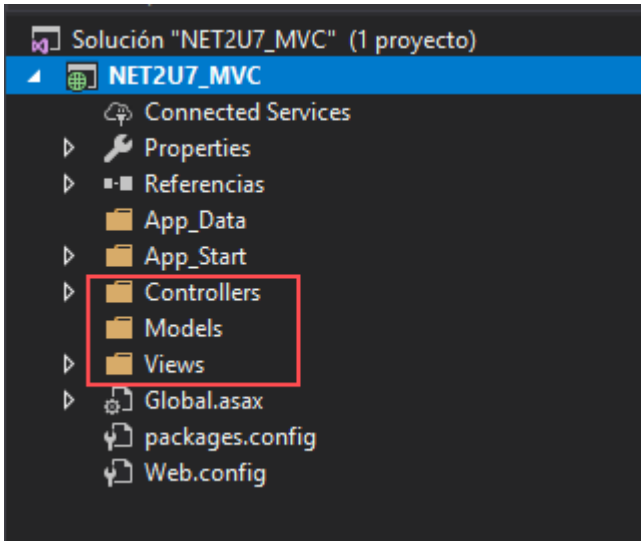
Creamos una aplicación de tipo web ASP.NET:



En el próximo paso nos aseguramos de seleccionar MVC en las opciones:



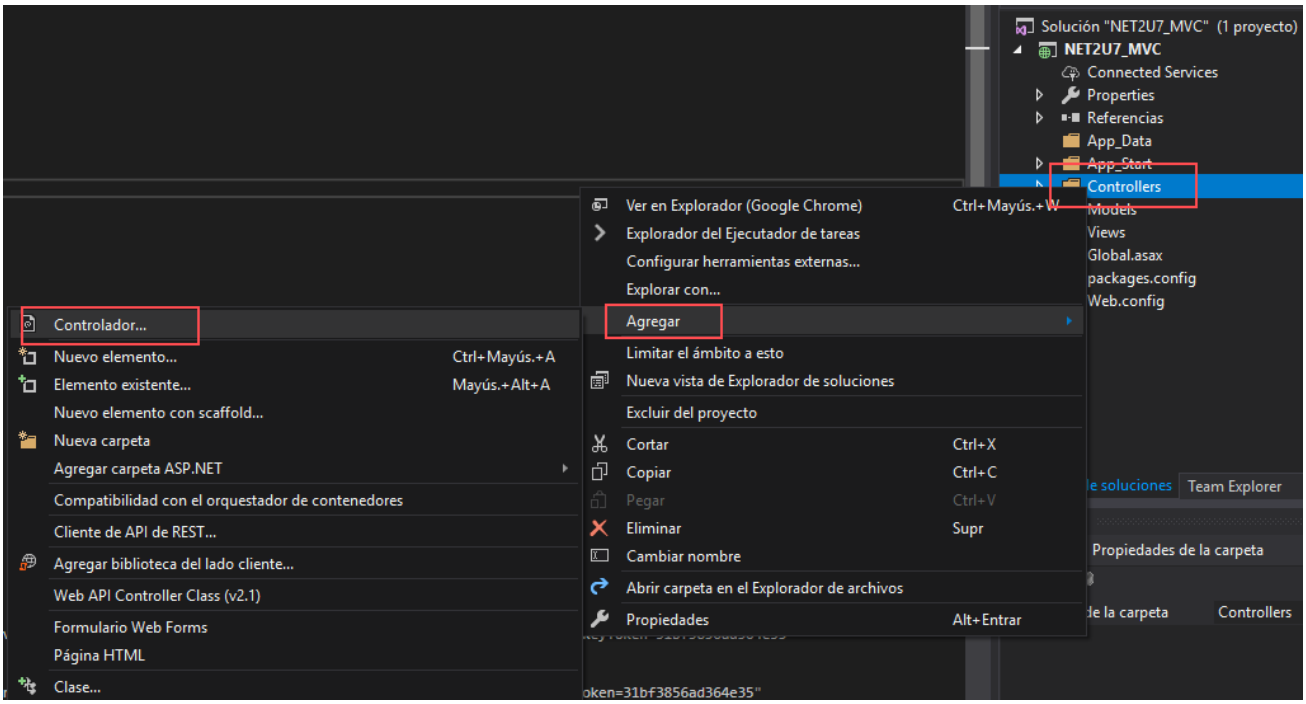
A partir de las acciones anteriores un nuevo proyecto va a ser creado con la siguiente estructura:



Nótese la existencia de nuevas carpetas que serán utilizadas para separar la lógica de nuestra aplicación y crear en ellas Controladores, Modelos y Vistas de manera separada.

Para que nuestra aplicación funcione mínimamente necesitamos un controlador el cual hará las veces de atender al usuario y enviarle el mensaje “Hola Mundo”

Para ellos procedamos a agregar un nuevo controlador:

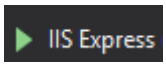


Para realizar la acción descripta debemos codificar el control de la siguiente manera:

```
public class HomeController : Controller
{
    // GET: PrimerEjemplo
    public string Index()
    {
        return "Hola Mundo MVC";
    }
}
```

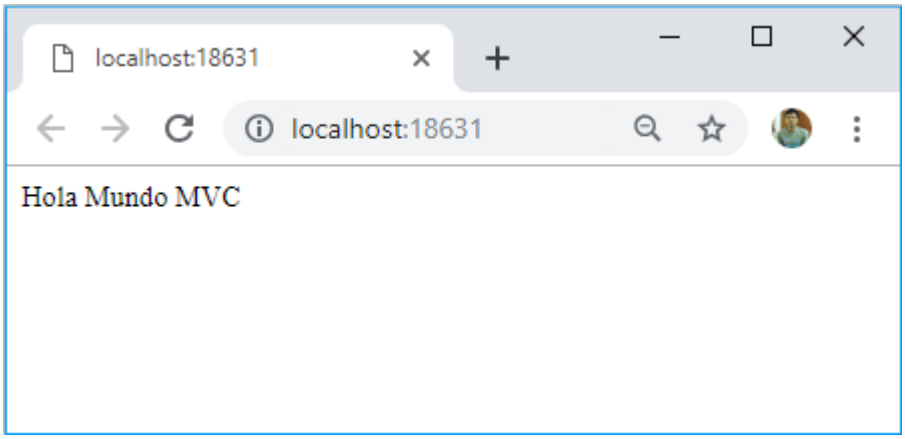
Este código nos va a devolver un `string` al buscar el home de nuestra Página web. Cuando busquemos la dirección sin una ruta especificada esta va a caer en la dirección por defecto que es el index.

El controlador puede devolver una vista, realizar una acción o en este caso devolver un string directamente al usuario.



Vamos a ejecutar nuestra página:

Y el resultado va a ser el siguiente:

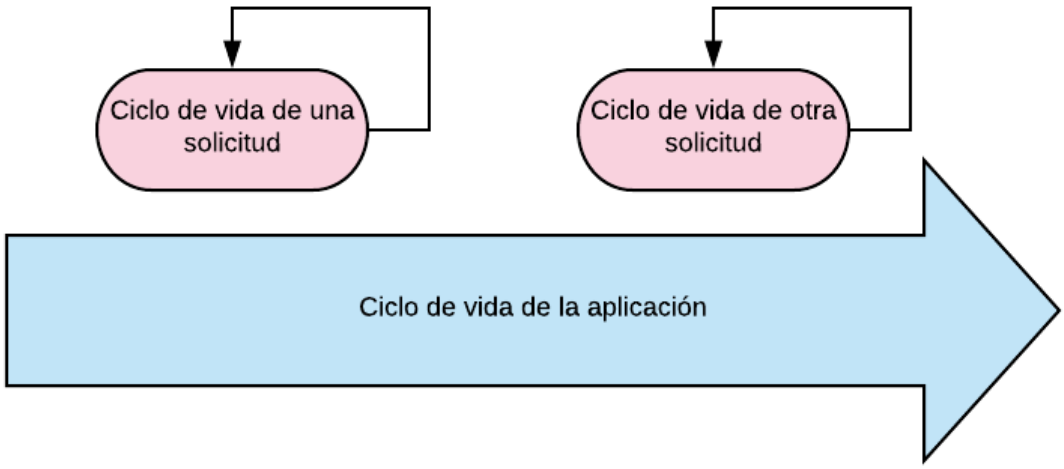


Felicitaciones ya dimos el primer paso en el mundo MVC. Vamos ahora a ver otros conceptos que van a ayudarnos a entender más en detalle su funcionamiento para poder realizar aplicaciones más complejas.

Como funciona MVC

ASP.NET MVC cuenta principalmente con dos ciclos de vida:

- Ciclo de vida de la aplicación
- Ciclo de vida de las solicitudes de usuario (requests)



Ciclo de vida de la aplicación

Este comprende desde el inicio de la aplicación web, es decir, desde el momento que se inicia el servidor IIS con nuestra aplicación, hasta el momento en el que se apaga el servicio.

Ciclo de vida de los requests o solicitudes de usuarios

Esta es la secuencia que corre cada vez que un usuario hace un pedido de HTTP por medio del navegador web. El punto de entrada como ya hemos discutido comienza con el **Routing**, en el momento de recibir un **request** (pedido o solicitud) ASP.NET sabe a donde enviar el pedido por medio de el modulo de **routing URL**.

El framework MVC convierte los datos del **routing** en un **controlador** que va a atender esos pedidos.

Luego que este controlador fue creado el gran paso siguiente es el "**Action Execution**", en el cual un componente llamado **action invoker** encuentra y selecciona la acción apropiada a realizar por el controller.

Una vez que esa acción fue realizada y tenemos el resultado listo entonces tenemos que devolverlo, esto es el **Result Execution**. MVC distingue entre declarar el resultado a executar el resultado. Si este resultado es un tipo de vista, entonces se llama al **View engine** y es responsable de renderizar la página web al usuario.

En cambio si el resultado no es una vista, no se llama al View engine sino que el **action result** se ejecuta por si mismo. El **Result execution** en este caso es el que genera la respuesta al HTTP request.

Routing

El proceso de Routing es fundamental como ya hemos visto para el funcionamiento de nuestra web de tipo ASP.NET MVC ya que gracias a el podemos configurar el enrutamiento en el mismo.

Esto significa que si recibo un request proveniente de la URL http://miweb.com/lista/agregar/5 ASP.NET va a interpretar esa dirección gracias al routing configurado y entonces va a entender que el controlador **ListaController** va a atender la solicitud por medio de la acción **agregar** con el dato **5**. Una vez procesado también va a devolver un resultado con una vista o no al usuario final.

System.Web.Routing es el ensamblado encargado del enrutamiento y no es exclusivo de MVC si no que también es usado por otros componentes de ASP.NET tradicional.

Configuración del Routing

Veamos donde se encuentran los datos del routing en nuestra aplicación.

Global.asax: Aquí es donde se ejecuta la configuración de routing al comenzar la aplicación

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace NET2U7_MVC
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

RouteConfig.cs es llamado en el código anterior a registrar estas rutas para el usuario, en RouteConfig.cs, que se encuentra en la carpeta **App_Start** vamos a poder ver, modificar y agregar o quitar rutas a nuestro antojo:

```
public class RouteConfig
   {
       public static void RegisterRoutes(RouteCollection routes)
       {
           routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

           routes.MapRoute(
               name: "Default",
               url: "{controller}/{action}/{id}",
               defaults: new { controller="Home", action="Index", id = UrlParameter.Optional }
           );
       }
   }
```

HomeController.cs finalmente va a ser quien atiende este request o solicitud del usuario por medio de la acción del routing de arriba “Index”:

```
public class HomeController : Controller
{
    // GET: PrimerEjemplo
    public string Index()
    {
        return "Hola Mundo MVC";
    }
}
```

En **RouteConfig** puedo agregar otras rutas simplemente agregando el código de la siguiente manera:

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller="Home", action="Index", id = UrlParameter.Optional }
);

routes.MapRoute(
    name: "Usuario",
    url: "Usuario/{nombre}",
    defaults: new { controller = "Usuario", action = "Buscar", nombre = UrlParameter.Optional }
);
```

nótese como puedo modificar la estructura del routing: no siempre debe ser `{controller}/{action}/{id}` también puedo hacer algo como `Usuario/{nombre}` y la acción va a estar implícita en la declaración, resultando en una URL mucho más corta.

Controllers

Los controladores o controllers son el componente principal en una web MVC. Son quienes deciden que modelo se va a utilizar, luego que datos del modelo son los apropiados para tomarlos procesarlos (o no) y devolverlos al usuario por medio de una respectiva view o vista.

Básicamente los controllers se encargan del flujo de la aplicación tomando la entrada procesandola y devolviendo una salida esperada.

Implementación de controladores

Los controllers no son más que clases C# en .NET que heredan sus características del controlador base que está codificado en el ensamblado **System.Web.Mvc.Controller**

En MVC los métodos públicos del controlador se denominan **action method** o métodos de acción, lo que significa que pueden ser invocados desde la web por medio de una URL.

Tomemos el caso anterior del usuario e implementemos un controlador para ese caso:

```
routes.MapRoute(
    name: "Usuario",
    url: "Usuario/{nombre}",
    defaults: new { controller = "Usuario", action = "Buscar", nombre = UrlParameter.Optional }
);
```

Lo que se busca es crear un controlador que tome el nombre que le estoy pasando por parámetro, lo procese y me devuelva información pertinente al usuario.

Por lo tanto nuestro controlador usuario se va a ver así:

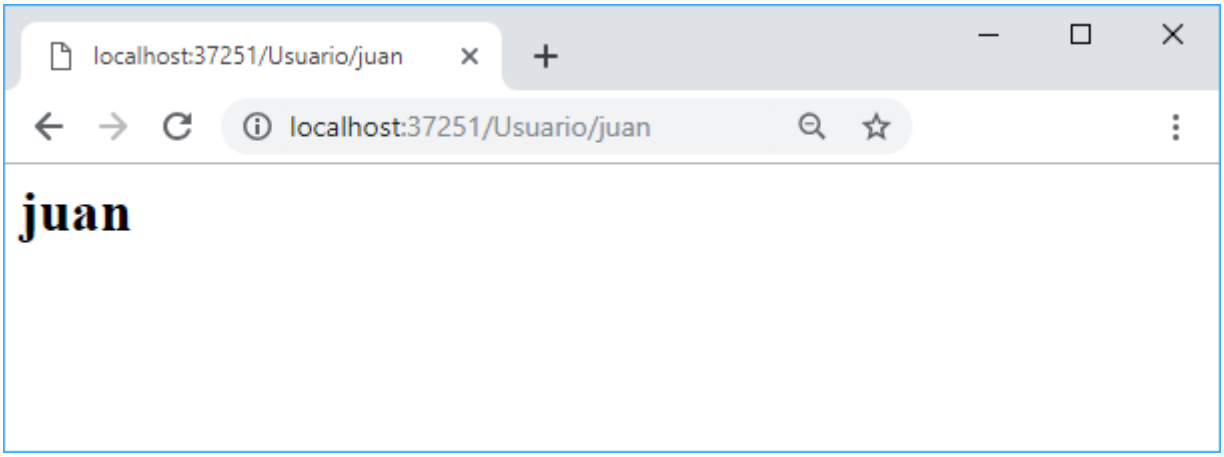
```
public class UsuarioController : Controller
{
    // GET: Usuario
    public ActionResult Buscar(string nombre)
    {
        var input = Server.HtmlEncode(nombre);
        input = "<h1>" + input + "<h1>";
        return Content(input);

    }
}
```

Al asignarle un parámetro al método obliga a ASP.NET MVC a buscar un valor que corresponda al mismo en los datos del request. El método luego utiliza ese dato para procesarlo o hacer una búsqueda y luego devolverlo al cliente

En este caso estoy usando una acción de tipo “Content” y devuelvo el nombre formateado como html, en la próxima sección vamos a ver otro tipo de acciones.

La función **HTMLEncode** me ayuda a evitar que alguien quiera inyectar código maligno en mi Web, es altamente recomendable que siempre la utilicen al obtener datos desde una URL.

Finalmente el resultado es el siguiente:



Actions

Los métodos Action MVC de ASP.NET son responsables de ejecutar las solicitudes y generar respuestas a ellas. Por defecto, generan una respuesta en forma de ActionResult. Las acciones suelen ser uno a uno con las interacciones del usuario.

Por ejemplo, ingresamos una URL en el navegador web o hacemos click en un link o botón; Esto va a hacer que se envíe una solicitud al servidor. Cada una de estas interacciones va a tener información que el Framework MVC utiliza para invocar una acción.

La única restricción del método es que no debe ser estático pero no hay restricciones en su valor de retorno, puede devolver strings, integers etc.

Tipos de acciones

Las Acciones básicamente devuelven distintos tipos de **Action Results.** Esta clase **ActionResult** es la base para todos los demás objetos que heredan de ella, a continuación veamos cuales son y que representan.

ContentResult	Devuelve un string
FileContentResult	Devuelve el contenido de un archivo
FilePathResult	Devuelve el contenido de un archivo por medio de su ubicación
FileStreamResult	Devuelve el contenido de un archivo en formato stream

EmptyResult	No devuelve nada
JavaScriptResult	Devuelve un script de Javascript para su ejecución.
JsonResult	Devuelve datos formateados en tipo JSON
RedirectToResult	Nos redirecciona a una URL especificada.
HttpUnauthorizedResult	Devuelve el código HTTP 403 de acceso no autorizado
RedirectToRouteResult	Nos redirecciona a una acción o control diferente
ViewResult	Devuelve el resultado en tipo Vista.
PartialViewResult	Devuelve el resultado en tipo Vista.

Utilicemos el ejemplo anterior para utilizar unas de estas acciones: **RedirectToAction**

Básicamente lo que estoy haciendo es hacer que redirija por defecto todo el trafico que cae en **Index** de mi controlador **Home** a mi método de acción **Buscar** en el Controlador **Usuario**. Nótese como le estoy pasando parámetros.

```
public class HomeController : Controller
{
   public ActionResult Index()
   {
       return RedirectToAction("Buscar", "Usuario", routeValues: new { nombre = "UsuarioDeEjemplo" });
   }
}
```

Hasta ahora hemos cubierto nociones básicas de estructura, configuración y funcionamiento de un sistema ASP.NET MVC. También hemos cubierto los temas de Routing, Controllers y Acciones.

Con lo que conocemos hasta ahora podemos construir una aplicación ASP.NET MVC con código C# para realizar acciones y una visualización simple. Nos falta todavía trabajar con las vistas (páginas web) y los datos (modelos) para poder cerrar todos los conceptos y crear páginas web profesionales con MVC.