Como Empezar

# Requerimientos

1. Windows XP, Vista, Server 2003, Server 2008 o Windows 7.
2. Visual Studio 2008 con SP1 (cualquier edición), Free Visual Web Developer 2008 Express con SP1, Visual Studio 2010 (cualquier edición) o Free Visual Web Developer 2010. No se puede compilar aplicaciones ASP.NET MVC 2 con Visual Studio 2005.

# Instalación

Descargar ASP.NET MVC 2 desde cualquiera de las siguientes direcciones:

[Download ASP.NET MVC 2 using the Microsoft Web Platform Installer](http://go.microsoft.com/fwlink/?LinkID=185037)

[Download ASP.NET MVC 2 from the Download Center](http://go.microsoft.com/fwlink/?LinkID=157074)

\*Nota: Esta instalación solo es obligatoria si utilizas VS 2008

# Herramientas Adicionales

Para este curso utilizaremos SqlServer 2008 Express, lo pueden descargar desde la siguiente dirección:

<http://www.microsoft.com/downloads/details.aspx?FamilyId=7522A683-4CB2-454E-B908-E805E9BD4E28&displaylang=en>

¿Qué es ASP.NET MVC?

# ¿Qué es el Modelo-Vista Controlador?

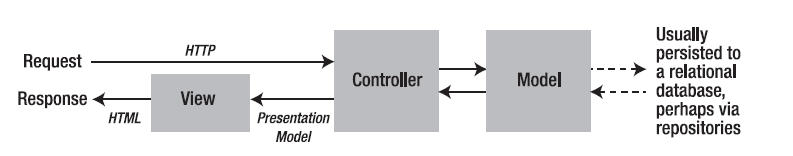
Es un patrón de arquitectura que se usa para separar una aplicación en por lo menos 3 partes principales:

Vista.- Es la parte encargada de mostrar la interfaz de usuario.

Modelo.- No tiene una definición única y esta depende de la forma en la cual diseñemos nuestra aplicación:

* Conjunto de clases que han sido diseñadas para satisfacer las necesidades de una vista en particular. En este caso, las clases toman el nombre de “ViewModels”.
* Conjunto de clases que contienen las reglas de negocio y tienen la información de cómo los datos serán cambiados y manipulados.

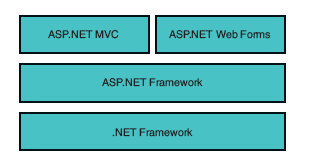
Controlador.- Es el encargado de recibir y manejar las peticiones realizadas por los usuarios, y mostrar la vista adecuada.

Arquitectura MVC[[1]](#footnote-1)

# ¿Qué es ASP.NET MVC?

ASP.NET MVC es una framework de desarrollo web que combina la efectividad y ventajas de una arquitectura modelo-vista-controlador, las ideas y técnicas del desarrollo ágil y las mejores partes de la plataforma ASP.NET existente. Es una alternativa completa al desarrollo tradicional con WebForms que nos provee ventajas considerables frente a este.

La framework MVC está construida sobre el core de ASP.NET lo que permite seguir utilizando muchas de sus características (Master Pages, Membership, etc) y se encuentra dentro del Namespace *System.Web.Mvc*.

  
ASP.NET Frameworks [[2]](#footnote-2)

# Implementación MVC en ASP.NET MVC

En ASP.NET MVC, los controladores son clases .NET, usualmente derivadas de la clase base “Controller”. Cada método público del controladores es llamado “action method” que es automáticamente asociado con una URL mediante un esquema de configuración y además puede mostrar la vista que considere adecuada.

ASP.NET MVC suporta múltiples “view engines” (interpretan el código de una vista y lo traducen en código html). Por defecto se utiliza ASP.NET WebForms como View Engine lo que nos permite escribir html y código C# para injectar y responder los datos enviados desde el controlador.

ASP.NET MVC no provee una infraestructura en particular para el modelo, sin embargo existen una gran cantidad de frameworks y herramientas que están a nuestra disposición.

# ASP.NET “Tradicional” - WebForms

ASP.NET significó un gran cambio cuando apareció por primera vez, no solo en términos del nuevo .NET Framework sino en que pretendía cerrar la brecha entre el desarrollo de Windows Forms (statefull) y el desarrollo web - HTML (stateless).

Se intentó ocultar toda la complejidad que significaba el manejo y conocimiento del HTTP y el HTML (en ese momento desconocido para muchos desarrolladores), mediante el modelado de la interfaz de usuario a través de una jerarquía de controles en el lado de servidor, que automáticamente se renderizaban como HTML y se conectaban con las acciones realizadas en el cliente, mediante eventos que se ejecutaban también en el lado del servidor.

Los desarrolladores no tenían nunca más que preocuparse de las llamadas y respuestas HTTP y podrían pensar en términos de una UI que mantenía su estado.

WebForms representa una excelente capa de abstracción para trabajar con una clásica UI orientada a eventos sobre la Web.

# ¿Cuál es el problema con el uso de WebForms?

WebForms fue una buena idea, se veía muy prometedor en el principio pero a medida que paso el tiempo resultó ser más complicado. Al pasar los años se fueron encontrando una serie de deficiencias:

* ViewState: Es el mecanismo de mantener el estado entre las solicitudes HTTP, a menudo en gigantescos bloques de datos que se transfieren entre el cliente y el servidor en cada solicitud. Esto puede significar que los visitantes de un sitio tengan una larga espera cada vez que realizan una acción en el cliente como hacer clic en un botón.
* Page Life Cycle: Es el mecanismo mediante el cual se conectan los eventos realizados en el cliente con los eventos realizados en el servidor, la manipulación de estos eventos puede ser extraordinariamente complejo y delicado, y pocos desarrolladores tienen éxito en su manejo.
* Limitado control sobre el HTML: Los controles de servidor se encargan de renderizarme a sí mismos como HTML pero no necesariamente el HTML que se quiere. Muchas veces este HTML no cumple con los estándares o no hacen buen uso del CSS.
* Falsa sensación de separación de responsabilidades: ASP.NET “code-behind” proporciona un medio para separar el código C# de la aplicación en una nueva clase separándola del HTML, pero en realidad se alienta a los desarrolladores a mezclar el código de la presentación con la lógica de la aplicación lo que produce código muy frágil y difícil de entender. Recuerden diferentes archivos no necesariamente significa diferentes responsabilidades.
* Casi imposible de probar: cuando WebForms fue inicialmente diseñado, no se tenía idea que los test automatizados se volvería un parte tan necesaria e importante dentro del desarrollo de software como lo es hoy en día. Es por esto que no sorprende que su diseño haga casi imposible el uso de test automatizados.

# Principales beneficios de ASP.NET MVC

**Key Benefits of ASP.NET MVC**

A huge corporation like Microsoft can afford to rest on its laurels for a while, but not forever.

ASP.NET has been a great commercial success so far, but as discussed, the rest of the web

development world has moved on, and even though Microsoft has kept dusting the cobwebs

off WebForms, its essential design has started to look quite antiquated.

In October 2007, at the very first ALT.NET conference in Austin, Texas, Microsoft vice

president Scott Guthrie announced and demonstrated a brand-new MVC web development

platform, built on ASP.NET, clearly designed as a direct response to the criticisms laid out previously.

Here’s how it overcomes ASP.NET’s limitations and brings Microsoft’s platform back to

the cutting edge.

**Model-View-Controller Architecture**

ASP.NET MVC provides greatly improved separation of concerns thanks to its adoption of

MVC architecture. The MVC pattern isn’t new—it dates back to 1978 and the Smalltalk project

at Xerox PARC—but it’s gaining enormous popularity today as an architecture for web applications,

perhaps because of the following:

• User interaction with an MVC application naturally follows a cycle: the user takes an

action, and then in response the application changes its data model and delivers an

updated view to the user. And then the cycle repeats. This is a very convenient fit for

web applications delivered as a series of HTTP requests and responses.

• Web applications already necessitate combining several technologies (e.g., databases,

HTML, and executable code), usually split into a set of tiers or layers, and the patterns

that arise naturally map onto the concepts in MVC.

ASP.NET MVC implements a modern variant on MVC that’s especially suitable for web

applications. You’ll learn more about the theory and practice of this architecture in Chapter 3.

Through this design, ASP.NET MVC directly answers the competition of Ruby on Rails and

similar platforms, making a serious effort to bring this style of development into the mainstream

of the .NET world, capitalizing on the experience and best practices discovered by developers

using other platforms, and in many ways pushing forward beyond what even Rails can offer.

CHAPTER 1 **n** WHAT’S THE BIG IDEA? **7**

**Extensibility**

Your desktop PC’s internal components are independent pieces that interact only across

standard, publicly documented interfaces, so you can easily take out your graphics card or

hard disk and replace it with another one from a different manufacturer, confident that it will

slot in and work. In just the same way, the MVC Framework is built as a series of independent

components—satisfying a .NET interface or built on an abstract base class—so you can easily

replace the routing system, the view engine, the controller factory, or any other framework

component, with a different one of your own implementation. In fact, the framework’s

designers set out to give you three options for each MVC Framework component:

**1.** Use the *default* implementation of the component as it stands (which should be

enough for most applications).

**2.** Derive a *subclass* of the default implementation to tweak its behavior.

**3.** *Replace* the component entirely with a new implementation of the interface or abstract

base class.

It’s like the Provider model from ASP.NET 2.0, but taken much further—right into the heart

of the MVC Framework. You’ll learn all about the various components, and how and why you

might want to tweak or replace each of them, starting with Chapter 7.

**Testability**

MVC architecture gives you a great start in making your application maintainable and

testable, because you will naturally separate different application concerns into different,

independent software pieces.

Yet the ASP.NET MVC designers didn’t stop there. They took the framework’s componentoriented

design and made sure each separate piece is ideally structured for automated testing.

So, you can write clean, simple unit tests for each controller and action in your application,

using fake or mock implementations of framework components to simulate any scenario. The

framework’s design works around the limitations of today’s testing and mocking tools, and

adds Visual Studio wizards to create starter test projects on your behalf (integrating with open

source unit test tools such as NUnit and MBUnit as well as Microsoft’s MSTest), so even if

you’ve never written a unit test before, you’ll be off to a great start. Welcome to the world of

maintainable code!

Throughout this book, you’ll see examples of how to write automated tests using a variety

of testing and mocking strategies.

**Tight Control over HTML**

The MVC Framework recognizes the importance of producing clean, standards-compliant

markup. Its built-in HTML helper methods do of course produce XHTML-compliant output,

but there’s a bigger change of mindset at work. Instead of spewing out huge swathes of barely

readable HTML code to represent what should be simple UI elements like lists, tables, or string

literals, the MVC Framework encourages you to craft simple, elegant markup styled with CSS.

(Plus, Visual Studio 2008’s massively improved CSS refactoring support finally makes it possible

to keep track of and sensibly reuse your CSS rules no matter how big your project gets.)

**8** CHAPTER 1 **n** WHAT’S THE BIG IDEA?

Of course, if you do want to throw in some ready-made widgets for complex UI elements

like date pickers or cascading menus, ASP.NET MVC’s “no special requirements” approach to

markup makes it dead easy to use best-of-breed open source UI libraries such as jQuery or the

Yahoo UI Library. Chapter 12 of this book demonstrates many of these techniques in action,

producing rich, cross-browser interactivity with a minimum of fuss. JavaScript developers will

be thrilled to learn that ASP.NET MVC meshes so well with the popular jQuery library that

Microsoft ships jQuery as a built-in part of the default ASP.NET MVC project template.

ASP.NET MVC–generated pages don’t contain any ViewState data, so they can be hundreds

of kilobytes smaller than typical pages from ASP.NET WebForms. Despite today’s fast broadband

connections, this bandwidth saving still gives an enormously improved end user experience.

**Powerful New Routing System**

Today’s web developers recognize the importance of using clean URLs. It isn’t good for

business to use incomprehensible URLs like /App\_v2/User/Page.aspx?action=show%20prop&

prop\_id=82742—it’s far more professional to use /to-rent/chicago/2303-silver-street.

Why does it matter? Firstly, search engines give considerable weight to keywords found in

a URL. A search for “rent in chicago” is much more likely to turn up the latter URL. Secondly,

many web users are now savvy enough to understand a URL, and appreciate the option of

navigating by typing into their browser’s address bar. Thirdly, when someone feels they can

understand a URL, they’re more likely to link to it (being confident that it doesn’t expose any

of their own personal information) or share it with a friend (perhaps reading it out over the

phone). Fourthly, it doesn’t pointlessly expose the technical details, folder, and file name

structure of your application with the whole public Internet (so you’re free to change the

underlying implementation without breaking all your incoming links).

Clean URLs were hard to implement in earlier frameworks, but ASP.NET MVC uses the

brand-new System.Web.Routing facility to give you clean URLs by default. This gives you total

control over your URL schema and its mapping to your controllers and actions, with no need

to conform to any predefined pattern. Of course, this means you can easily define a modern

REST-style URL schema if you’re so inclined.

You’ll find a thorough treatment of routing and URL best practices in Chapter 8.

**Built on the Best Parts of the ASP.NET Platform**

Microsoft’s existing platform provides a mature, well-proven suite of components and facilities

that can cut down your workload and increase your freedom. Firstly and most obviously,

since ASP.NET MVC is based on the .NET 3.5 platform, you have the flexibility to write code in

any .NET language4 and access the same API features, not just in MVC itself, but in the extensive

.NET class library and the vast ecosystem of third-party .NET libraries.

Secondly, ready-made ASP.NET platformfeatures such as master pages, Forms Authentication,

membership, roles, profiles, and globalization can significantly reduce the amount of code

you need to develop and maintain in any web application, and these are just as effective in an

MVC project as in a classic WebForms project. Certain WebForms’ built-in server controls—and

your own custom controls from earlier ASP.NET projects—can be reused in an ASP.NET MVC

application (as long as they don’t depend on WebForms-specific notions such as ViewState).

CHAPTER 1 **n** WHAT’S THE BIG IDEA? **9**

4. You can even build ASP.NET MVC applications in IronRuby or IronPython, although most businesses

are likely to stick with C# and VB .NET for the time being. This book focuses exclusively on C#.

Development and deployment are covered, too. Not only is ASP.NET well integrated into

Visual Studio, Microsoft’s flagship commercial IDE, it’s *the* native web programming technology

supported by the IIS web server built into Windows XP, Vista, 7, and Server products. IIS

7.0 adds a set of enhanced features for running .NET managed code as part of the request

handling pipeline, giving special treatment to ASP.NET applications. Being built on the core

ASP.NET platform, MVC applications get an equal share of the benefits.

Chapter 14 explains what you need to know to deploy ASP.NET MVC applications to IIS

on Windows Server 2003 and Server 2008. Chapter 15 demonstrates the core ASP.NET platform

features you’re likely to use in an MVC application, showing any differences in usage between

MVC and WebForms applications, along with tips and tricks needed to work around compatibility

issues. Even if you’re already a seasoned ASP.NET expert, there’s a good chance you’ll

find one or two useful components you haven’t yet used.

**.NET 3.5 Language Innovations**

Since its inception in 2002, Microsoft’s .NET platform has evolved relentlessly, supporting and

even defining the state-of-the-art aspects of modern programming. The most significant recent

innovation is *Language Integrated Query (LINQ)*, along with bucketloads of ancillary enhancements

in C# such as lambda expressions and anonymous types. ASP.NET MVC is designed with

these innovations in mind, so many of its API methods and coding patterns follow a cleaner,

more expressive composition than was possible when earlier platforms were invented.

**ASP.NET MVC Is Open Source**

Faced with competition from open source alternatives, Microsoft has made a brave new move

with ASP.NET MVC. Unlike with any previous Microsoft web development platform, you’re

free to download the original source code to ASP.NET MVC, and even modify and compile

your own version of it. This is invaluable for those occasions when your debugging trail leads

into a system component and you want to step into its code (even reading the original programmers’

comments), and also if you’re building an advanced component and want to see

what development possibilities exist, or how the built-in components actually work.

Of course, this ability is also great if you don’t like the way something works, find a bug, or

just want to access something that’s otherwise inaccessible, because you can simply change it

yourself. However, you’ll need to keep track of your changes and reapply them if you upgrade

to a newer version of the framework. Source control is your friend here.

ASP.NET MVC has been licensed under Ms-PL (www.opensource.org/licenses/ms-pl.html),

an OSI-Approved open source license, which means you can change the source code, deploy

it, and even redistribute your changes publicly as a derivative project. However, at present

Microsoft is *not* accepting patches to the central, official build. Microsoft will only ship code

that’s the product of their own development and QA teams.

You can download the framework’s source code from <http://tinyurl.com/cs3l3n>.

# Cuando usar ASP.NET MVC o Web Forms

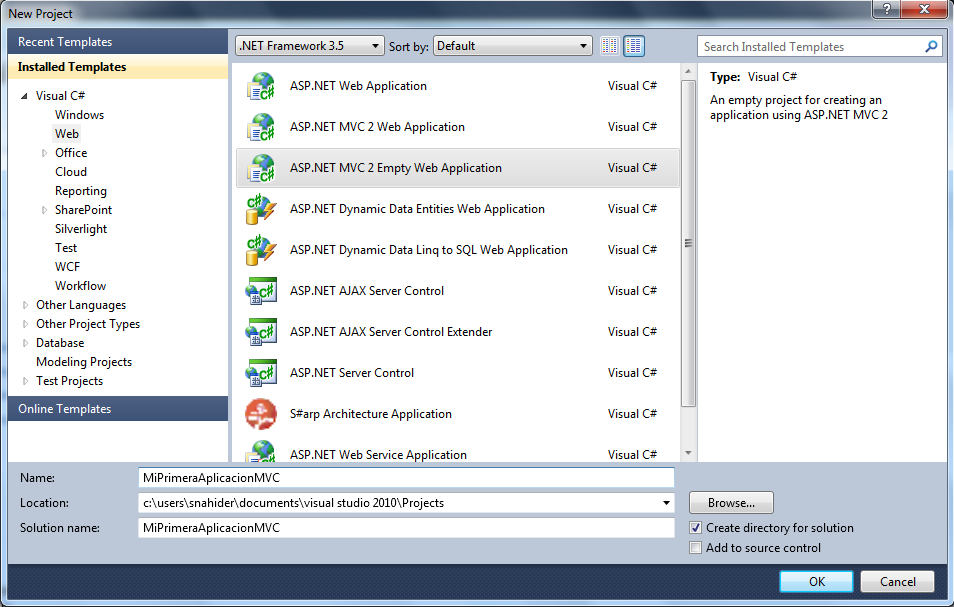
Nuestra Primera Aplicación MVC

# Creando Nuestro Proyecto

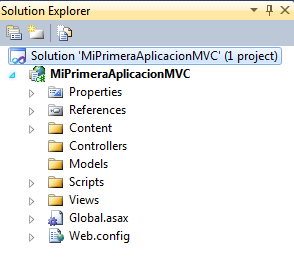
Una vez que tenemos instalado ASP.NET MVC 2 tenemos 2 templates con los que podemos comenzar un nuevo proyecto:

* ASP.NET MVC 2 Web Application: Proyecto en el que ya viene incluido funcionalidad de registro, login, navegación y una hoja de estilos.
* ASP.NET MVC 2 Empty Web Application: Contiene solo los mínimos archivos y directorios necesarios para comenzar la mayoría de las aplicaciones ASP.NET MVC 2.

Para lograr un mejor entendimiento de cómo trabaja realmente la framework comenzaremos utilizando la plantilla en blanco.



Tendremos la siguiente estructura de proyecto:

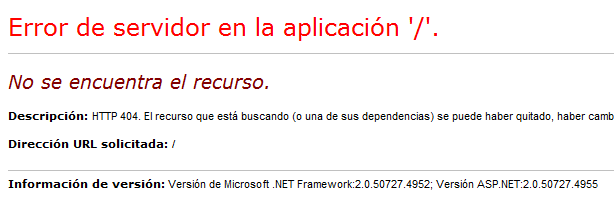


Existen 3 directorios principales (Controllers, Models,Views) donde colocaremos las clases que conformarán nuestra aplicación MVC.

Nosotros no tenemos que decir en ninguna parte cuales son los nombres de estas carpetas, simplemente se espera que estén ahí por convención.

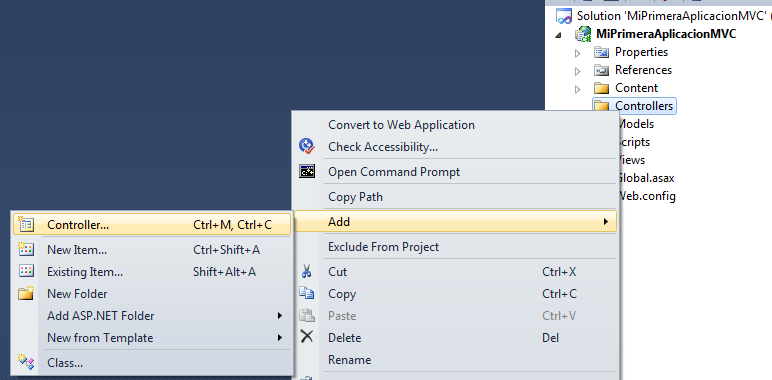
Entonces aquí vemos que desde la estructura del proyecto ASP.NET MVC aplica un importante concepto llamado: Convention over Configuration, con el cual no tenemos que pre-configurar nada sino seguir ciertos lineamientos conocidas por todos (convenciones) y todo funcionará perfectamente.

Ahora si ejecutamos F5 veremos lo siguiente debido a que no existe ningún controller y estas son las clases encargadas de recibir todos los request o peticiones hacia la aplicación.

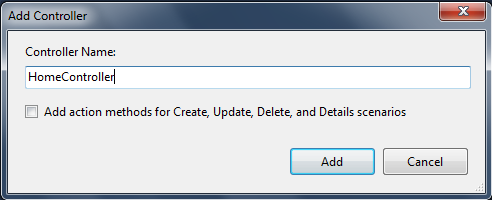


# Agregando nuestro primer Controller

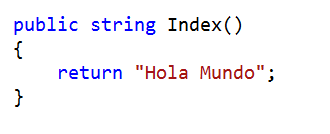
Si recordamos, por convención todos nuestros controladores se deben encontrar en la carpeta controllers, entonces procedemos a crear el primero:



En este punto es donde nos encontramos con una segunda convención: El nombre de cada controlador debe terminar con el postfijo “Controller”, entonces a este controlador le damos por nombre “HomeController”.

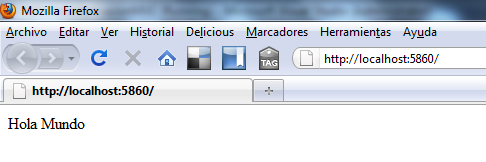


Borramos el código por defecto y escribimos lo siguiente:



Dentro del controlador, por convención cualquier método público es una acción. Los request llegan desde el cliente, estas son delegadas a un controller y este lo delega una acción, que será el método encargado de procesar la petición.

Ejecutamos F5 y veremos que el mensaje sale en la pantalla.



# Como sabe que se debe invocar el HomeController

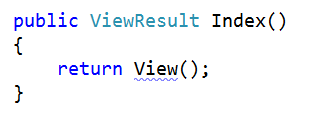
Lo hace en base a un sistema de enrutamiento que decide cual es el controlador y action que tomará el pedido de acuerdo a un mapeo con la URL.

Si no le indicamos nada a través de la URL, la configuración por defecto establece que la petición llegará al HomeController y a la acción Index. Asimismo, el mapeo por defecto es: [http://[Controller]/[Action](http://[Controller]/%5bAction)].

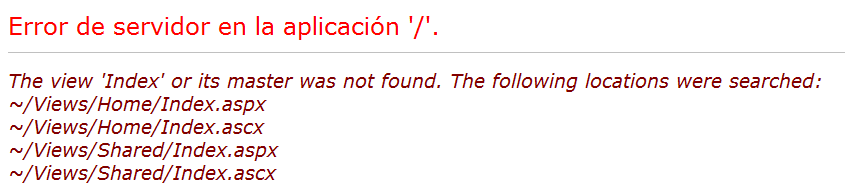
Entonces las siguientes peticiones serán procesadas por la acción Index del HomeController: http://, http://Home y <http://Home/Index>.

# Creando una Vista

Actualmente nuestro controller envia un cadena al navegador, pero en aplicaciones reales lo más probables es que enviemos HTML, entonces necesitamos indicarle a nuestra acción que vamos a devolver una vista, para esto escribimos el siguiente código:

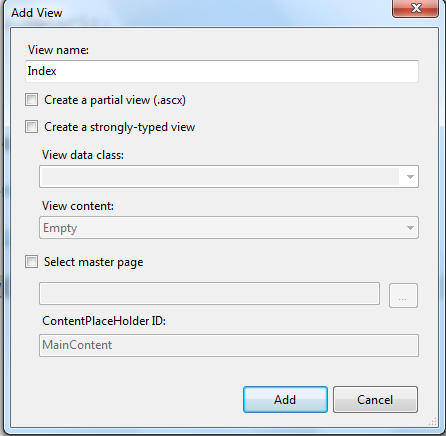


Retornando un ViewResult a través de un View sin parámetros es la forma más sencilla de indicarle a nuestra acción que vamos a devolver una vista. Existen otros objetos que podemos devolver desde la acción para indicar diferentes cosas, a todos estos objetos se les denomina “accion results” y derivan del tipo base ActionResult. Ahora ejecutamos la solución F5.

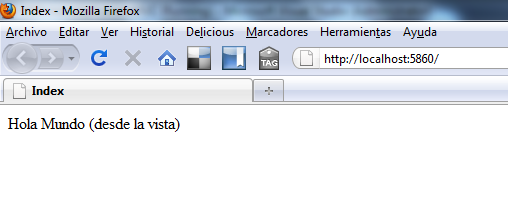


Si observamos bien el error nos indica que nuestra acción intentó devolver una vista, pero que esta no existe, también nos indica otra cosa muy importante y es los lugares por donde buscó la vista. Indicamos que por convención las vistas se deben encontrar en la carpeta Views y se relacionan con las acciones a través de una convención de nombres. Cuando la framework busca la vista por defecto para la acción Index del HomeController esta será buscada en los directorios de la figura anterior según el patrón: Views/[Controller]/[Action].

Para agregar la vista creamos la carpeta Home dentro de Views y damos clic derecho sobre esa carpeta y seleccionamos “Add View”.



Veremos que nos creará un archivo con contenido html, escribimos en el “Hola Mundo (desde la vista)” y ejecutamos con F5.

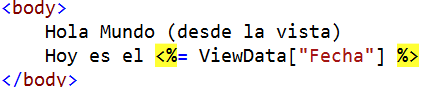


# Enviando datos dinámicos

El controller es el encargado de construir los datos que serán enviados a la vista para que los muestre como Html. ASP.NET MVC nos ofrece muchas formas de enviar datos a la vista, una de ellas es utilizando la estructura de datos ViewData.

Escribimos los siguiente en nuestra acción y en nuestra vista.

## 



Al ejecutar la aplicación veremos el siguiente mensaje:

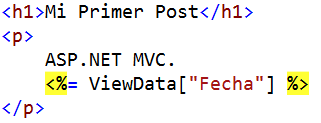


# Aplicación Inicial

Para seguir explorando algunas características de la framework vamos a agregar una pequeña funcionalidad para que agregar comentarios sobre un artículo en una página web.

* Mostrar en la página principal un artículo.
* Tener un formulario donde se puedan ingresar comentarios y que muestre un mensaje de agradecimiento al finalizar el envio.

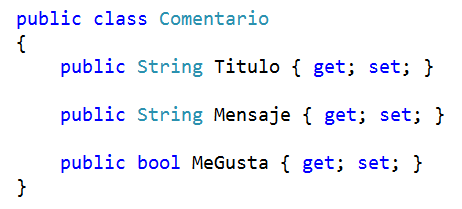
## Modificamos nuestra vista



## Diseñamos nuestro modelo.

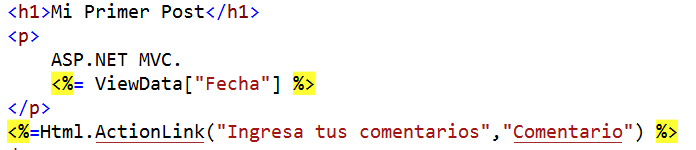
Podríamos seguir agregando más vistas y controllers pero en este momento vamos a pensar acerca de la aplicación que estamos construyendo.

En MVC la M representa el modelo que es la representación lógica dentro del software de los objetos, reglas y procesos que existen en el mundo real, es decir representa el dominio sobre el cuál va a trabajar la aplicación. Dentro de nuestro contexto, el dominio para la aplicación es bastante simple y lo podemos modelar utilizando la siguiente clase.



## Llamando acciones desde la Vista

Para poder ingresar nuestros comentarios necesitamos que en la vista se muestre un formulario. Para esto colocaremos un link en la pantalla inicial que nos llevará hacia el formulario.

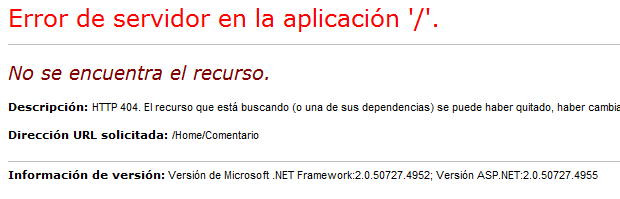


Si al ActionLink se le indica únicamente el nombre de la acción se inferirá el controller de acuerdo al Path donde se encuentra la vista.

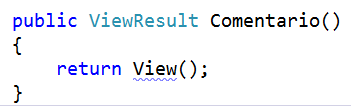
Al ejecutar el proyecto podremos ver la vista.



Pero la hacer click en el enlace veremos que nos devuelve un error de página no encontrada, si revisamos la URL: http://localhost:5860/Home/Comentario, veremos que intentó enviar la petición a la acción comentario y al controller Home. Recordar que en ASP.NET MVC las Urls no corresponde a recursos físicos sino a un controlador y una acción que se obtienen de acuerdo a un mapeo con la URL.

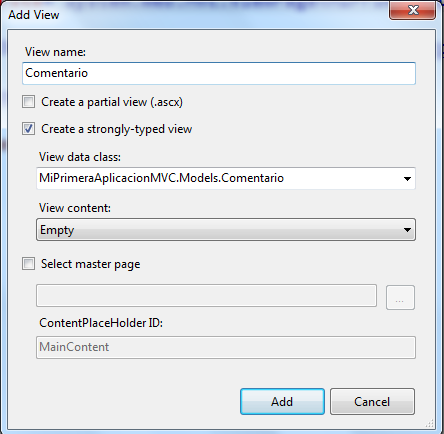


Para esto tendremos que agregar una nueva acción dentro del controller.



## Vistas fuertemente tipadas.

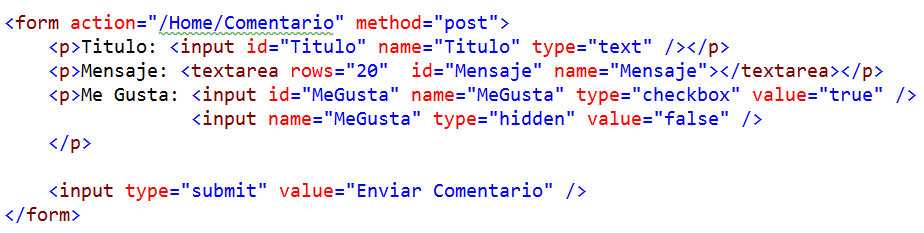
Nos falta agregar la vista, pero en este caso lo haremos diferente, a diferencia de la vista anterior cuyo objetivo era mostrar un conjunto diferente de cosas y en este caso queremos que la vista nos muestre un tipo específico de objeto. A este tipo de vistas se las denomina “Strongly Typed View”. Para esto creamos la vista de la siguiente manera:



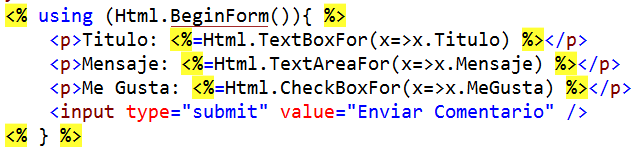
Podemos ver que a diferencia de la vista anterior, esta hereda de una clase diferente.



Podríamos crear nuestra vista de la siguiente manera.



Pero podríamos aprovechar que estamos utilizando una vista fuertemente tipada y utilizar los Html Helpers.



Podemos acceder directamente a las propiedades de nuestro objeto que queremos mostrar ya que se trata de una vista fuertemente tipada, sino esto no sería posible.

Si miramos el html generado veremos que es exactamente igual al caso anterior e inclusive la url del action del formulario se ha inferido de acuerdo al action del controller que muestra la vista.

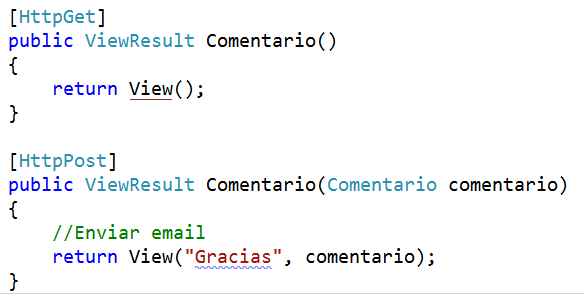
## Ingresando Datos

Si presionamos el botón veremos que nos devuelve la misma vista, esto debido a que la url del formulario es el action del controller que tiene como función mostrar la vista, entonces necesitamos otro action que se encargue de procesar los datos que vienen del formulario.

Previamente recordemos algo:

* Request por GET: Usualmente el navegador envía este tipo de petición cuando se presiona un link o se desea mostrar un formulario en blanco.
* Request por POST: Se utiliza usualmente para enviar datos desde los formularios para que estos sean procesados. El helper BeginForm envia datos utilizando este tipo de request.

ASP.NET MVC nos permite indicarle explícitamente a nuestras acciones si estas van a procesar request por POST o GET y esto lo hacemos utilizando los atributos de método [HttpGet] o [HttpPost]. Utilizando estos atributos procedemos a modificar nuestro controlador.



## Introducción a Model Binding

Nuetra primera acción es muy similar a la que ya conocíamos, pero la segunda es mucho más interesante ya que estamos pasando como parámetro un objeto Comentario con los datos provenientes del formulario.

Como es posible que la acción puede inferir que los datos pasados a través del formulario, esto es posible a través de característica denominada “Model Binding” que permite inferir el objeto de acuerdo a la colección de llaves/valores que vienen en el request y que corresponden a los nombres de los controles del formulario.

Con esta información construye el objeto, ingresando los valores del request en las propiedades que tienen el mismo nombre a la llave.

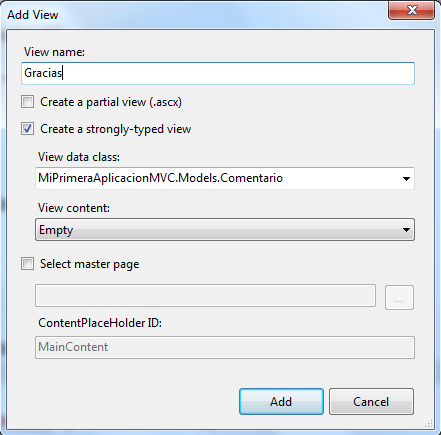
Esto permite evitar otras sentencias como Request.Form[] para obtener los valores del request.

## Indicando una vista de manera explícita y pasando objetos a la vista.

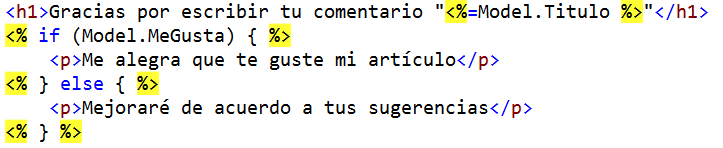
Si recordamos, llamamos al método View() sin parámetros este mostrará la vista que tenga el mismo nombre del Controller, pero también podemos indicar que vista mostrar de manera explícita, para esto le pasamos como parámetro el nombre de la vista.

Asimismo podemos pasar a través del método View el Modelo que queremos mostrar.

Debería existir una vista en la carpeta /Views/Home/Gracias pero aún no existe, entonces procedemos a crearla.



Escribimos el siguiente contenido.



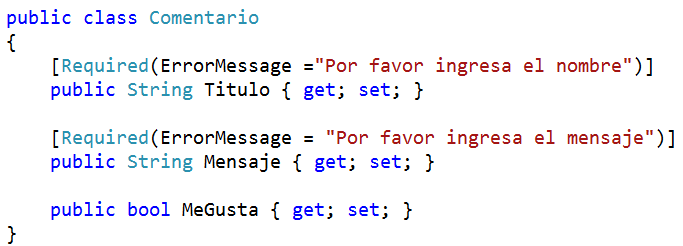
Otra de las facilidades que tenemos al utilizar una vista tipada es que podemos acceder al objeto de la vista a través de la propiedad Model.

Ahora podemos ejecutar la aplicación y ver que todo está funcionando.

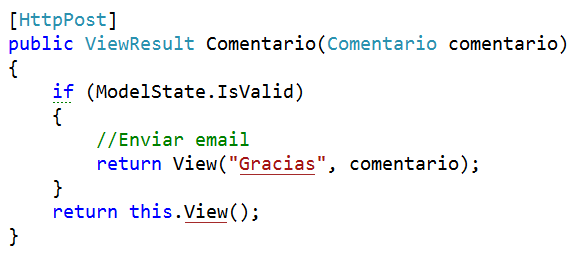
# Agregando validaciones

Hasta ahora hemos permitido el ingreso de cualquier contenido a través del formulario, entonces es momento de agregar ciertas validaciones. Sabemos que las validaciones pertenecen al dominio y no a la pantalla de usuario, por lo tanto debemos aplicar estas validaciones a nuestra clase Comentario.

Las librerías de .NET nos permiten indicar de manera declarativa ciertas validaciones en forma de atributos. Para esto modificamos nuestra clase Comentario de la siguiente manera.



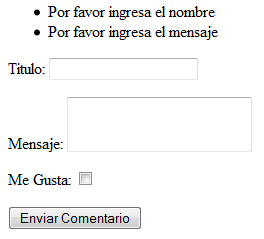
ASP.NET MVC reconocerá automáticamente estos atributos al momento de realizar el model binding y determinará si el objeto es válido o no. Agregamos la siguiente lógica en nuestro controller para que muestra nuevamente el formulario en caso haya algún error.



Para mostrar los errores en la vista, agregamos el siguiente helper dentro del BeginForm:



Probamos las validaciones sin ingresar ningún valor en las cajas de texto y vamos que nos devuelve los mensajes de error.



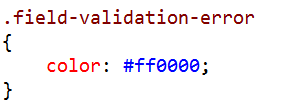
Ahora si nosotros ingresamos contenido solo para el título e intentamos enviar el formulario, observaremos que el error se muestra pero adicionalmente la caja de text aparece con el valor del título ingresado. Sabemos que ASP.NET MVC es “stateless” a diferencia de WebForms, entonces lo que permite este comportamiento es la utilización del ModelBinding que es capaz de devolver el mismo objeto a retornar a la vista.

## Resaltando los campos con error

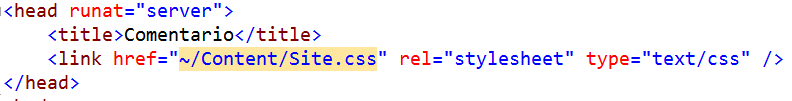
Si nosotros observamos el código fuente generado en el formulario con errores, observaremos que a las cajas de texto con error se les agrega la clase de estilos “input-validation-error”.



Entonces la forma más fácil de mostrar de manera diferente los controles donde se ha producido algún error es agregando una hoja de estilos. La plantilla de proyecto en blanco que hemos utilizado viene con una hoja de estilos por defecto que permiten mostrar de manera diferente las cajas de texto que tengan esta clase css.



Ahora agregamos la hoja de estilos al formulario y probamos la página.



# Resumen

La construcción de pequeña aplicación nos ha permitido dar una mirada rápida de cómo trabaja la arquitectura de MVC. En los siguientes capítulos veremos todo el potencial de la framework y la plataforma utilizando una aplicación más compleja.

ENTITY FRAMEWORK

# Database First

## Pasos Previos

Crear una bd.  
Ejecutar los scripts.

## Ejemplo

* Creamos un nuevo proyecto con solución “EntityFramework” y nombre de proyecto “DbFirst”.
* Eliminamos la clase por defecto.
* Agregamos nuevo elemento “Entity Data Model” con el nombre de Model. La primera ventana nos dice si queremos crearlo desde una bd ya existente o desde un modelo en blanco. Seleccionamos el primero.
* Luego agregamos la cadena de conección “Entities”. Y no solo agregamos el nombre de la cadena sino el nombre del entity container, que realmente será una clase instanciada que contendrá a todas las entidades y sus asociaciones. Accederemos muy seguido a esta clase, así que cambiamos de nombre
* Ahora nos mostrará el contenido de nuestra BD, seleccionamos los objetos que queremos. También cambiamos el namespace del modelo a “Model” y pluralizamos los nombres de nuestras clases.
* Cuanto termina la creación lo que vemos es la representación gráfica de EDM que se ha creado.
* Ahora el diseñador nos creó el modelo en base a lo que creó en la bd. Podemos ver las 2 entidades con sus propiedades pero también observamos una asociación entre ellas 2 que ha sido creada en base a las constraints de la bd. En este caso una relación uno a muchos. Observamos que la relación está representada por una clase y una propiedad escalar.
* También tenemos las navigation properties que representan a las entidades relacionar a las cuales podemos navegar. Ejm: Desde el Post podemos acceder a todos sus comentarios o desde el comentario a su respectivo post.
* Una de las ventajas, de este modelo es que podemos eliminar, renombrar clases o propiedades.
* Este modelo también contiene metadata de como las entidades están relacionadas a la bd, esto lo podemos ver haciendo clic derecho sobre la entidad “Table Mapping”. En la parte derecha tenemos las columnas y en la izquierda las propiedades.
* Si observamos en el explorador de soluciones veremos 2 archivos el primero es un xml que contiene información sobre nuestro modelo y el segundo es un archivo de clases, EF nos genera automáticamente una clase por cada entidad del modelo y también para el entitycontainer que hablamos anteriormente. ( lo exploraremos luego)

Si abrimos el archivo de clases veremos todas nuestas clases se encuentran ahí y también nuestro container.

Si creamos un diagrama de clases, para navegar mejor, veremos que cada entidad hereda de entity object que es como EF puede manejarlas.

Si observamos una clase, veremos que tiene sus campos y propiedades, la mayoría correspondientes a la BD, pero también veremos que existen otros, principalmente usados para el trabajo interno de EF, por lo tanto estas clases no son completamente limpias (pero también tiene otros por ejemplo si miramos las referencias tendremos que también incluye las entity key references que son usadas por EF, en las colecciones veremos que estás son del tipo EntityCollection. Veremos que también existen varios métodos, que realmente son parciales por lo tanto podríamos agregar cierta lógica desde una clase externa a esta.)

Una clase que también observamos acá es el container, que viene a ser la representación de todo el modelo, a través de esta clase es como nosotros realizaremos los queries, actualizaciones, inserciones ,etc. Si vemos cada una de nuestras entidades está representada dentro del container a través de una colección Object Set.

Una cosa importante sobre este archivo es que cada vez que actualizemos el modelo, este archivo será completamente generado de nuevo y los cambios que hayamos realizado se borrarán. Es por esto que si nos damos cuenta todas estas clases son parciales, es decir que podemos definir una clase en 2 archivos, por lo tanto para poner lógica que vamos a modificar tendríamos que crearla en otra clase.

Pero donde se encuentra la información con la cual genera estas clases, esta se encuentra dentro del model.edmx que es un archivo xml. Lo podemos abrir haciendo clic derecho en él y open with xml.

Dentro de este xml podemos ver las 3 partes , de las cuales hablamos anteriormente, de las q está compuesto el edmx, dentro de la parte conceptual podemos observar la información con la cual se generan las clases.

* Tambien se agregó un app.config con un entity connection string que no es una cadena de conección normal y contiene información relevante a EF.

# Model First

## Ejemplo

* Crearemos el modelo correspondiente a los departamentos de una empresa y sus empleados.
* Creamos un nuevo proyecto “Model First” y agregamos un nuevo elemento Entity Data Model con Modelo en Blanco.
* Nos creará un espacio en blanco donde nosotros podemos diseñar nuestro modelo. Para esto podemos hacerlo desde el toolbox o con clic derecho.
* Por ejemplo vamos a crear una entidad con clic derecho y add new entity. Nos aparece un dialogo donde podemos ingresar los datos de la entidad.
* Si vemos las propiedades observamos algunos de los valores que hemos elegido pero también información correspondiente a la bd como por ejemplo que este campo será identity.
* Por ejemplo demos enter para crear una nueva propiedad, en sus propiedades podemos ingresar la información necesaria, la parte de facets corresponde a información de la bd y no de la entidad.

Agregamos otros escalar “age” del tipo int veremos que ciertas propiedades ya no se encuentran disponibles por su tipo.

Agregamos una entidad departamento y luego una asociación, nos pedirá ingresar datos, ingresar como 0 a 1 en departamento, dentro de la información nos pedirá si queremos navegaciones desde cada entidad y si queremos que nos cree ids juntos con la navegación.

Creamos una entidad Salesman sin id para probar la herencia, ingresamos un campo comisión con scale y precision. Luego en el vacio realizamos clic derecho y creamos una herencia y le ingresamos los datos correspondientes. Existen varias estrategias para llevar nuestra herencia a un modelo relacional de datos, la que utilizan EF por defecto es Table per Type, aprovechamos para cambiar el entitycontainer. Podemos observar esta configuración mirando las propiedades del modelo. Para poder cambiar esta estrategia podemos hacerlo manualmente o descargar un power pack. Por último colocamos el employees como abstracto.

* Para generar nuestra base de datos en base al modelo, realizamos clic derecho y generar base de datos, recordar en este punto que lo que realmente hace es crear nuestros scripts de bd y para lo cual también debe existir nuestra bd. Ingresamos los datos de conección.
* Observando los resultados, vemos que es un script válido para 2005..azure y también veremos nuestras tablas y el contenido de la tabla de la herencia y también los constraints que nos está generando. Al dar finalizar nos crea el script de bd, el app. Config con la cadena de conección y también las clases.
* Este script lo podemos pasar a las personas encargadas de BD o ejecutarlo nosotros mismos, esto depende un poco de nuestro proceso, para ejecutarlo damos clic derecho sobre y ejecutar. Luego de finalizado podemos observar en el server explorer que se ha ejecutado.

## Limitaciones

* No podemos actualizar el modelo con los cambios de la bd.
* SI actualizamos el modelo, se recreará completamente la bd.

# Linq to Entities

* Ejecutar el script data.
* Realizar los ejemplos de “todos los departamentos, un departamento,, solo empleados con departamentos, eager y lazy” y para l2entities y method extensions.
* Resaltar que existen otras formas de realizar queries y también utilizar procedures.

# Manipulando el Modelo

* Realizar los ejercicios de manipulación.
* No olvidar levantar el profiler y en cada “save" verificar el query.
* Antes de registrar algo recordar que el contexto debe conocer el nuevo objeto y por lo tanto lo agregamos.
* Al eliminar explicar que el contexto necesita tener el conocimiento del objecto para poder manipularlo es por esto que no podemos decir context.delete(id);
* Recordar que el contexto es el responsable del manejo de los objectos y por lo tanto tenemos que indicarle que los cambios se persistan en la bd.

1. Imagen sacada del libro PRO ASP.NET MVC / Steve Sanderson [↑](#footnote-ref-1)
2. Imagen sacada del libro ASP.NET MVC Framework Unleashed / Stephen Walther [↑](#footnote-ref-2)