Introducción a ASP.NET MVC

Introducción

- ASP.NET MVC es un framework para la creación de aplicaciones web que aplica el patrón **M**odel **V**iew **C**ontroller (Modelo-Vista-Controlador) del framework ASP.NET.

- ¿Qué es un **framework**?
  Siendo muy simple, es un esquema o patrón para el desarrollo y la implementación de una aplicación. El paradigma MVC separa en tu aplicación la gestión de los datos, las operaciones, y la presentación.
- ¿Qué ventajas tiene utilizar un 'framework'?
  - El programador no necesita plantearse una estructura global de la aplicación, sino que el framework le proporciona un esqueleto que hay que "rellenar".
  - Facilita la colaboración. Cualquiera que haya tenido que "pelearse" con el código fuente de otro programador (¡o incluso uno propio antiguo!) sabrá lo difícil que es entenderlo y modificarlo; por tanto, todo lo que sea definir y estandarizar va a ahorrar tiempo y trabajo a los desarrollos colaborativos.
  - Es más fácil encontrar herramientas (utilidades, librerías) adaptadas al framework concreto para facilitar el desarrollo.

El patrón MVC

- Model-View-Controller (MVC) ha sido un importante patrón de arquitectura desde hace muchos años.

- Es un modelo elegante para separar la lógica del acceso a datos a la lógica de interfaz de usuarios, por lo que se aplica muy bien en las aplicaciones web.

- Se puede encontrar el patrón MVC en Java y C++, en sistemas operativos Mac y Windows.

El patrón MVC

- **MVC** separa la interfaz del usuario de una aplicación en 3 aspectos principales:

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un archivo de texto o un registro en una base de datos).

- La **Vista**: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

Uso en aplicaciones web

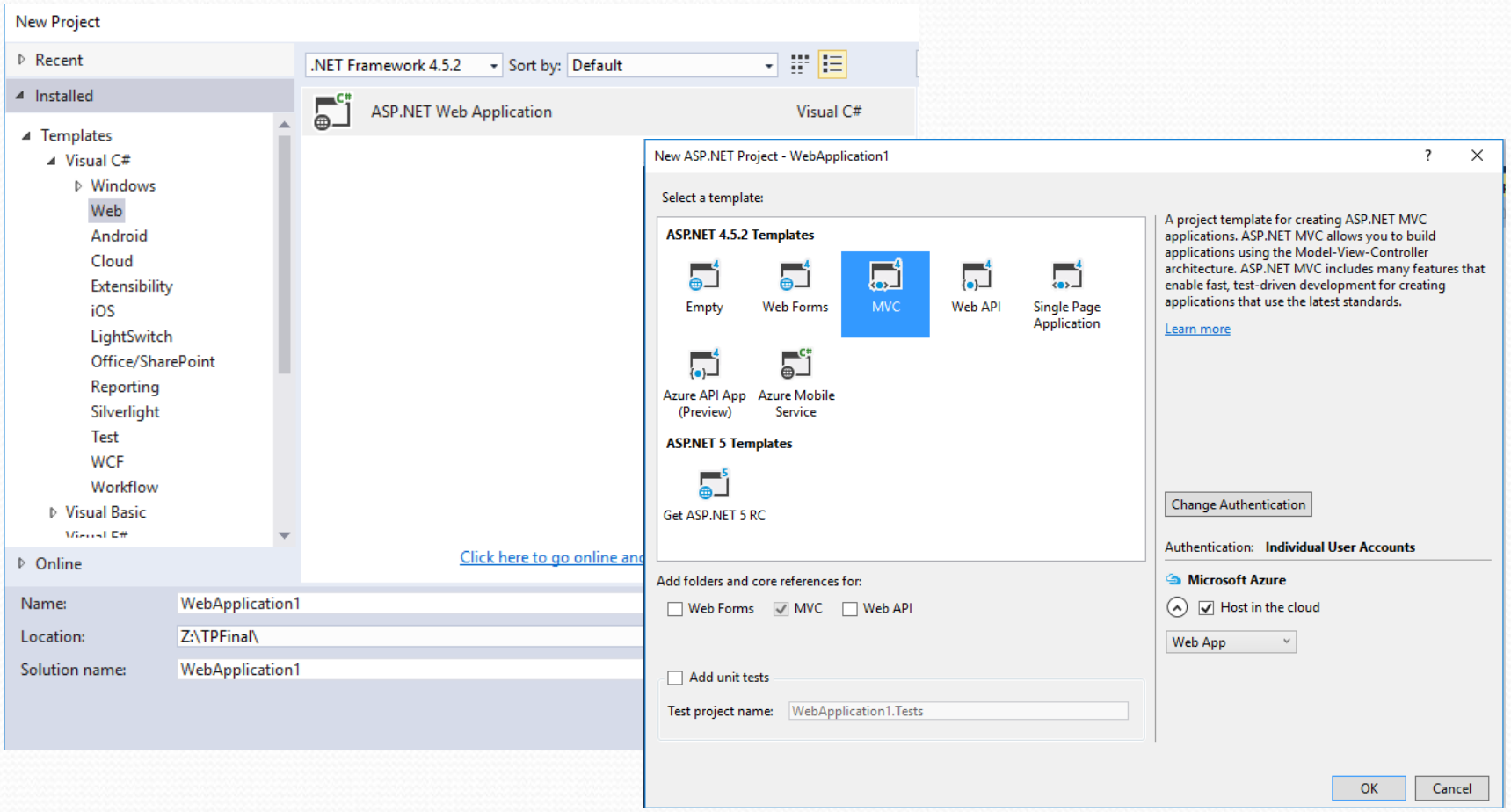
- Aunque originalmente **MVC** fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación.

- Los primeros frameworks MVC para desarrollo web planteaban un enfoque de sencillo en el que casi todas las funciones, tanto de la vista, el modelo y el controlador recaían en el servidor. En este enfoque, el cliente manda una petición de cualquier formulario al controlador y después recibe de la vista una página completa y actualizada.

- Tanto el modelo como el controlador (y buena parte de la vista) están completamente alojados en el servidor. Como las tecnologías web han madurado, ahora existen frameworks como JavaScriptMVC, Backbone o jQuery14 que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente (véase AJAX).

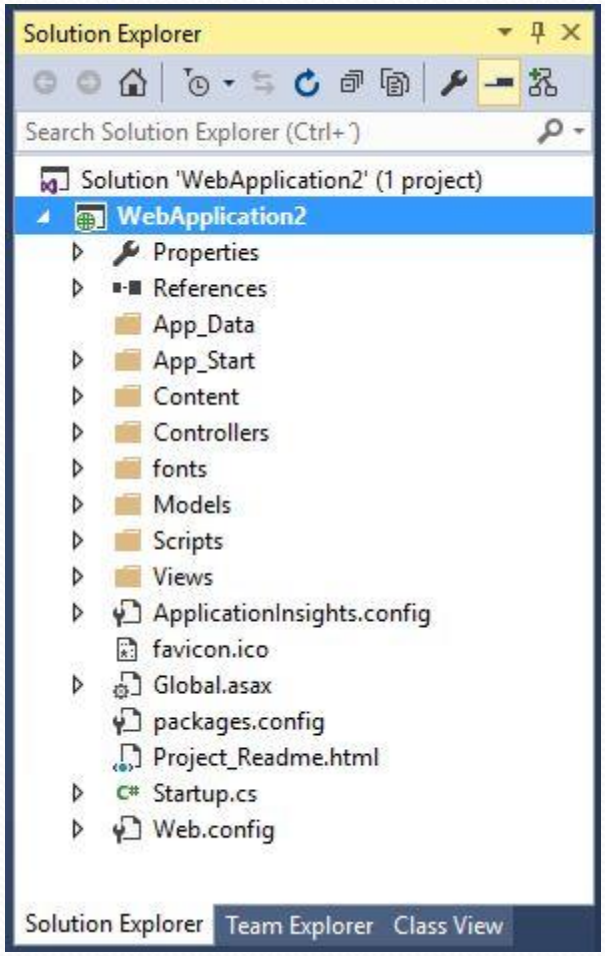
CREACIÓN DE UNA APLICACIÓN MVC5

- Crear el proyecto:
  File -> New -> Project -> ASP.NET MVC Web Application



Entendiendo la estructura

- /Controllers => Dónde se ponen las clases controladoras que manejan peticiones de URL
- /Models => Dónde se pone las clases que representan y manipulan los datos y objetos de negocio
- /Views => Dónde se ponen los archivos de plantillas de interfaz de usuario, tales como HTML
- /Scripts => Dónde se ponen los archivos JavaScript (.js)
- /Content => Dónde se ponen los CSS, imágenes y archivos públicos que no generen contenido dinámico
- /App_Data => Dónde se almacenan los archivos de texto o base de datos para almacenar la información



Controllers

- Los Controllers en el patrón MVC se encargan de **responder los inputs del usuario**, haciendo cambios en el Model. De esta manera, los controllers en MVC están **encargados del flujo de la aplicación**, el trabajo con los datos que se ingresan, y los datos que salen a la view que corresponde.

- Ejemplo:

  - **HomeController**: Responsable de la “home page” en la raiz del sitio web y de la “about page”

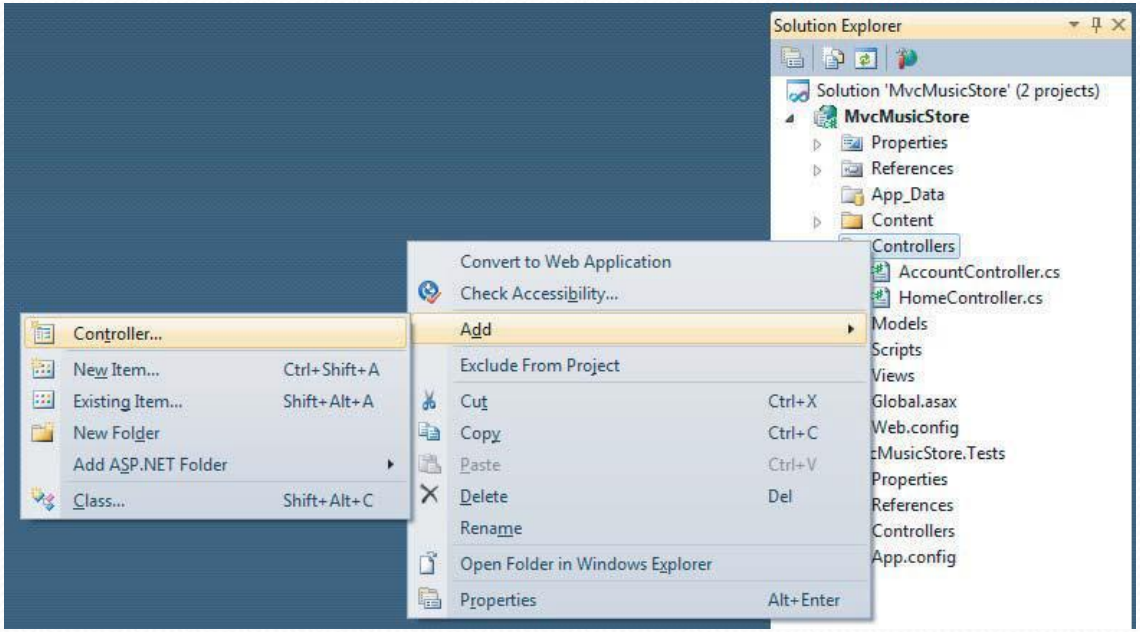
  - **AccountController**: Responsable de los pedidos relacionados con la cuenta del usuario, como el login y la registracion de usuario

  - **AlumnoController**: Responsable de Grabar datos relacionados a las inasistencias y llegadas tarde de un alumno.

Creando un Controller

- Comenzamos creando un controlador que maneje los pedidos relacionados con un catalogo de productos en una tienda. Este controlador va a tener 3 escenarios:

  - **Index page** listará todas las categorías de productos de una tienda.
  - Presionando sobre una categoría, nos dirigiremos a una página que liste los productos de esa categoría en particular.
  - Presionando sobre un producto iremos a una página de detalle que mostrará la información particular del producto seleccionado.



Creando un Controller

- El nuevo **TiendaController** contendrá luego de crearlo un método **Index**.
- Vamos a agregar 2 métodos adicionales para implementar los otros 2 escenarios que queremos que nuestro TiendaController administre: **Buscar** y **Detalles**.
- Estos métodos (Index, Buscar y Detalles) en el controller se llaman: **controller actions methods**.
- El trabajo de estos actions, es responder los pedidos URL, realizar operaciones correspondientes y devolver al navegador, o a quien lo invocó, la respuesta.

Escribiendo Actions Methods de un Controller

- Para tener una idea de como funcionan, seguir los siguientes pasos:
  - Cambiar el valor de retorno del método **Index()** para que devuelva un **string** en vez de un **ActionResult** y cambiar el valor del return a **"Hola desde Tienda.Index()"** como se muestra abajo:

```
//
// GET: /Tienda/
//
public string Index()
{
        return "Hello from Tienda.Index()";
}
```

- Podemos crear los dos métodos siguientes: Buscar y Detalles.
- Para acceder a estos controllers desde un navegador, se invocan los actions methods de los controllers y los returns de strings ([url]/tienda/index, [url]/tienda/buscar y [url]/tienda/detalles)

Escribiendo Actions Methods de un Controller

- Navegar **/Tienda/Detalles** provocó que se ejecute el método **Detalles** dentro de la clase **TiendaController**, sin ninguna configuración adicional.
  El que se encarga de esto es el sistema Routing (lo vamos a ver mas adelante).

- Solo pusimos texto en un navegador con un **Controller**, no utilizamos ni un **Model** ni una **view**.

- Aunque los models y las views son muy utilizadas y útiles en ASP.NET MVC, los controllers son realmente el corazón de una aplicación.
  Cada pedido va a un controller, con o sin necesidad de utilizar un model o una view.

Parámetros en Actions Methods de un Controller

- La clase, el método y los parámetros de un controller son especificados en la URL que se invoca , y el resultado es un string que se le devuelve al navegador.
- Ejemplo:

```
//
// GET: /Tienda/Detalles?texto=Leo
//
public string Detalles(string texto)
{
    string message = “Estas dentro del método detalles de la Tienda e ingresaste: “ + texto;
    return message;
}
```

Parámetros en Actions Methods de un Controller

- Vamos a utilizar **HttpUtility.HtmlEncode** para validar el input del usuario.
- Esto evita que los usuarios inyecten código JavaScript o HTML malicioso desde un parametro (Ej. /Tienda/Buscar?producto=<script>window.location='http://hacker.example.com'</script>)
- //
- // GET: /Tienda/Buscar?producto=x&gondola=1
- //
- public string Buscar(string producto, int gondola)
- {

  string message = HttpUtility.HtmlEncode("Estás buscando el Producto = " + producto);

  message .= " Que está en la góndola: " + gondola.ToString()

  return message;
- }

Resumen

- Los controladores son los conductores de una aplicación MVC, orquestando las interacciones del usuario, los objetos del modelo y de las vistas.
- Ellos son responsables de responder a los input del usuario, la manipulación de los objetos del modelo apropiados, y luego seleccionando la vista apropiada para mostrar de nuevo al usuario en respuesta al input inicial.

Crear el siguiente Trabajo Práctico

1. Crear un nuevo proyecto ASP.NET MVC llamado “TiendaMVC”
2. Crear un nuevo Controller llamado **TiendaController**
3. Agregar los siguientes **Actions** a TiendaController: **Index, Buscar y Detalles**
4. Cambiar el tipo de dato de retorno de métodos reemplazando “ActionResult” por “string” y que diga el string: “Estoy en el metodo X” (X es el nombre del metodo actual, ejemplo Index, “Estoy en el metodo Index”)
5. Cambiar el método **Detalles** para que reciba un parámetro llamado ID de tipo “int” y que lo muestre por pantalla con el mensaje “Este es el detalle del producto con id=[ID]”, donde [ID] es el parámetro que se envió en la URL.
6. Cambiar el método **Buscar** para que reciba los parámetros “producto” de tipo “string” y “tipo” de tipo int y muestre por pantalla “El producto [producto] se encuentra en la góndola [xxx]”, donde [producto] sea el parámetro enviado y [xxx] será la góndola que dependa del “tipo” ingresado (Utilizar HttpUtility.HtmlEncode)
   Ej. De return “El producto Alfajores está en la góndola 4”

Vistas - Introducción

- El siguiente paso después de crear el controlador es personalizarlo para visualizarlo. Esta fase se desarrolla con las vistas.
  Hoy veremos la parte más básica y en días posteriores veremos opciones más complejas.
- Si seguimos con el controlador demo que hemos usado anteriormente, lo primero que debemos hacer es volver a cambiar el tipo de devolución de datos de nuestros métodos de string a ActionResult.
  Luego cambiar el return “cadena de texto” por return View().

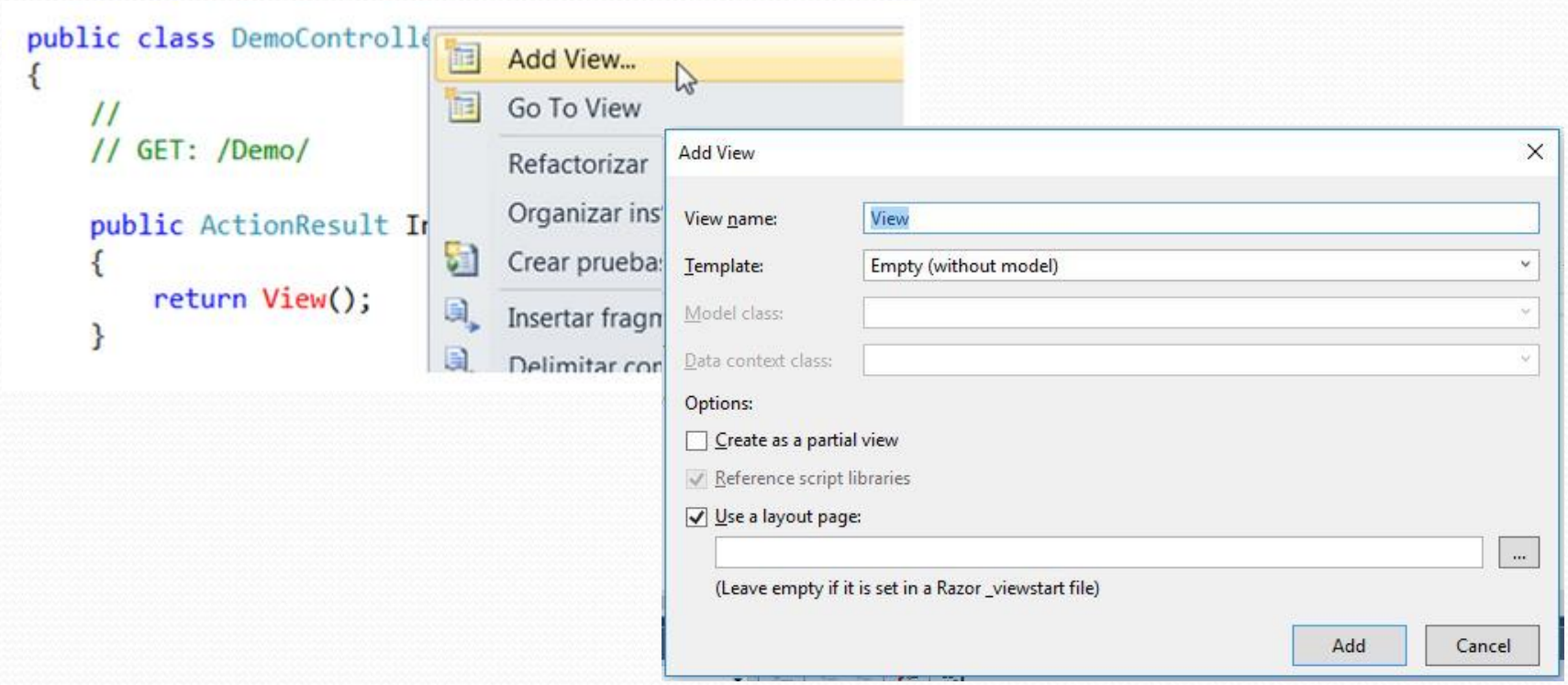
```
public string Index()
{
    return "Acceso al método por defecto";
}
```

Pasa a

```
public ActionResult Index()
{
    return View();
}
```

Vistas - Introducción

El que la palabra View aparezca en rojo nos indica que no hemos declarado la vista para ese método. Para hacerlo nos ponemos dentro del código del método (lo más cómodo es hacerlo sobre la declaración de éste) y pulsamos botón derecho y seleccionar Add View…



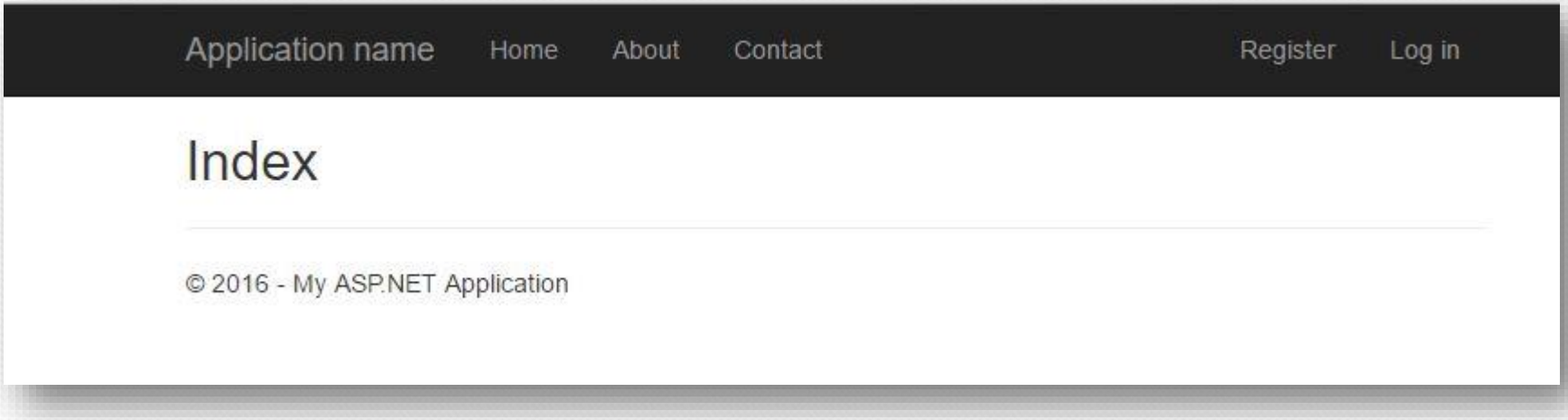
Vistas - Introducción

Nos genera un código muy básico

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

Que si lo visualizamos en el navegador obtendremos



Vistas

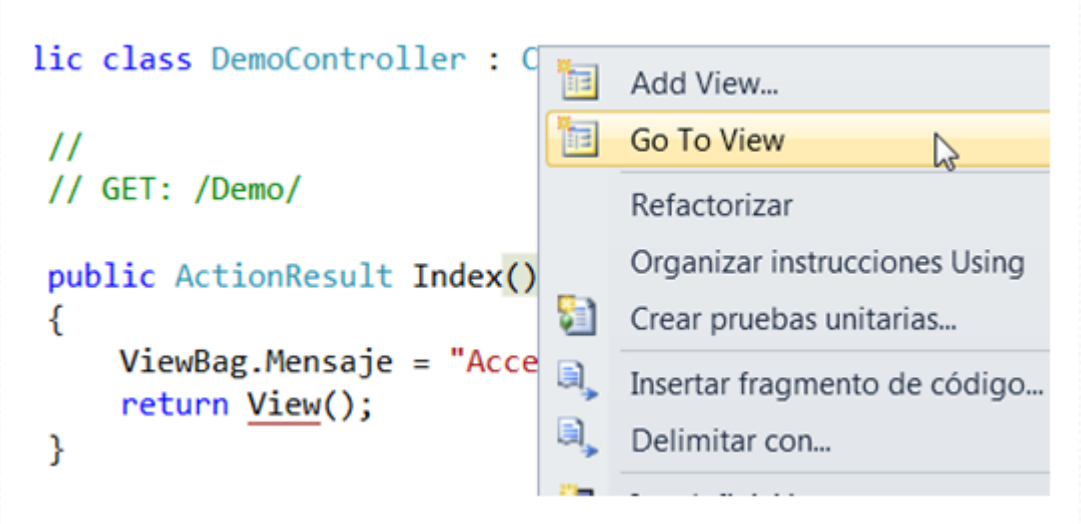
Del código generado destacar tres cosas:

1. La sintaxis @{ .. } es de Razor y nos permite ejecutar código C# dentro de la vista que contiene código HTML.
2. Aparece un objeto ViewBag que no hemos declarado, pero que tiene propiedades, en este caso Title.
   Este objeto se usa para pasar datos entre nuestro controlador y la vista.
3. Por default usa como master page el que esté definido por defecto
   (Views > Shared > _Layout.cshtml)
4. Por ejemplo si queremos mostrar el mensaje «Estás en la página index» dentro de la **vista** y que **controlador** sea quien lo mande, deberíamos incluir una línea en el controlador, asignando una nueva propiedad a ViewBag.
   En este caso se los asigno a Mensaje, pero podría usarse cualquier otro nombre.

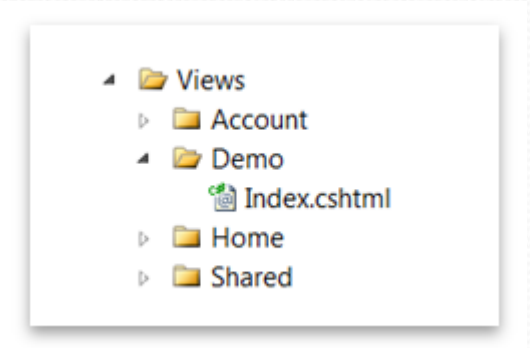
```
public ActionResult Index()
{
   ViewBag.Mensaje = «Estás en la página index";
   return View();
}
```

Vistas

Pulsando el botón derecho dentro del código del método y en esta ocasión seleccionar **Add View**



Accediendo a la capeta **Views**, desplegar Demo y visualizaremos Index.cshtml. Doble clic sobre el fichero y lo tendremos abierto para trabajar.



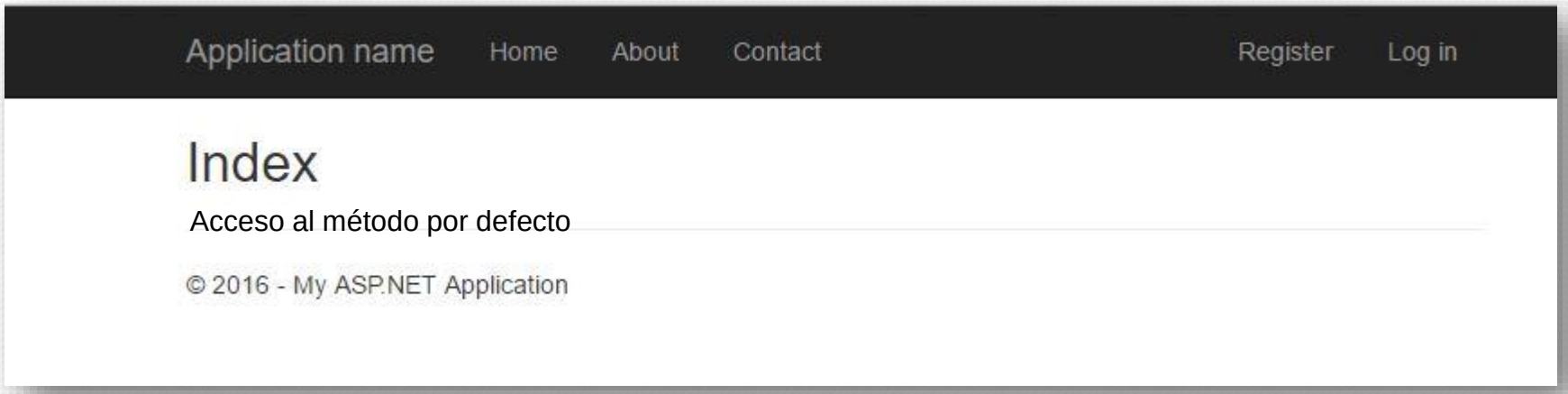
Vistas

Ahora incorporamos código para visualizar el mensaje. Con Razor se hace con @NombreVariable o @NombreObjeto.Propiedad

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
@ViewBag.Mensaje
```

El resultado sería el siguiente.



Razor

En una sintaxis basada en C# (aunque se puede programar en Visual Basic) que permite usarse como motor de programación dentro de las vistas de nuestros controladores. Es una de las novedades de ASP.NET MVC 5.
No es el único motor para trabajar con ASP.NET MVC.

También se puede destacar que dispone IntelliSense dentro de Visual Studio con lo que agiliza enormemente programar dentro de las vistas.

Razor - Características

1. **Compacto, expresivo, y fluído**: Razor reduce al mínimo el número de caracteres necesarios en un archivo, y permite un flujo de trabajo rápido y fluido. No es necesario interrumpir la codificación para indicar de forma explícita los bloques de servidor dentro de su HTML. El analizador es lo suficientemente inteligente para deducir esto de su código. Esto permite una sintaxis muy compacta y expresiva.

2. **Fácil de aprender**: Te permite ser productivo rápidamente, con pocos conceptos. Sólo es necesario usar su experiencia en tu lenguaje predilecto y sus conocimientos en HTML.

3. **No es un nuevo lenguaje**: Permite a los desarrolladores utilizar sus conocimientos en C# o VB y con Razor entregar una sintaxis de plantilla que permite construir HTML con el idioma de su elección.

4. **Funciona con cualquier editor de texto:** Razor no requiere una herramienta específica y le permite ser productivo en cualquier editor de texto.

Razor – Reglas de sintaxis para c#

1. Los bloques de código Razor son encerrados entre **@{ ... }**.
2. Las expresiones en línea (variables y funciones) comienzan con **@**.
3. Las sentencias de código terminan con punto y coma (**;**).
4. Las variables son declaradas con la palabra clave **var**.
5. Las cadenas de caracteres (strings) son encerradas entre comillas.
6. El código C# es sensitivo a mayúsculas y minúsculas.
7. Los archivos de C# tiene la extensión .cshtml..

Vistas - Forms

1. Creamos nuestra vista con Forms e invocamos a la acción del controlador

```
<h2>@ViewBag.Mensaje</h2>
<form method="post" action='@Url.Action("Detalles", "Tienda")'>
    Usuario
    <input type="text" name="Usuario"/>
    Contraseña
    <input type="text" name="Contrasena"/>
    <input type="submit" />
</form>
```

1. Se vería de la siguiente manera:

Controlador - Como recibimos lo que se ingresó

1. Creamos dos acciones llamadas de la misma manera pero una tendrá los parámetros que recibirá y además el método de envío de la información

```
public ActionResult Detalles()
{

    ViewBag.Mensaje = "Bienvenido a mi Página de Detalle";
    return View();
}

[HttpPost]
public ActionResult Detalles(string Usuario, string Contrasena)
{
    // realizar proceso de búsqueda

    ViewBag.Mensaje = "Gracias por ingresar tus datos " + Usuario + "!!!!";
    return View("Gracias");
}
```

Vistas

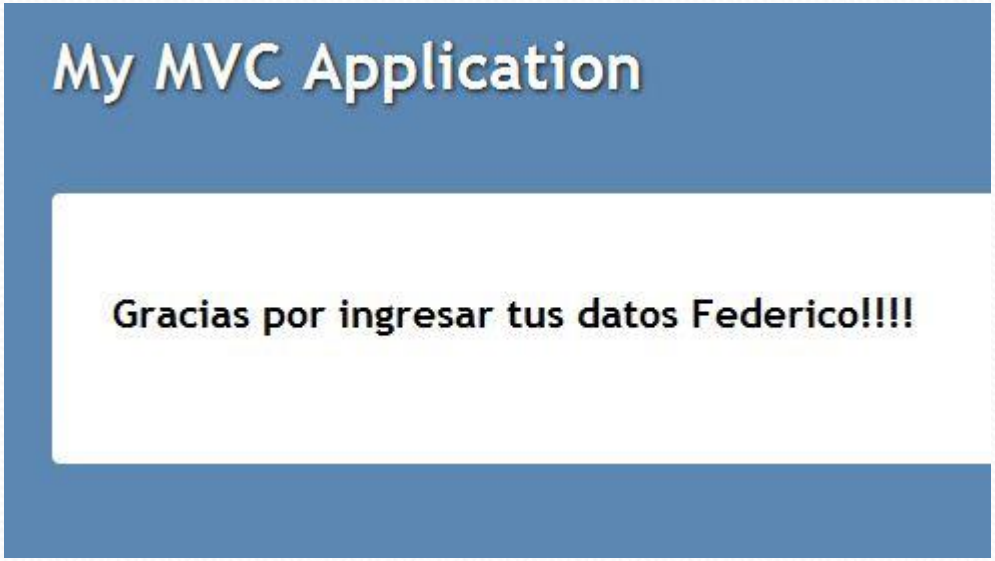
2. Noten que en el método del POST lo enviamos a una vista llamada **Gracias**

Gracias.cshtml | Detalles.cshtml | TiendaContr

```
@{
    ViewBag.Title = "Gracias";
}

<h2>@ViewBag.Mensaje</h2>
```

2. Por lo que luego de ingresar los datos mostrará la siguiente pantalla



Crear el siguiente Trabajo Práctico

1. Crear un nuevo proyecto ASP.NET MVC 3 llamado “TiendaMVC”
2. Reemplazar “Bienvenido a ASP.NET MVC!” en el método **Index** con otra frase (Alguna que invite al tipo de tienda que elijan)
3. Crear un nuevo Controller llamado **RegistraciónController**
4. Agregar los siguientes **Actions** a TiendaController: **Index, Registración, Login , OlvidoPassword y DatosIngresados.**
5. Crear las vistas para todos los Actions:
    1. **Index:** Contendrá un mensaje indicando que el usuario debe presionar el link Login.
    2. **Login**: Contendra los campos de login y los links a registrar y olvide mi contraseña
    3. **Registración**: Tendrá los campos y el botón del TP Anterior
    4. **OlvidoPassword:** Tendrá los campos y el botón del TP Anterior
6. Todos los actions llevarán a la vista **DatosIngresados** enviando, mediante el objeto ViewBag, todo lo que el usuario ingresó.