



# **CIFP César Manrique**

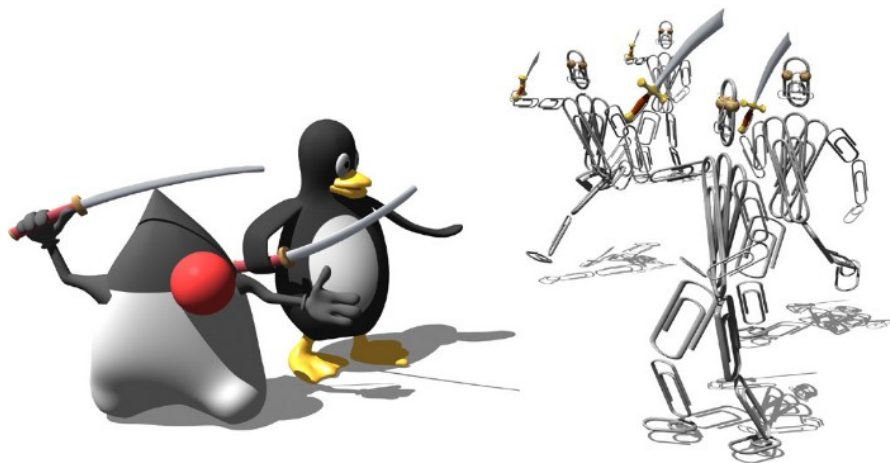
**UT 8. ASP .NET MVC**

**CFGS DAW – Desarrollo web en entorno  
servidor**



# ASP .Net MVC 5

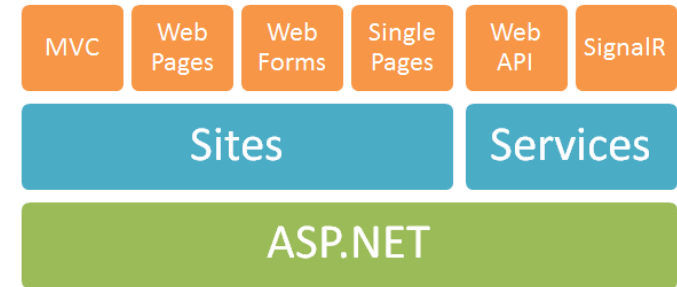
<https://www.asp.net/mvc/mvc5>





# CONTENIDOS

- Introducción
- Modelo MVC
  - Controladores
  - Routing
  - Action Method y ActionResult
  - Vistas
  - Selectores
  - Model binding



- .Net es un truño..., Java tiene sólidos principios de diseño...
- Vas a trabajar 10 años con .Net
- Java es un truño... ¡Viva .Net!

*Somos mercenarios  
(Fernando 2016)*



## Modelos de programación en .Net

- De forma similar a Java, donde podemos encontrar frameworks como JSF y Spring MVC (uno basado en un potente pero pesado modelo de componentes de servidor, el otro más ligero basado en request), en .Net encontramos **ASP .Net Web Forms** y **ASP .Net MVC**
- Estableciendo un paralelismo, muy a grosso modo:
  - ◆ ASP .Net Web Forms → Java Server Faces **JSF**
  - ◆ ASP .Net MVC → **Spring MVC**
- A grandes rasgos podríamos decir que los frameworks basados en árboles de **componentes de lado servidor**, son indicados para aplicaciones de **Intranet**, los segundos, orientados a **request**, serían más aconsejados en aplicaciones de **Extranet** o donde la integración con frameworks cliente (JavaScript, Angular) sea indispensable



# ASP .Net Web Forms vs. ASP .Net MVC

- ASP .Net Web Forms tiene dos problemas principales:
  - Es pesado y muy orientado a POST, debido a que mantiene el árbol de componentes de servidor, requiere de mucho trasiego de datos entre cliente servidor: el pesadísimo **VIEWSTATE**

```
<body>
  <form method="post" action="webForm1.aspx" id="form1">
    <div class="aspNetHidden">
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
        value="hfrayhq09cEK735FA/btisy7ds1d0NoQ8BnjmUll+1itwCvmpHE60QLVr1h/Rx10E0pzeEz6XyQchdzcj5BNF9LRF0zZd+SAH3XzF3vE Lxp9I2mDkRZATU8Na3y0WiaXgP0uBvRLu61xkKkQ6q817F71
        Gkx9L80G0vzpu35c1PD8LxgCugOra7NbdwL3HT8QSLH3ZF1EehAXet0YUAEbuRNRiKoaactcnVd8cCBUA0Yjnn8zo9ohWu11x3/63E0Wan/BTXsM4jrbzloQ0MCv7yQQssh1YD1K6pG/Q17LesvbNcPe2
        cmQ0genfXXCK7LLXSNzIzbx+huUyEkqHBCZKX0EUQ6XZAS5U1qnpOILa8BmCzE/66mL1YpGUPQCfSADCHV8U1M5C1Yw75u9ywl701H2J1Q5gE1tex004CmoD1Pvq+H41E1D20TCf3xH84zR9aX1zQumw4
        7Kq0pD1PHVb70NHN+p4k5GqCtHjDg8z1sOr3Q1d9q2U0uXpUx0NvaVcIEuSut+P104yqld0tHTRNOrbdsBbmJeo8myTnDbHC06z08/3y70kYv9z8EpVH89yZm0ldtzw1YpUlnThZvR+Earifxykm3u1xNw4
        r5Y9MnavPLD2nPi1y9bW63PjnhUVK1J0dfHj5ySR1AG0u4y7f8HcaMhMcdtrSCJ1K2GxfnfmoHE1f5z3bhRpd6yOGKJN+uQ0y3Y0trB4RRPYSash80vpgL4Axf7cKSymLNIzqnrSx04vH2pZUT11yF
        Q0uuH7t1cxhd0466fv3QL5a1KKc0F3H21Z1yuaSVK0nssQ/p6gt,yaeQf74u9jrcFNXP6YX0tucP8RhdIbngbhpLnRudWfn7+/6jse8XU50RrevH435dmy2ofYUkXF/s6vXtHChC27Q87Z1/0v
        P2h3o990Ra3LURdrsgk579mCYkN0+svQ00EKA/EmdTa1lAc+huuXurty3HT7z5QFb1a8efq4K7X27D5z5XK5ZClnRudWfn7+/6jse8XU50RrevH435dmy2ofYUkXF/s6vXtHChC27Q87Z1/0v
        hA0yG0vzrH90bDen8Xgq9wH5777TgBuig7QqPbb8TVatwH4HY/7aDKrvqGRK90gbP2vc4IaaC752NxfL8w2b1eaZJCA3C1u+3xpFers1zC6GRY1H6SdFanX0eoTC1g18fUatazR5bGK79Kk8ti
        lKdMvY0A1hua69G6fk04VhMQV2tq/UmeCm8pcDeec4qH56t8nG05UwPshTRP1N1rtmVnc0YehIn18ov13gdI00oPn/wb9t305FuiF51AXMg9ESuJa/1odaJf+/nD1guLV46pUCR9d0vH268X1b0ri
        muRkUk1mur19ZaGhd8m3sojH+sE1csjFucxQU8806vY5hC48KeIdjUscDnhS7CPc9t14b07U0ecy81FAX1YfK10vyyxo9T35Fj0frjstiafz2xfGZqsdt11QmbsnCqUuytDWGG2TP9Qcs/8s77Xj0yJc08si
        yHq8ArlPhf9o6xXJ0uUkUakGiv8Ks9vH0Q0azQ8B0s28H6X+2VBNZ2PjESV95ChkQ0hI2KK3+YHhXPF5z90SIRn1Lex8D5/B/zbfw82TwAkRyRmu3uHv67n2NuHdf8/VLSbe+Z9oq4z821V30jG1F4ci
        4g4Xa/01VFDkxk1d8vcB1lp8kdxCSHFjYX35Y0Lrfs0o2LauyU0jy+8+kmv1hubs1gHXckXJaaBYTdr1p+63A/9etzhxmE5J8/dh5kUeg4ta7X+q6KDuQ8SgpdW1W6Y9A2V25FP9/2JQ0d5gAsQd5mJ
        3611c4QpGT1ztya+Q2mC6HMDhnlEMr755z2m3sLLV0Z0S+rjef/aAYCr6Gc5UPRF93zoara3PQ200c4NM/zfucjTz03NaD7Hr5vLc0nc4m1CnByanR7Pg8Xtq+4NakX17VRVw154eZn941j7L37A140rCl
        jKZnLF1d3QJd80KAp681qt29IIasu20qKd0e6xtFctq5E5FI1EHRUXVsp1jKfotof70/Knjl0iNU5S20GZdTckK8Pve2H1e1gR4cxk7AF1yyH2fSsrFSU7I10VgqOPE1c1w681TQ1F2TK5Y940q0M
        Ek0Nvhd02J08tmgdUSREthmlvDV+Ytb3Z7s8Xry00Q+ASjP66/PhqXaunrveA13uPk1oX1IdQl/+2y51281k+0uo+j+X2sAH50Y7x0k8sP08uamuSEHf1RzbnHf0Jcatu3140018e0e+/L/t9/zg001f
        tLx0+dtbu71k7j0oTKF429AX/GHDTK6X09sXsTwuXsSroqD7P8TnIVBj69xydx+e0KLUkTtY0Zg56V550F8H23Q1qoHqTKLH6gRz+4anuJjshGZ6CyhQ2913LYB157Xf091bH7TgdR5wqeP9Ny9CqUu
        I6MaN4ZoMhN54xun3FstX/pj4e561pHYk07PqTstltu0uhs57/qMAU88Xqu2Fzr+RLN59E2GP10DC4dNldhtYc+q0koNP4Zbkzn8X5Zo6FckR1R1sRsk90wIDg0gMebrTtRaL126bu7jGnKF1g7qg8I
        CPhp1g1s1e85o/c2N6Vcfc7X4RQ7P1Pks0ty+Om/q4CbLHdGLXtInkTISHC0THU4ovvHlJmnsE3M/v0URD8zPglR+6b2VIQ/1Czsp+BP0dg257j0rH1Hbsssc2RL0hE157P9uY4e19Xr0I2hmjha0j2Poi
        KIS/PH0Coby8f3dxsIH7XnqD09v0jcs3EE09NgkU081JChIP7a0L7cb/sSn/7T++2vha1pdaM1E0pp/bf4P/PP65+mcZbdfCw6fvQ2cH1FkTrRenOHZRoQ511Intk0GhF5TNT7eH0XndT1t8rsk/bjJrsILI
        VhndIey1pt2opemfHrJ5R1tHKECgk16b60L3u7JQJ0WQ1c4jEY26m12DyVCUCh1BAYa6gH8Q8c28G/R4Eh+8CwcQmc5B6dq2qnd05d0+23kURncISu0R2Vg7U1ES/WUPzhifK08UvU/+3gvCXB8Pwz8
        tVhndIDoyyuvva8IUrF8t/VzRw0hyq78hAK5638Aeq3Au7hydyRcUuM2gc+qPwa6yL0ArVuFz/Qr2V7zhe0klb3GomDsuWu4gH1aousFFD4Xs2F4yovtdfCBIVxcUUDG5qumUJ2z5K/z0nQKCI8+00q
        vG8bns85Ms1L0w78HyVHTYfy1sKXJd3uRzISGQKottVMDkV7ID6FQ7nbZnJaH7VpNwHq2VPA7Nab9zke1NHRVR55/zHEIdeBmmZErRoVfGt291bd1wzfiA8FXpaxjyJNuecsdq15kdZaa
        /nxtNF+74CnPhIBPs+SeE1yCh3XnLBur1UKxR8Pnd19GdeHuaqXexF0Q2Kz5DOPC7Ru11huos7F0n0pd46/Hd/n1eEF+u1PQLA7e1gY8JsuNkgf8Cnu7YYPQot7ztH30MsEx24H001+u4Nab85RYP+Cd4gd
        xmZF6s0C440qj6dU6jZ5s0p6Wt1LKNRKM2n9v3y5eFURcpVU08rFvXc+q2vQdht6g+0z+5pYRyKADu19K0e4xTLOELUkG0BQBUm92NmeP9TupicvXs5tdvPv9C6g7Tc14pdadu0UEth/df19ag
        xp1FIANk11vg7ia77YHb8nTXUzeZTh0b/7h52zK3f7D6x97T/mY38fCnKCKGuaC7CZ2hsQ5aVckA3J+Rab8ThkU49SU9R6tnxpyW4jRxtgdvDmdhP8otX+HkR9qXhVklal0vneZjzn7HY9FRP09pK1Zl
        X68N8e82LmofTxCec75ovvDed0KLL08dH2+cf+0R5sL1YV1y7J3T0FvF8eG5VvG30eFls2X9TE11R1w13a0eLs2ErX1vX305Zcd0bVFA3Kd+hFR0T+7anv9NCE3HTK8fduSc3Hf0d0f/nABi
```

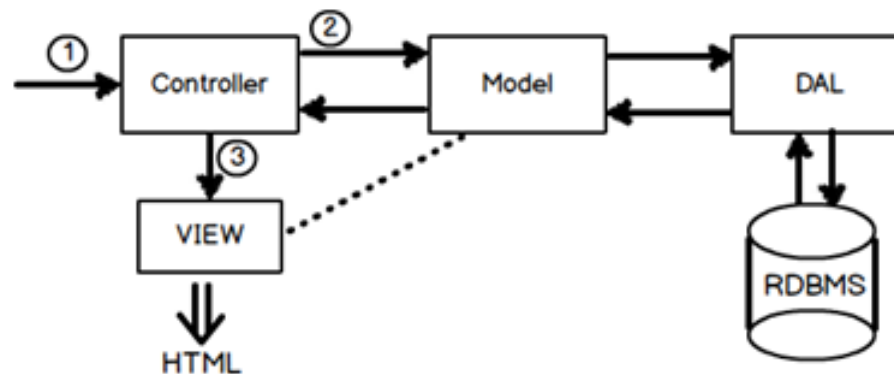
- Esto implica mayor necesidad de ancho de banda y menor velocidad
- Otro problema es que el código HTML generado es muy difícil de tratar e integrar con frameworks de lado cliente, tipo JavaScript

## ASP .Net MVC

- Web Forms es lento, pesado y la lógica está muy acoplada a la vista, por culpa de los ficheros Code Behind

**NOTA:** aunque JSF sigue una filosofía similar, basada en árbol de componentes de servidor, pero su diseño evita muchos de los problemas de Web Forms.

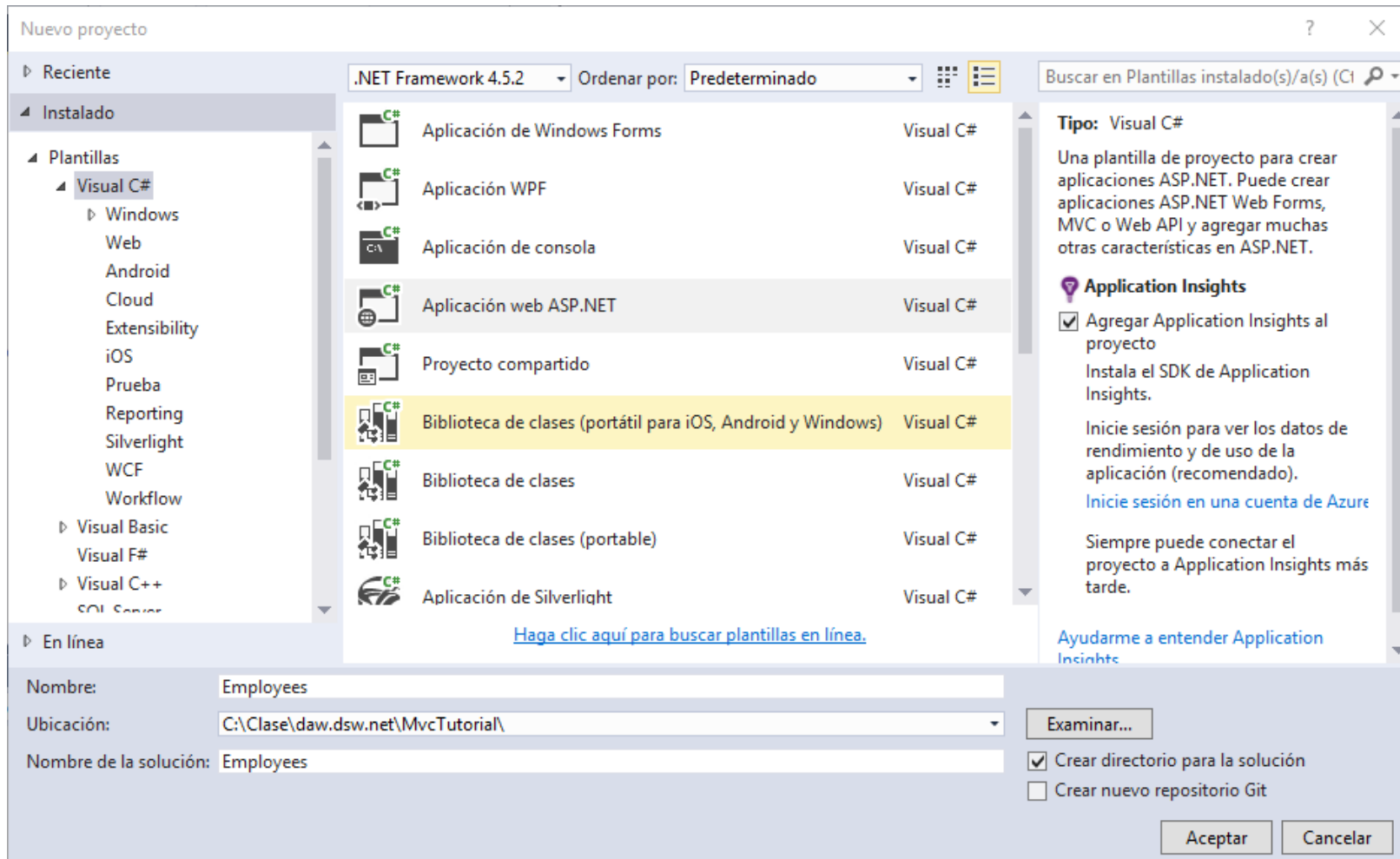
- ASP .Net MVC se inspira en el patrón Model View Controller y busca separar mejor las responsabilidades entre las capas de presentación, lógica de negocio y acceso a datos. Es una filosofía similar a la de Servlets + JSP, pero incorporando mejoras en el enrutamiento de las peticiones (request)





# Crear proyecto y solución Employees

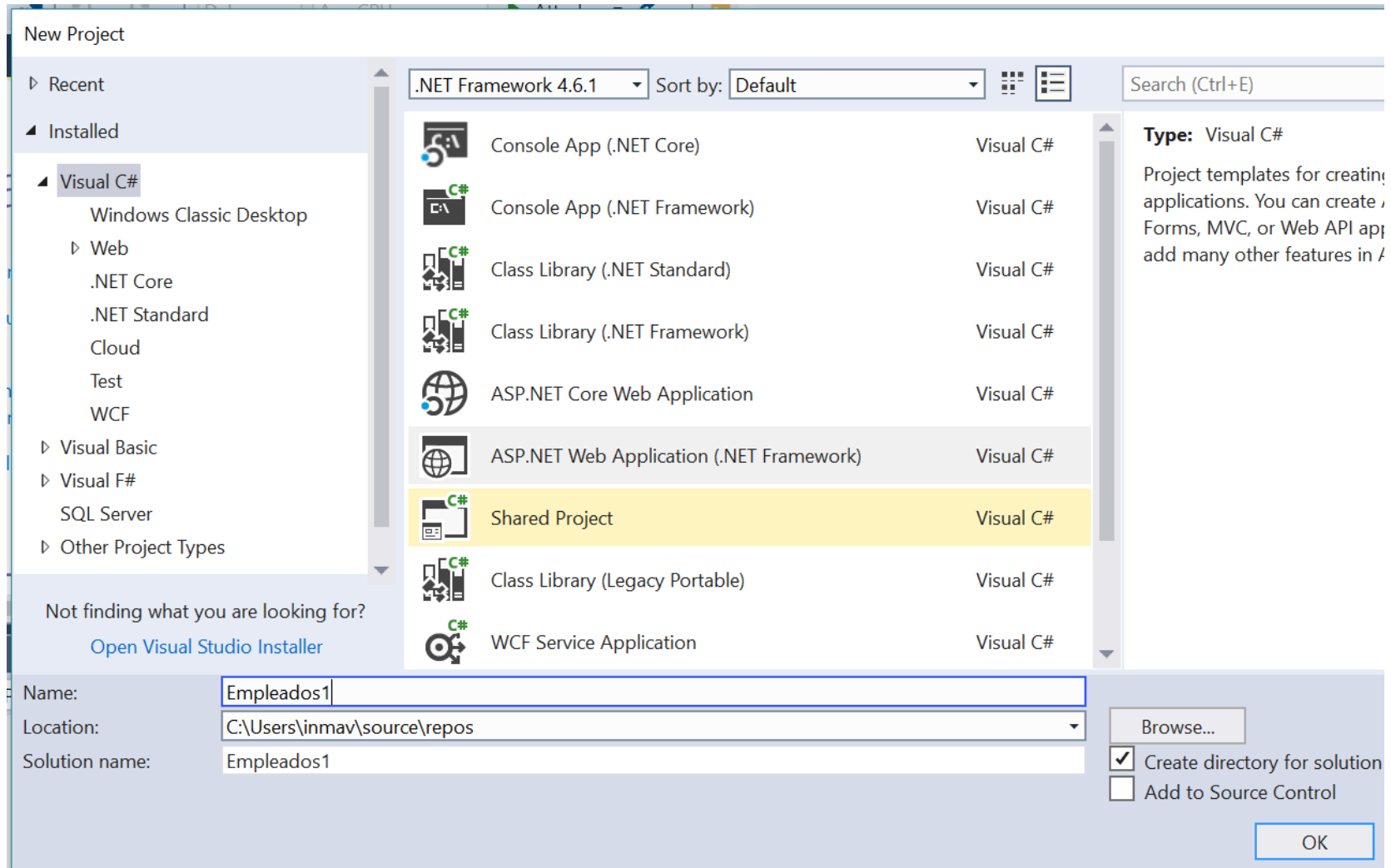
- Creamos un proyecto C# de tipo “Aplicación web ASP.Net”





# Crear proyecto y solución Employees

- Creamos un proyecto C# de tipo “Aplicación web ASP.Net”

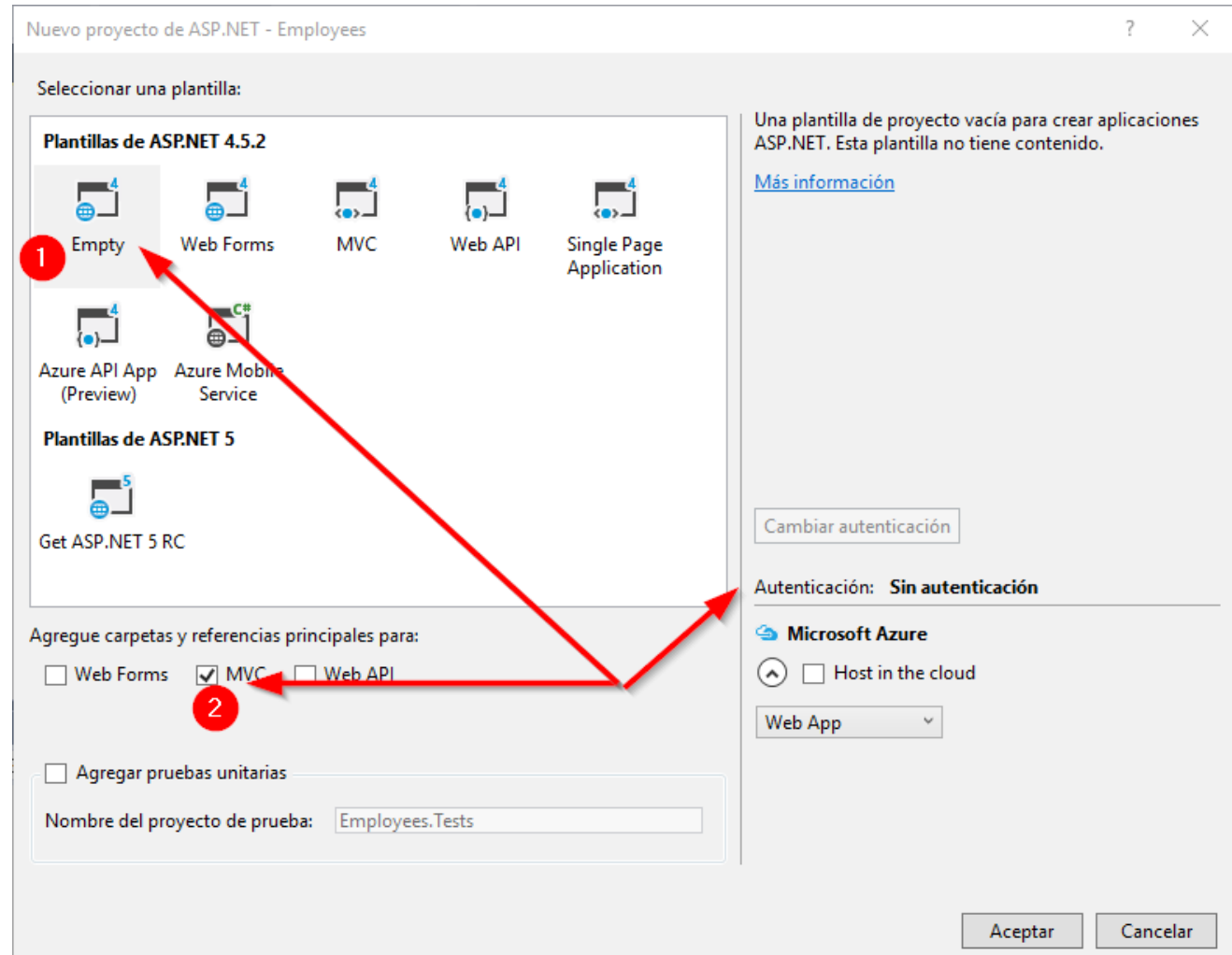






## Plantilla de proyecto

- Plantilla vacía (1)
- Pero añadimos referencias a MVC (2)
- No agregamos proyecto de pruebas











# Plantilla de proyecto

- Plantilla vacía (1)
- Pero añadimos referencias a MVC (2)
- No agregamos proyecto de pruebas

New ASP.NET Web Application - Empleados1

 Empty  Web Forms  MVC  Web API  Single Page Application

 Azure API App

An empty project template for creating ASP.NET applications. This template does not have authentication. [Learn more](#)

[Change Authentication](#)

Authentication: **No Authentication**

Add folders and core references for:

☐ Web Forms ☒ MVC ☐ Web API

☐ Enable Docker support (Requires [Docker for Windows](#))

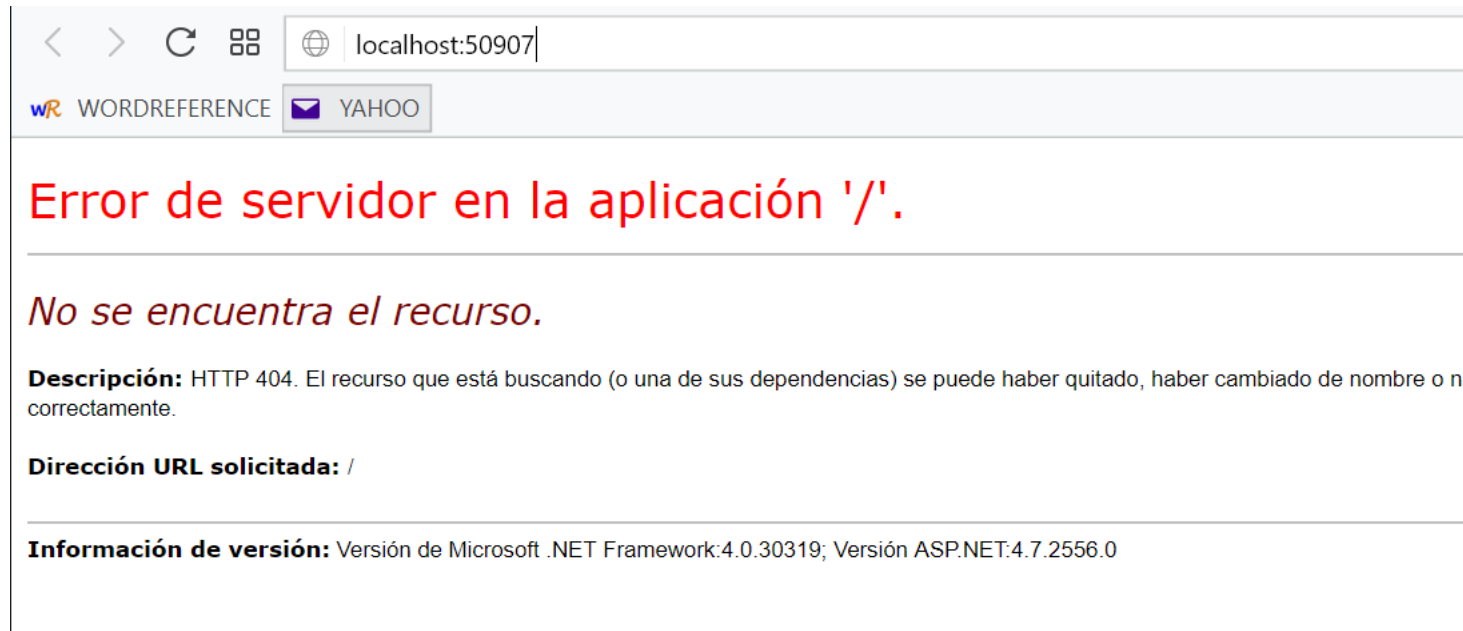
☐ Add unit tests

Test project name:

OK



# Se acabó la programación.....nuestra primera Aplicación web sin escribir ni una línea.



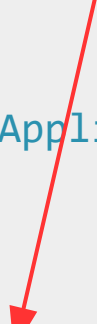
Menos mal que no ha funcionado....después de tantos años estudiando....



## GLOBAL.ASAX

- ¿Dónde están las welcome file?
- Archivo de aplicación Global.asax. Se encuentra en la raíz del proyecto. En él se declaran todos los eventos a nivel de aplicación como el inicio y fin de la misma, las peticiones ...  
<http://www.tuprogramacion.com/programacion/que-es-el-global-asax-y-como-funciona/>
- Una de las tareas es registrar las rutas de la aplicación

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





## Ruta por defecto

- En Global.asax vimos una clase RouteConfig que definía la tabla de enrutamiento de nuestra aplicación.
- Examinando esta clase vemos la ruta por defecto
- La petición GET se mapea, de forma que la 1ª componente se asocia al controller, la 2ª al método del controller y la 3ª a los parámetros.
- Por defecto se añade el HomeController, método Index

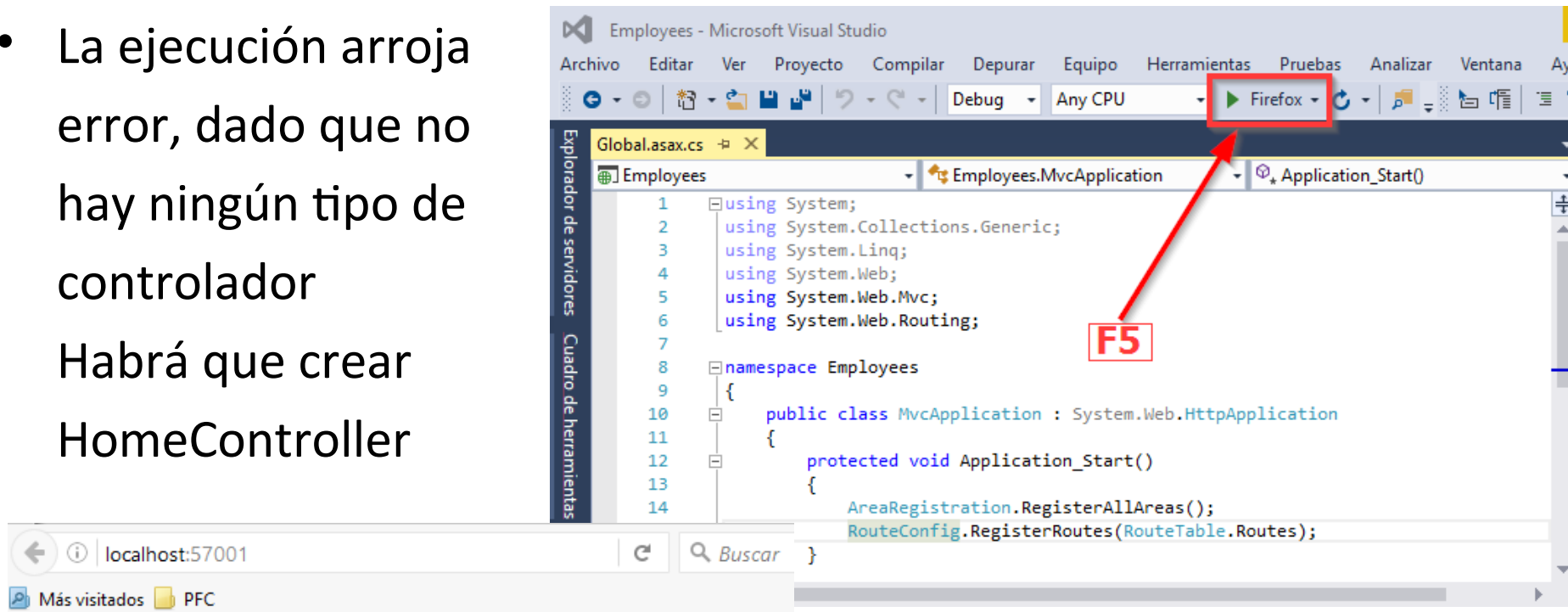
```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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



## Ejecutando el proyecto

- Sin añadir nada, vamos a ejecutar el proyecto
- La ejecución arroja error, dado que no hay ningún tipo de controlador  
Habrá que crear HomeController



Error de servidor en la aplicación '/'.

*No se encuentra el recurso.*

**Descripción:** HTTP 404. El recurso que está buscando (o una de sus dependencias) se puede haber quitado, haber cambiado asegúrese de que está escrita correctamente.

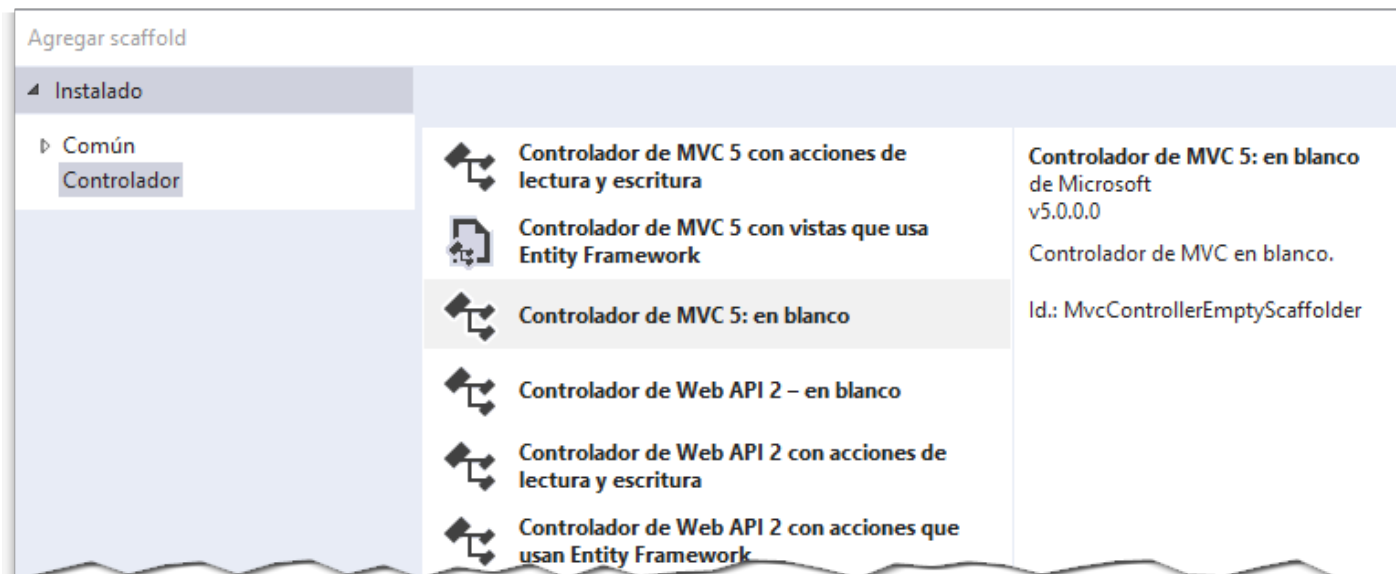
**Dirección URL solicitada:** /

**Información de versión:** Versión de Microsoft .NET Framework:4.0.30319; Versión ASP.NET:4.6.1073.0



## Controllers

- Podemos ver el controlador como un Servlet Java. Es un endpoint .Net que está mapeado contra una URL según el patrón que hemos visto en el routing.
- Para añadir el controlador nos posicionamos sobre la carpeta **Controllers** y: **Agregar > Controlador...**
- Seleccionamos Controlador de MVC 5: en blanco (**HomeController**)





## Ejecución del proyecto

- La ejecución da error, dado que no hemos creado la vista

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

Error de servidor en la aplicación '/'.

No se encuentra la vista 'Index' ni su vista maestra o no hay un motor de búsqueda que admita las ubicaciones de búsqueda. Se buscó en las siguientes ubicaciones:

~/Views/Home/Index.aspx  
~/Views/Home/Index.ascx  
~/Views/Shared/Index.aspx  
~/Views/Shared/Index.ascx  
~/Views/Home/Index.cshtml  
~/Views/Home/Index.vbhtml  
~/Views/Shared/Index.cshtml  
~/Views/Shared/Index.vbhtml

- Vamos a modificar el tipo de retorno del **Action Method** para que devuelva **ContentResult**, que es un resultado de tipo String

```
// GET: Home  
public ContentResult Index()  
{  
    ContentResult result = new ContentResult();  
    result.Content = "Hola mundo";  
    return result;  
}
```





## Ejecución del proyecto

- La ejecución da error, dado que no hemos creado la vista

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

Error de servidor en la aplicación '/'.

No se encuentra la vista 'Index' ni su vista maestra o no hay un motor de búsqueda que admita las ubicaciones de búsqueda. Se buscó en las siguientes ubicaciones:

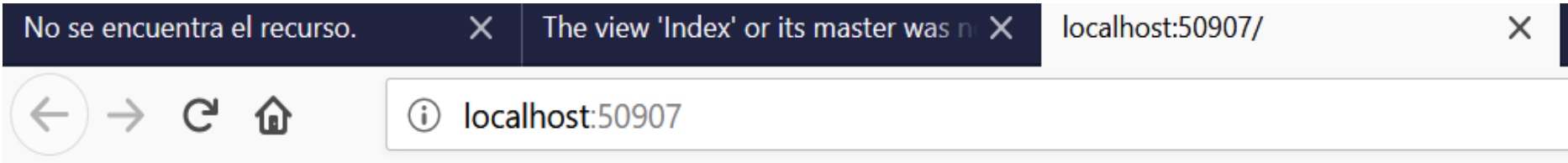
~/Views/Home/Index.aspx  
~/Views/Home/Index.ascx  
~/Views/Shared/Index.aspx  
~/Views/Shared/Index.ascx  
~/Views/Home/Index.cshtml  
~/Views/Home/Index.vbhtml  
~/Views/Shared/Index.cshtml  
~/Views/Shared/Index.vbhtml

- Vamos a modificar el tipo de retorno del **Action Method** para que devuelva **ContentResult**, que es un resultado de tipo String

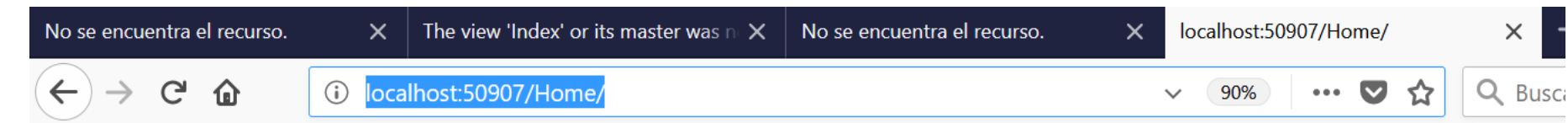
```
// GET: Home  
public ContentResult Index()  
{  
    ContentResult result = new ContentResult();  
    result.Content = "Hola mundo";  
    return result;  
}
```



# Probamos y ahora sí.



Hola mundo



Hola mundo



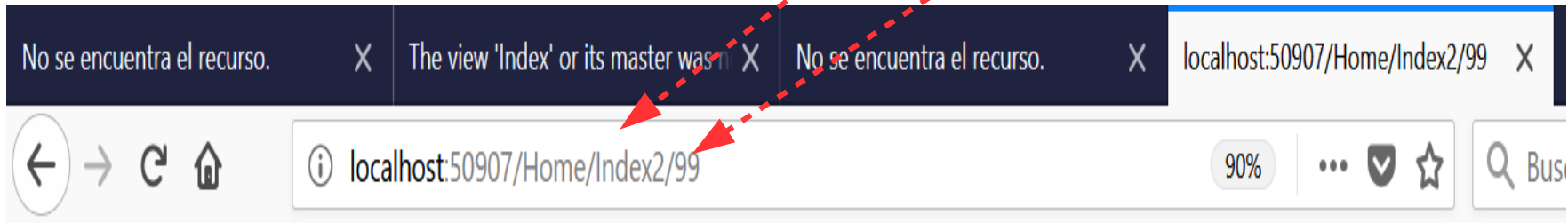
## Action Method y routing

- Vamos a crear un nuevo **Action Method**, de nombre **Index2**

```
public String Index2(Int32 id)
{
    return "Index 2, id=" + id;
}
```

url: "{controller}/{action}/{id}"

- Comprobamos como, según el enrutamiento, la URL mapea contra este método Index2



Index 2, id=99

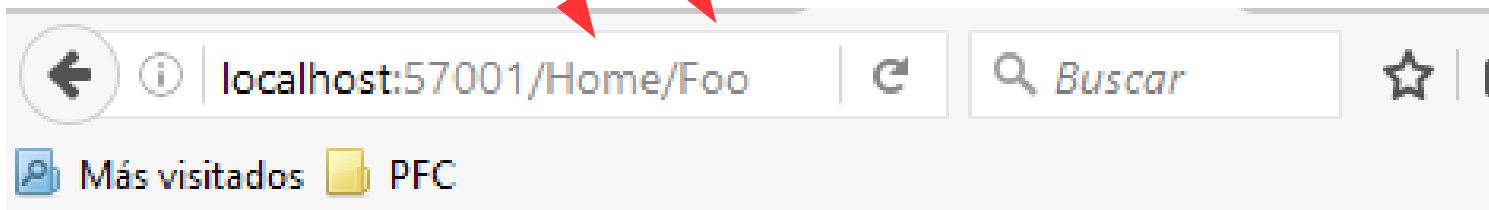


## Problemas de seguridad

- Este “maravilloso” sistema de routing tiene sus cositas

```
public class HomeController : Controller
{
    public String Foo()
    {
        return "Supersecreto";
    }
}
```

Podemos sacar cosas que no queremos  
**POR ERROR**  
(ponerlo, al menos, private)



Supersecreto



# Y nuestro pequeño controlador con sus métodos queda algo así.

// GET: Home

```
public ActionResult Index()
{
    ActionResult result = new ActionResult();
    result.Content = "Hola mundo";
    return result;
    //return View();
}
public String Index2(Int32 id)
{
    return "Index 2, id=" + id;
}
public String Foo()
{
    return "Supersecreto";
}
private String Foo2()
{
    return "Supersecreto";
}
```



## ActionResult

- Determinan el tipo de retorno de un Action Method
- Tipos de retorno de los métodos del controller

Name	Behavior
<b>ContentResult</b>	Returns a string
<b>FileContentResult</b>	Returns file content
<b>FilePathResult</b>	Returns file content
<b>FileStreamResult</b>	Returns file content
<b>EmptyResult</b>	Returns nothing
<b>JavaScriptResult</b>	Returns script for execution
<b>JsonResult</b>	Returns JSON formatted data
<b>RedirectToResult</b>	Redirects to the specified URL
<b>HttpUnauthorizedResult</b>	Returns 403 HTTP Status code
<b>RedirectToRouteResult</b>	Redirects to different action/different controller action
<b>ViewResult</b>	Received as a response for view engine
<b>PartialViewResult</b>	Received as a response for view engine



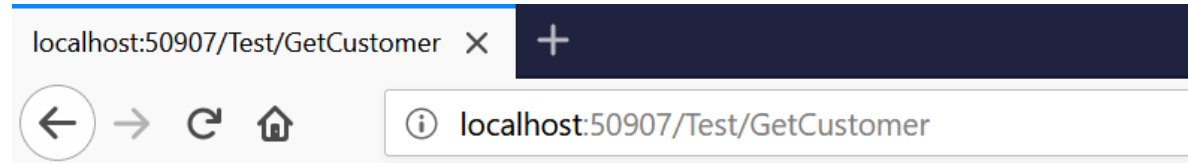
**Un poco más.** Cuando queremos devolver un objeto de tipo por ejemplo “customer”, lo devuelve como implementación del objeto ‘ToString()’, por defecto este método devuelve nombres totalmente cualificados de la clase, que es `NameSpace.ClassName`;

```
namespace Empleados1.Controllers
```

```
{
    public class Customer
    {
        public string CustomerName { get; set; }
        public string Address { get; set; }
    }
}
```

```
public class TestController : Controller
{
    public Customer GetCustomer()
    {
        Customer c = new Customer();
        c.CustomerName = "Customer 1";
        c.Address = "Address1";
        return c;
    }
}
```

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

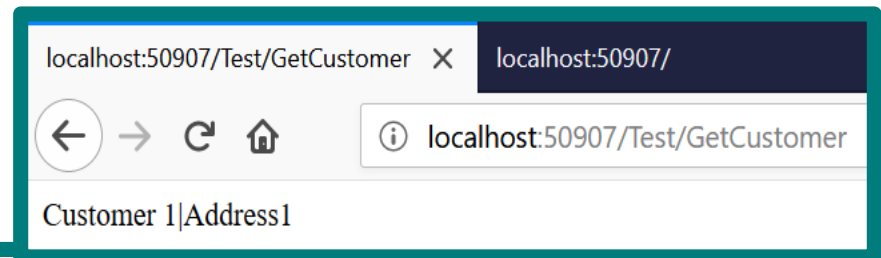


Empleados1.Controllers.Customer

**Si queremos obtener lo valores:**

```
namespace Empleados1.Controllers
```

```
{
    public class Customer
    {
        public string CustomerName { get; set; }
        public string Address { get; set; }
        public override string ToString()
        {
            return this.CustomerName + "|" + this.Address;
        }
    }
}
```



**Vistas** <http://techfunda.com/howto/19/naming-conventions-in-asp-net-mvc>

- En ASP MVC no existe el concepto de página como tal
- Todas las peticiones son gestionadas por un controlador y “mapeadas” a métodos del controlador. Estos métodos pueden redirigir a otro método o, por ejemplo, devolver un tipo de ActionResult
- Uno de los tipos ActionResult más usados es **ViewResult**, que devuelve una vista
- Una vista podemos verla como una plantilla HTML con código incrustado
- El motor de vistas que vamos a usar es **Razor**, que Microsoft dice “es lo más”  
(parece un paso atrás, respecto a JSTL)







## Añadiendo una vista

- Dentro de “Views” tenemos una carpeta para cada controlador
- Agregamos una vista con **Agregar > Vista ...**
- Creamos la vista **Index** (nombre del ActionMethod)
- Dentro de esta carpeta tenemos vistas, cuyo nombre equivale al nombre del Action Method del controller

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



+[@] Index.cshtml



## Código de la vista

```
@{  
    Layout = null;  
}  
  
<!DOCTYPE html>  
  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>View</title>  
</head>  
<body>  
    <h1>Hola Mundo</h1>  
</body>  
</html>
```



## Selectors

- Se aplican a Action Method e influncian qué métodos son llamados en base a las peticiones, ayudando al mecanismo de routing para ver qué método se tiene que ejecutar según qué URL de petición
- Hay tres tipos:
  - **ActionName**. Permite mapear una componente de URL a un nombre de método, cuando deseamos que el método tenga un nombre distinto al de la URL
  - **ActionVerbs**. Permite mapear un método a un tipo de petición HTTP específico. Por ejemplo podemos tener dos métodos con un mismo nombre y con [HttpGet] especificar cuál mapea
  - **NoAction**. Permite indicar que NO mapee un método según la componente de URL



## Convention over Configuration

- Concepto popularizado por Ruby On Rails
- Significa que si seguimos unas reglas preestablecidas, nos vamos a ahorrar mucho trabajo de configuración, dado que si nos atenemos a las reglas muchas cosas funcionarán (se enlazarán) de forma automática
- En MVC tenemos tres directorios base
  - Controllers
  - Models
  - Views
- No los definimos en ningún fichero de configuración, vienen por defecto y siguiendo las convenciones de .Net MVC podemos hacer muchas cosas sin necesidad de configuración



## Convenios aplicables a controladores y vistas

- Convenios de ASP.Net MVC:
  - ➔ Cada nombre de clase de controlador lleva el sufijo Controller: Product**Controller**, Home**Controller**... y se ubican en el directorio Controllers
  - ➔ Existe un directorio Views para todas las vistas de la aplicación
  - ➔ Las vistas de un controlador se ubican en un subdirectorio de Views, cuyo nombre es el del controlador (quitando el sufijo -Controller)
  - ➔ Por ejemplo las vistas de **Product**Controller se encuentran en **/Views/Product**



## Data Model

- Vamos a añadir unos link al HomeController, en su vista de Index (si no la hemos creado nos ponemos en el método Index y agregar vista)

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

Iremos a ProductController

```
<h2>Menú</h2>
```

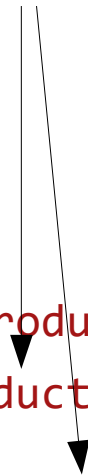
```
<ul>
```

```
<li>@Html.ActionLink("Home", "Index", "Home")</li>
```

```
<li>@Html.ActionLink("Create product", "Create", "Product")</li>
```

```
<li>@Html.ActionLink("List products", "Index", "Product")</li>
```

```
</ul>
```



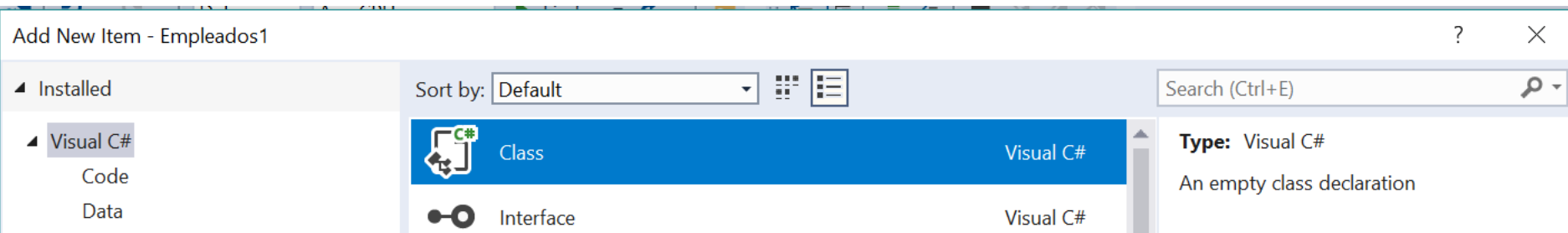
Y si queremos lidiar con productos...llega el momento de **MODEL**



## Modelo: Product

- Creamos un POCO para almacenar productos (POCO: Plain Old CLR Object) qué **poco estilo** tiene .Net ;-)

```
namespace Employees.Models
{
    public class Product
    {
        public int Id { get; set; }
        public String Name { get; set; }
        public String Category { get; set; }
    }
}
```





## Inicializar en el constructor del controlador lista de products

- Ale, ale a crear una lista de Product; variable products
- Definir el método index, que devolverá la lista de productos
- Añadir la vista y decir que es de tipo list y que utiliza el modelo Product
- Agregar luego un Create
- El Create ¿qué hace? Muestra el formulario de creación o inserta un objeto
- Hombre nos interesan dos Create, uno para la primera vez y otro para el PostBack...
- Solución





## Añadimos el producto

- Si has llegado aquí, el fin del mundo ha llegado: .Net ha triunfado y no quedan programadores en la tierra, sólo nanobots que repiten continuamente: *¿Qué deseas hacer hoy?* (esto es por si viene el inspector, para que le digáis que necesito una baja)
- Estooooo, si hemos llegado hasta aquí y hemos insertado el producto. Una vez insertado le decimos un RedirectAction a la acción index, para que nos muestre el nuevo producto insertado
- Oye, pero ¿veis el nuevo producto insertado en el listado?



## A partir de aquí

- Vamos a ponernos con un tutorial que está bien
  - <https://www.codeproject.com/Articles/866143/Learn-MVC-Project-in-days-Day>



# PREGUNTAS

