# ASP.NET MVC5

Javier Martín



## Conocimientos

- Mínimos:
  - HTML, CSS, DOM, XML
  - JavaScript
  - C# o VB.NET
  - ASP.NET
  - ADO.NET
- Deseables
  - Entity Framework
  - LINQ
  - JSON, AJAX
  - JQuery,
  - Patrones:
    - MVC, MVP, MVVM
    - DI, IoC
  - Nuget

# INTRODUCCIÓN



## Definición

- ASP.NET MVC es el marco de trabajo que da soporte al modelo arquitectónico Modelo-Vista-Controlador (MVC) separa una aplicación en tres componentes principales: el modelo, la vista y el controlador.
- Proporciona una **alternativa** al modelo de formularios Web Forms de ASP.NET para crear aplicaciones web.
- Es un marco de presentación de poca complejidad y fácil de comprobar.
- Se integra con las características de ASP.NET existentes, tales como páginas maestras y la autenticación basada en pertenencia.
- El marco de MVC se define en el ensamblado System.Web.Mvc.

# Que NO es ASP.NET MVC

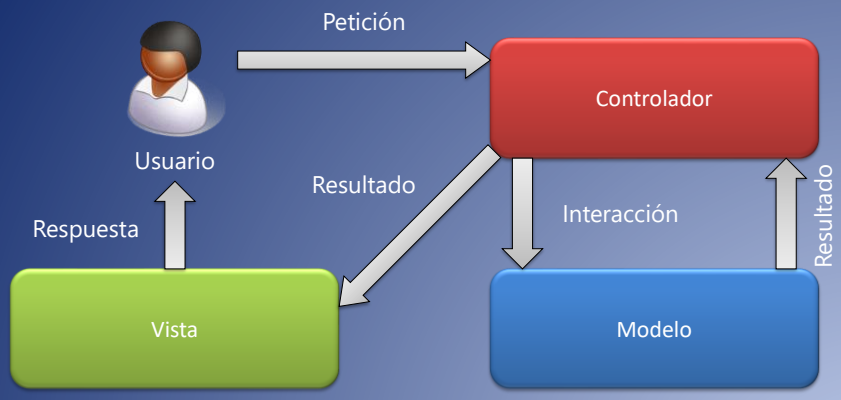
- NO es «una nueva versión de ASP.NET»
- NO es «el sustituto de webforms»
- NO es una «vuelta a los 90»
- Es un framework moderno, novedoso y ágil aunque esta basado en ideas muy anteriores a los 90 (Smalltalk -76)



# El patrón MVC

- El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software (presentación) que separa los datos y la lógica de negocio de una aplicación del interfaz de usuario y del módulo encargado de gestionar los eventos y las comunicaciones.
- Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones, prueba y su posterior mantenimiento.
- Para todo tipo de sistemas (Escritorio, Web, Movil, …) y de tecnologías (Java, Ruby, Python, Perl, Flex, SmallTalk, .Net …)

# El patrón MVC en ASP.NET





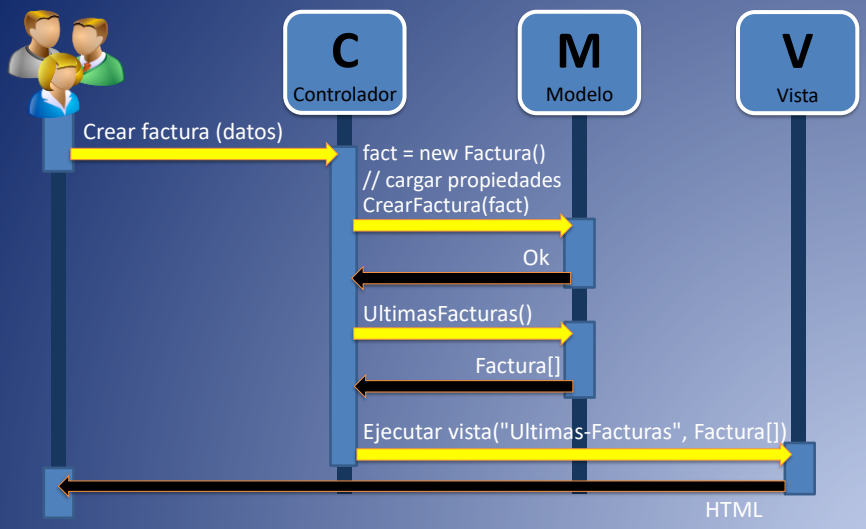

# El patrón MVC

**M** Modelo
- Representación de los **datos del dominio**
- Lógica de **negocio**
- Mecanismos de **persistencia**

**V** Vista
- **Interfaz** de usuario
- Incluye elementos de **interacción**

**C** Controlador
- **Intermediario** entre Modelo y Vista
- **Mapea acciones** de usuario → acciones del Modelo
- **Selecciona** las vistas y les **suministra** información

# El patrón MVC

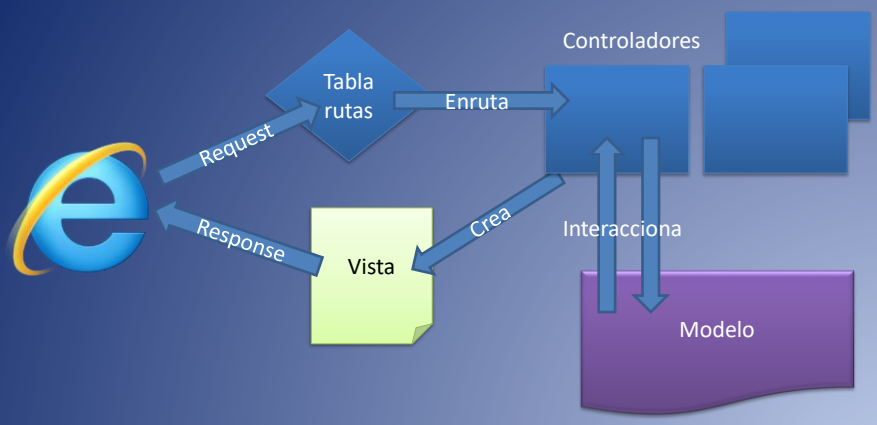





# Componentes principales en MVC

- Routing: Route tables
- Model: Bussiness Logic & Domain Entities
- Controllers: Presentation Logic
- Views: Presentation Model, Model state

# Modelo, Vista, Controlador






# Tabla de rutas

- Dada una URL decide qué acción de qué controlador procesa esta acción
- Sólo tiene en cuenta la URL (nada de parámetros POST, query string, …)
- Ruta por defecto:
  - /{controller}/{action}/{id}
- Ruta con área:
  - nombreArea/{controller}/{action}/{id}

# Controladores

- Los controladores son los componentes que controlan la interacción del usuario, trabajan con el modelo y por último seleccionan una vista para representar la interfaz de usuario.
- Exponen acciones que se encargan de procesar las peticiones
- Cada acción debe devolver un resultado, que es algo que el framework debe hacer (mandar una vista, un fichero binario, un 404, …)
- Hablan con el modelo pero son «tontos»



# Modelo

- Los objetos de modelo son las partes de la aplicación que implementan la lógica del dominio de datos de la aplicación.
- A menudo, los objetos de modelo recuperan y almacenan el estado del modelo en una base de datos.
- Encapsula toda la lógica de nuestra aplicación
- Responde a peticiones de los controladores

# Vistas

- Las vistas son los componentes que muestra la interfaz de usuario de la aplicación.
- Normalmente, esta interfaz de usuario se crea a partir de los datos de modelo.
- Se encarga únicamente de temas de presentación.
- Es «básicamente» código HTML (con un poco de server-side)
- NO acceden a BBDD, NO toman decisiones, NO hacen nada de nada salvo…
- … mostrar información



# Características

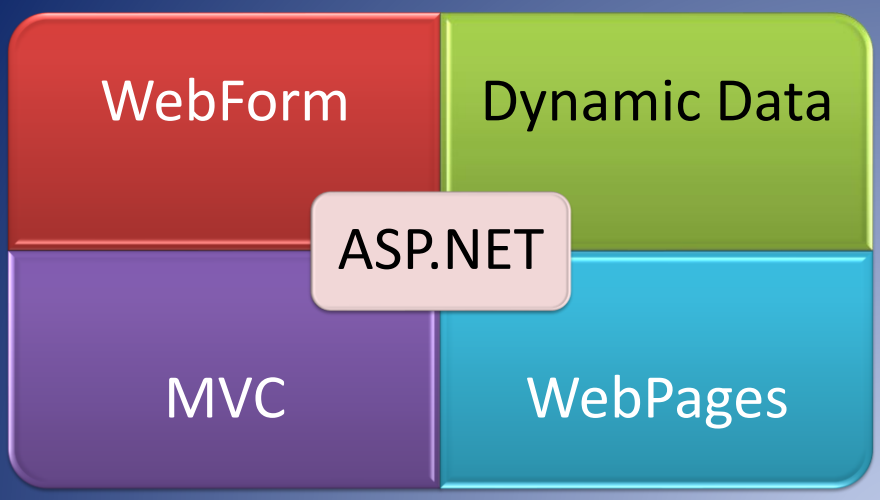
- El modelo MVC ayuda a crear aplicaciones que separan los diferentes aspectos de la aplicación (lógica de entrada, lógica de negocios y lógica de la interfaz de usuario), a la vez que proporciona un vago acoplamiento entre estos elementos.
- El modelo especifica dónde se debería encontrar cada tipo de lógica en la aplicación.
  - La lógica de la interfaz de usuario pertenece a la vista.
  - La lógica de entrada pertenece al controlador.
  - La lógica de negocios pertenece al modelo.
- Esta separación ayuda a administrar la complejidad al compilar una aplicación, ya que le permite centrarse en cada momento en un único aspecto de la implementación.
- El acoplamiento vago entre los tres componentes principales de una aplicación MVC también favorece el desarrollo paralelo.

## Puntos fuertes de MVC…

- Puntos fuertes de MVC…
  - Modelo muy simple de entender
  - Modelo muy cercano a la web
  - Admite una buena separación de responsabilidades
  - Permite la automatización de las pruebas unitarias de modelos y controladores.
- … y no tan fuertes
  - Mucha menos abstracción que WebForms
  - Curva de aprendizaje más alta



## Integración en ASP.NET



# Ventajas ASP.NET MVC

- Separación de responsabilidades
- Facilidad para unit testing
- Flexibilidad y extensibilidad
- Escalabilidad y rendimiento
- Uso de convenciones
- URL amigables
- Control total sobre el marcado
- Cercanía a la realidad de la web
- Integración natural con Ajax
- Construido sobre ASP.NET
- Es open source



# ¿Cuándo usar ASP.NET MVC?

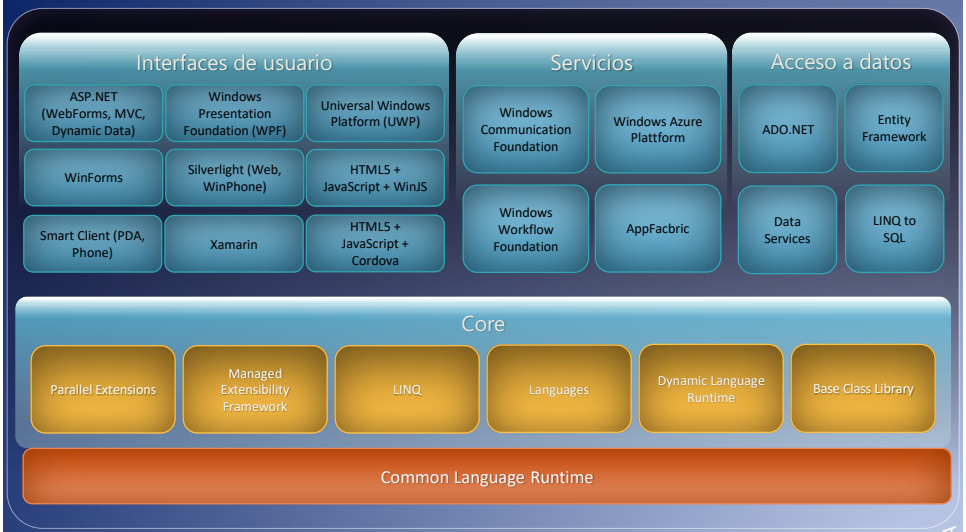
- Confortable para desarrolladores Web tradicionales
- No usa el estado de vista ni formularios basados en servidor. Esto hace que el marco de MVC sea ideal para los desarrolladores que deseen un control completo sobre el comportamiento de una aplicación y el Markup HTML
- Soporta Unit Testing y metodologías TDD (Test Driven Development) y Agile
- Alienta a crear Aplicaciones más Prescriptivas
- Extremadamente Flexible y Extensible
- Funciona bien para las aplicaciones web en las que trabajan equipos grandes de desarrolladores y para los diseñadores web que necesitan un alto grado de control sobre el comportamiento de la aplicación.

# ¿Cuándo usar ASP.NET WebForms?

- Familiar a programadores de clientes Windows: modelo de programación basado en controles y eventos
- Los controles encapsulan HTML, JS y CSS
- Incluye controles de UI ricos – datagrids, charts, AJAX
- Las diferencias entre navegadores son manejados por el sistema.
- En general, es menos complejo para el desarrollo de aplicaciones, puesto que los componentes se integran estrechamente y suelen requerir menos código que el modelo MVC.
- Funciona bien para los equipos pequeños de desarrolladores web y los diseñadores que deseen aprovechar el gran número de componentes disponible para el desarrollo rápido de aplicaciones.



# Mapa de tecnologías



Creación de proyectos

# ARQUITECTURA DE UNA APLICACIÓN MVC



## Tipos de Proyectos

| Plantilla | Descripción |
|---|---|
| Vacío | Proyecto de ASP.NET MVC 4 solo con las referencias y la estructura. |
| Básico | Proyecto básico de ASP.NET MVC 4. |
| Aplicación de Internet | Proyecto predeterminado de ASP.NET MVC 4 con un controlador de cuentas que usa la autenticación de formularios. |
| Aplicación de Intranet | Proyecto de ASP.NET MVC 4 predeterminado que usa autenticación de Windows. |
| Aplicación móvil | Proyecto de ASP.NET MVC 4 para dispositivos móviles con un controlador de cuentas que usa la autenticación de formularios. |
| Web API | Proyecto de ASP.NET Web API. |
| Aplicación de Facebook | Proyecto ASP.NET MVC 4 para una aplicación de Facebook. |

# Carpetas de MVC

| Carpeta | Propósito |
|---|---|
| Controllers | Controladores de responden a la entrada desde el navegador, decidir qué hacer con él y devolver la respuesta al usuario. |
| Models | Modelos de sostienen y manipulan datos |
| Views | Vistas sostener nuestras plantillas de interfaz de usuario |
| Views/Shared | Vistas y elementos compartidos |
| App_Start | Clases de configuración |
| Content | Esta carpeta contiene nuestras imágenes, CSS y cualquier otro contenido estático |
| Filters | Filtros de acción |
| Images | Ficheros con imágenes |
| Scripts | Esta carpeta contiene los archivos de nuestros JavaScript |
| Areas | Contienen las carpetas de las áreas y sus subcarpetas (Controllers, Views, Models) |



# Carpetas de ASP.NET

| Carpeta | Propósito |
|---|---|
| App_Browsers | Contiene definiciones del explorador (archivos .browser) que ASP.NET utiliza para identificar los exploradores individuales y determinar sus funciones. |
| App_Data | Contiene los archivos de datos de aplicación incluso los archivos MDF, archivos XML, así como otros archivos de almacén de datos. |
| App_GlobalResources | Contiene recursos (archivos .resx y .resources) que se compilan en los ensamblados con ámbito global. |
| App_LocalResources | Contiene recursos (archivos .resx y .resources) que están asociados con una página específica, control de usuario o página principal en una aplicación. |
| Bin | Contiene ensamblados compilados (archivos .dll) para los controles, componentes u otro código al que desea hacer referencia en su aplicación. |

# Archivos

- Global.asax : contiene código que se deriva de la clase HttpApplication y representa la aplicación.
- Web.config: configuración de la aplicación en XML.
- favicon.ico: icono de página.
- En la carpeta de App_Start las clases de registros son:
  - AuthConfig: Servicios de autenticación y validación de identidades.
  - Startup.Auth: Configuración OAuth para Google, Microsoft, Facebook y Twitter.
  - BundleConfig: Agrupar y minimizar/compactar CSS y JS (Jquery, Modernizr , …)
  - RouteConfig: Mapeo de la tabla de rutas.
  - FilterConfig: Los filtros globales se ejecutan para todas las acciones de todos los controladores.
  - WebApiConfig: Mapeo de los servicios WebAPI.



# Fases de ejecución de MVC

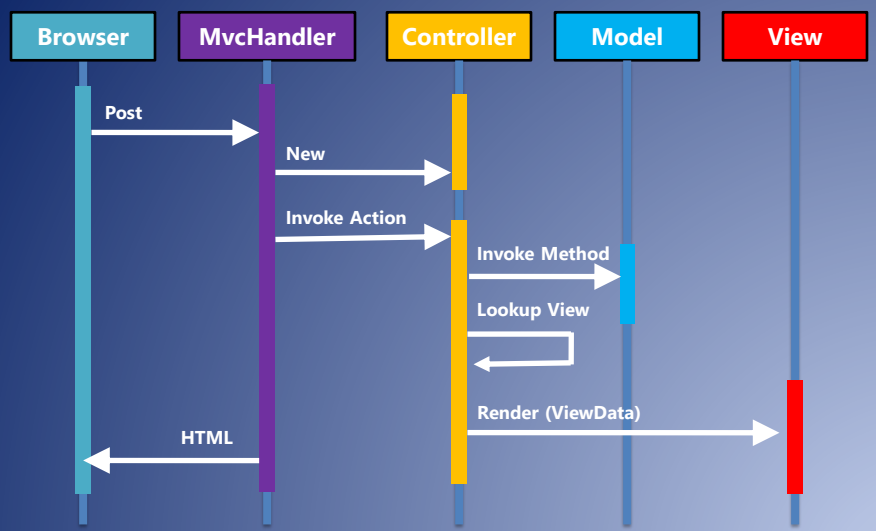
1. Recibir la primera solicitud para la aplicación
   En el archivo Global.asax, los objetos Route se agregan al objeto RouteTable.
2. Realizar el enrutamiento
   El módulo UrlRoutingModule utiliza el primer objeto Route coincidente de la colección RouteTable para crear el objeto RouteData, que a continuación utiliza para crear un objeto RequestContext.
3. Crear el controlador de solicitudes de MVC
   El objeto MvcRouteHandler crea una instancia de la clase MvcHandler y la pasa instancia RequestContext al controlador de eventos.
4. Crear el controlador
   El objeto MvcHandler utiliza la instancia RequestContext para identificar el objeto IControllerFactory (normalmente una instancia de la clase DefaultControllerFactory) para crear la instancia del controlador.
5. Ejecutar el controlador
   La instancia MvcHandler llama al método Execute del controlador.
6. Invocar la acción
   Para los controladores que heredan de la clase ControllerBase, el objeto ControllerActionInvoker que está asociado al controlador determina qué método de acción de la clase de controlador se llama y, a continuación, lo llama.
7. Ejecutar el resultado
   El método de acción recibe los datos proporcionados por el usuario, prepara los datos de respuesta adecuados y, a continuación, ejecuta el resultado devolviendo un tipo de resultado.

# Posibles resultados de la acción

| Resultado | Descripción |
|---|---|
| ViewResult | Representa una vista como una página web. |
| PartialViewResult | Representa una vista parcial, que define una sección de una vista que se puede representar dentro de otra vista. |
| RedirectResult | Redirecciona a otro método de acción utilizando su URL. |
| RedirectToRouteResult | Redirecciona a otro método de acción. |
| ContentResult | Devuelve un tipo de contenido definido por el usuario. |
| JsonResult | Devuelve un objeto JSON serializado. |
| JavaScriptResult | Devuelve un script que se puede ejecutar en el cliente. |
| FileResult | Devuelve la salida binaria para escribir en la respuesta. |
| EmptyResult | Representa un valor devuelto que se utiliza si el método de acción debe devolver un resultado null (vacío). |
| HttpStatusCodeResult | Proporciona un modo para devolver un resultado de la acción con un código de estado de respuesta HTTP. |



# El framework en funcionamiento



# ¿Cómo usamos componentes externos?

1. Localizar la página
2. Descargar la versión más reciente
3. Descomprimir
4. Añadir referencias
5. Leer documentación
6. Modificar configuración
7. ¿Hay dependencias?



# NuGet Package Manager

- Gestor de paquetes para desarrolladores
- Se instala como extensión
- Cuenta con una amplia base de datos actualizada de paquetes
- Simplifica el uso de componentes externos.
  - Localización
  - Descarga (¡con dependencias!)
  - Instalación / desinstalación
  - Configuración
  - Actualización
- Dispone de la consola de administración de paquetes.

Introducción

# MODELOS



## Modelo

- El modelo es la parte de la aplicación que es responsable de la aplicación básica o la lógica de negocio.
- Los objetos de modelo normalmente obtienen acceso a los datos desde un almacén persistente, como SQL Server, y realizan la lógica de negocios en esos datos.
- Los modelos son específicos de la aplicación y, por consiguiente, el marco de ASP.NET MVC no impone ninguna restricción sobre los tipos de objetos de modelo que se pueden generar.

# Opciones

- Se puede utilizar objetos DataReader o DataSet de ADO.NET o puede utilizar un conjunto personalizado de objetos de dominio.
- También puede utilizar una combinación de tipos de objeto para trabajar con datos.

- El modelo no es de una clase o interfaz determinada.
- Una clase forma parte del modelo no porque implemente una determinada interfaz o derive de una determinada clase base, sino debido al rol que la clase representa en la aplicación ASP.NET MVC y dónde la clase se encuentra en la estructura de carpetas de la aplicación.
- Una clase de modelo en una aplicación ASP.NET MVC no administra directamente la entrada del explorador ni genera la salida HTML al explorador.



# Lógica del dominio

- Los objetos del modelo son las partes de la aplicación que implementan la lógica del dominio, también conocida como la lógica de negocio.
- La lógica del dominio administra los datos que se pasan entre la base de datos y la interfaz de usuario.
- Por ejemplo, en un sistema de inventario, el modelo lleva un seguimiento de los elementos en almacenamiento y la lógica para determinar si un artículo está en inventario.
- Además, el modelo formaría parte de la aplicación que actualiza la base de datos cuando un artículo se vende y distribuye fuera del almacén.
- A menudo, el modelo también almacena y recupera el estado modelo en una base de datos.

# Arquitectura

- La carpeta recomendada para colocar las clases de modelo es la carpeta Models, proporcionada por Visual Studio en la plantilla de la aplicación ASP.NET MVC.
- Sin embargo, también es habitual colocar las clases modelo en un ensamblado independiente, para que puedan reutilizarse estas clases en aplicaciones diferentes.

- Utilizar las clases modelo con un controlador normalmente consiste en crear instancias de las clases modelo en acciones de controlador, llamando a métodos de los objetos modelo y extrayendo los datos adecuados de estos objetos para mostrarlos en vistas.
- Este es el enfoque recomendado para implementar las acciones. También mantiene la separación entre los elementos lógicos de la aplicación, lo que facilita probar la lógica de la aplicación sin tener que probarla mediante la interfaz de usuario.



Introducción

# CONTROLADORES

# Fundamentos

- El marco de ASP.NET MVC asigna direcciones URL a las clases a las que se hace referencia como controladores.
- Los controladores procesan solicitudes entrantes, controlan los datos proporcionados por el usuario y las interacciones, y ejecutan la lógica de la aplicación adecuada.
- Una clase de controlador llama normalmente a un componente de vista independiente para generar el marcado HTML para la solicitud.



# Implementación

- La clase base para todos los controladores es la clase ControllerBase, que proporciona el control general de MVC.
- La clase Controller hereda de ControllerBase y es la implementación predeterminada de un controlador.
- La clase Controller es responsable de las fases del procesamiento siguientes:
  - Localizar el método de acción adecuado para llamar y validar que se le puede llamar.
  - Obtener los valores para utilizar como argumentos del método de acción.
  - Controlar todos los errores que se puedan producir durante la ejecución del método de acción.
  - Proporcionar la clase WebFormViewEngine predeterminada para representar los tipos de página ASP.NET (vistas).
- Todas las clases de controlador deben llevar el sufijo "Controller" en su nombre.

# Métodos de acción

- El controlador define los métodos de acción: métodos públicos de la clase.
- Los controladores pueden incluir tantos métodos de acción como sea necesario.
- Los métodos de acción tienen normalmente una asignación unívoca con las interacciones del usuario.
- Los métodos del acción se invocan mediante solicitudes HTTP.
- Cada invocación cuenta con una dirección URL que MVC analiza usando las reglas de enrutamiento y determina la ruta de acceso del controlador.
- Con la dirección URL, el controlador determina el método de acción adecuado para administrar la solicitud.



# VISTAS

# Vista

- Una página de vista es una instancia de la clase ViewPage. Hereda de la clase Page e implementa la interfaz IViewDataContainer.
- La clase ViewPage define una propiedad ViewData que devuelve un objeto ViewDataDictionary. Esta propiedad contiene los datos que la vista debe mostrar.



# Clase de la View

```
public class WebViewPage<TModel> :
WebViewPage {
    public new AjaxHelper<TModel> Ajax { get; set; }
    public new HtmlHelper<TModel> Html { get; set; }
    public new TModel Model { get; }
    public new ViewDataDictionary<TModel> ViewData {
    get; set; }
}
```

# Razor

- Nuevo **motor de vistas**
- Originalmente aparece para las WebPage2
- Disponible en **WebMatrix**
- En MVC 3:
  - Se soporta Webforms…
  - pero el más **recomendado** es Razor.
- En MVC 4: Razor
- Sintaxis compacta y limpia
  - Menos **directivas**
  - **Integración** código-marcado más suave



# Sintaxis Razor

- Para agregar código de servidor a una página utilizando el carácter @
- Sintaxis compacta y limpia
  - Menos **directivas**
  - **Integración** código-marcado más suave
  - Combinación de texto, marcado y código en bloques de código
- Dos versiones: usando la sintaxis de C# o VB.NET
- Determinada por la extensión del fichero: .cshtml o .vbhtml

# Sintaxis Razor (C#)

- Selector de código Razor: @
- Bloque: @{ … }
- Comentario: @* … *@
- Instrucciones terminadas en ;
- Sensible a mayúsculas y minúsculas
- Acepta constantes literales de cadenas:
  var myFilePath = @"C:\MyFolder\";
- Acepta nombre cualificados: @class
- Operadores
  – . () [] = + - * / += -= == != < > <= >= ! && ||



# Variables

- Se declaran en un bloque
- Con tipo y sin tipo explicito
- Se recomienda la inferencia de tipos:
  @{ var variable=4; }
- Mostrar valores (inmerso en HTML o dentro de un bloque de código):
  <li>@variable</li> <li>@(variable+1)</li>
  <li>@objeto.propiedad</li>
  <li>@variable.metodo(1, 1)</li>

# Conversión

- Métodos de conversión y prueba:
  - AsBool(), IsBool(), AsInt(), IsInt(), AsFloat(), IsFloat(), AsDecimal(), IsDecimal(), AsDateTime(), IsDateTime(), ToString()
- HTML Encoding automático
- Conversión automática de rutas de acceso virtuales a físicos y URLs
  - var dataFilePath = "~/dataFile.txt";
  - @Server.MapPath(dataFilePath)
    C:\Websites\MyWebSite\datafile.txt
  - <img src="@Href("~/images")/Logo.jpg" />



# Bucles y lógica condicional

- Dentro de un bloque de código como una instrucción mas.
- Fuera de un bloque, auto comienzan el bloque:

```
@foreach (var myItem in Request.ServerVariables){
  <li>@myItem</li>
}
```

- Condicionales: if, switch
- Bucles: while, for, foreach
- Error: try/catch
- @using: espacio de nombre y uso

# HTML Helpers

- Son pequeños métodos que nos ayudan a escribir el HTML en las vistas.
- Son definidos como métodos extensores de la clase HtmlHelper.
- La vista expone una propiedad de tipo HtmlHelper con nombre Html.
- Esto nos permite usar una sintaxis del tipo Html.MiHelper(argumentos).
- La gracia de los helpers es que podemos crear los nuestros y usarlos siguiendo la misma sintaxis.



# Arquitectónicos

| Nombre | Descripción |
|---|---|
| Action | Invoca el método de acción secundario especificado y devuelve el resultado como una cadena HTML. |
| ActionLink | Devuelve un elemento delimitador (elemento a) que contiene la ruta de acceso virtual de la acción especificada. |
| RouteLink | Devuelve un elemento delimitador (elemento a) que contiene la ruta de acceso virtual de la acción especificada. |
| Partial | Presenta la vista parcial especificada como una cadena codificada en HTML. |
| RenderAction | Invoca un método de acción secundario especificado y presenta el resultado alineado en la vista primaria. |
| RenderPartial | Presenta la vista parcial especificada utilizando la aplicación auxiliar HMTL especificada. |
| BeginForm | Escribe una etiqueta de apertura <form> para la respuesta. Cuando el usuario envíe el formulario, un método de acción procesará la solicitud. |
| BeginRouteForm | Escribe una etiqueta de apertura <form> para la respuesta. Cuando el usuario envíe el formulario, el destino de ruta procesará la solicitud. |
| EndForm | Presenta la etiqueta </form> de cierre para la respuesta. |

# Plantillas

| Nombre | Descripción |
|---|---|
| Display | Devuelve el formato HTML para cada propiedad del objeto representado por una expresión de cadena. |
| DisplayFor | Devuelve el formato HTML de cada propiedad en el objeto representado por la expresión Expression. |
| DisplayForModel | Devuelve el formato HTML para cada propiedad en el modelo. |
| Editor | Devuelve un elemento input HTML para cada propiedad del objeto representado por la expresión. |
| EditorFor | Devuelve un elemento input de HTML para cada propiedad en el objeto representado por la expresión Expression. |
| EditorForModel | Devuelve un elemento input de HTML para cada propiedad en el modelo. |



# Visualización

| Nombre | Descripción |
|---|---|
| Label | Devuelve un elemento label de HTML y el nombre de la propiedad representada por la expresión especificada. |
| LabelFor | Devuelve un elemento label de HTML y el nombre de la propiedad representada por la expresión especificada. |
| LabelForModel | Devuelve un elemento label de HTML y el nombre de la propiedad representada por el modelo. |
| DisplayText | Devuelve el formato HTML para cada propiedad del objeto representado por la expresión especificada. |
| DisplayTextFor | Devuelve el formato HTML para cada propiedad del objeto representado por la expresión especificada. |

# Texto

| Nombre | Descripción |
|---|---|
| TextArea | Devuelve el elemento textarea especificado utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario. |
| TextAreaFor | Devuelve un elemento textarea HTML para cada propiedad del objeto representado por la expresión especificada. |
| TextBox | Devuelve un elemento input de texto utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario. |
| TextBoxFor | Devuelve un elemento input de texto para cada propiedad del objeto representada por la expresión especificada. |
| Password | Devuelve un elemento input de contraseña utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario. |
| PasswordFor | Devuelve un elemento input de contraseña para cada propiedad del objeto representado por la expresión especificada. |



# Selección

| Nombre | Descripción |
|---|---|
| CheckBox | Devuelve un elemento input de casilla utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario. |
| CheckBoxFor | Devuelve un elemento input de casilla para cada propiedad del objeto representado por la expresión especificada. |
| RadioButton | Devuelve un elemento input de botón de radio que se utiliza para presentar opciones mutuamente excluyentes. |
| RadioButtonFor | Devuelve un elemento input de botón de radio para cada propiedad del objeto representado por la expresión especificada. |
| DropDownList | Devuelve un elemento select de una sola selección utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario |
| DropDownListFor | Devuelve un elemento select HTML para cada propiedad en el objeto representado por la expresión especificada, usando los elementos de la lista especificados. |
| ListBox | Devuelve un elemento select de varias selecciones utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario. |
| ListBoxFor | Devuelve un elemento select HTML para cada propiedad del objeto representado por la expresión especificada y usando los elementos de lista especificados. |

# Especiales

| Nombre | Descripción |
|---|---|
| Hidden | Devuelve un elemento input oculto utilizando la aplicación auxiliar HTML especificada y el nombre del campo de formulario. |
| HiddenFor | Devuelve un elemento input oculto de HTML para cada propiedad del objeto representada por la expresión especificada. |
| Validate | Recupera los metadatos de validación para el modelo especificado y aplica cada regla al campo de datos. |
| ValidateFor | Recupera los metadatos de validación para el modelo especificado y aplica cada regla al campo de datos. |
| ValidationMessage | Muestra un mensaje de validación si existe un error para el campo especificado en el objeto ModelStateDictionary. |
| ValidationMessageFor | Devuelve el formato HTML para un mensaje de error de validación para cada campo de datos representado por la expresión especificada. |
| ValidationSummary | Devuelve una lista desordenada (elemento ul) de los mensajes de validación que están en el objeto ModelStateDictionary. |



# Ejemplo de Helpers

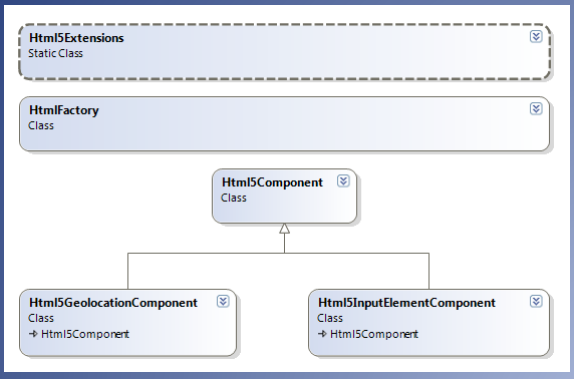
- Generar controles Web
  - CheckBox
  - Label
  - TextBox

```
@Html.CheckBox("checkbox");

@Html.Label("hola");

@Html.TextBox("textbox");
```

# Helpers de HTML5






# Helpers HTML5

- Input type number

```
@Html.Html5().Number(3)

@Html.Html5().Geolocation("onSucess")
```


- Geolocation

- accuracy : 20
- heading : null
- speed : null
- longitude : -3.720515099999999
- latitude : 40.41260949999999
- altitude : null
- altitudeAccuracy : null

# HTML5 Input Helper

```
public override void Render(HtmlTextWriter writer)
{
    TagBuilder input = new TagBuilder("input");
    input.GenerateId(this.Name);

    if (InputType.Equals(Html5InputTypes.DatetimeLocal))
    {
        input.MergeAttribute("type", "datetime-local");
    }
    else
    {
        input.MergeAttribute("type", InputType.ToString().ToLower());
    }

    if (!input.Attributes.ContainsKey("name"))
    {
        input.Attributes.Add("name", this.Name);
    }

    if (Value != null)
    {
        input.MergeAttribute("value", Value.ToString());
    }

    writer.Write(input.ToString(TagRenderMode.SelfClosing));

}
```



# System.Web.Helpers (Web Pages 2)

| Clase | Descripción |
|---|---|
| Chart | Muestra datos en forma de gráfico. |
| WebGrid | Muestra datos en una página web utilizando un elemento de tabla HTML. |
| WebImage | Representa un objeto que le permite mostrar y administrar imágenes en una página web. |
| ServerInfo | Muestra información acerca del entorno del servidor web que hospeda la página web actual. (Solo usar en depuración) |
| WebMail | Proporciona una manera de construir y enviar un mensaje de correo electrónico usando el SMTP. |
| Crypto | Proporciona métodos para generar valores hash y cifrar contraseñas u otros datos confidenciales. |
| Json | Proporciona métodos para trabajar con datos en formato JavaScript Object Notation (JSON). |
| WebCache | Proporciona una memoria caché para almacenar los datos a los que se accede frecuentemente. |

Ampliación

# CONTROLADORES



## Métodos de acción

- Todos los métodos públicos de la clase son métodos de acción
- Para evitar que un método público sea de acción se decora con [NonAction]
- Pueden tener parámetros.
- Se invocan mediante una URL (GET):
  ~/controlador/método/VPP?NP=VP&…
  - VPP: Valor del parámetro predeterminado (opcional)
  - NP=VP: Pares Nombre Parámetro = Valor Parámetro, opcionales, el primero precedido por ? y el resto por &.

# Parámetros de los métodos de acción

- Número de parámetros:
  - Ninguno
  - Uno, predeterminado: Por defecto “id” (definido al mapear la ruta).
    ~/controlador/método/1
  - Uno, no predeterminado: Requiere el par Nombre=Valor.
    ~/controlador/método/?p1=1
  - Varios: el primero puede ser el predeterminado, el resto requieren los pares Nombre=Valor
    ~/controlador/método/1?p2=2&p3=3
    ~/controlador/método/?p1=1&p2=2
- Los valores de los parámetros de método de acción se asignan automáticamente, se mapean por nombre.



# Métodos de acción

- Por defecto los parámetros se reciben vía GET, para indicar la recepción vía POST se decoran con [HttpPost].
- El decorador [ActionName] permite asignar un nombre al método de acción distinto al del nombre del método en la clase.
- [ValidateAntiForgeryToken] previene la falsificación de solicitud entre sitios (CSRF)

# Valor devuelto por el métodos de acción

- Se pueden crear métodos de acción que devuelven un objeto de cualquier tipo, como una cadena, un entero o un valor booleano.
- La mayoría de los métodos de acción devuelven una instancia de una clase derivada de ActionResult.
- La clase derivada de ActionResult determina el resultado de la acción.
- El controlador hereda métodos auxiliares que simplifican y automatizan la creación de los objetos ActionResult devueltos.
- El resultado más frecuente consiste en llamar al método View. El método View devuelve una instancia de la clase ViewResult, que se deriva de ActionResult.



# Tipos derivados de ActionResult

| Tipo | Método auxiliar | Descripción |
|---|---|---|
| ViewResult | View | Representa una vista como una página web. |
| PartialViewResult | PartialView | Representa una vista parcial, que define una sección de una vista que se puede representar dentro de otra vista. |
| RedirectResult | Redirect | Redirecciona a otro método de acción utilizando su URL. |
| RedirectToRouteResult | RedirectToAction<br>RedirectToRoute | Redirecciona a otro método de acción. |
| HttpStatusCodeResult | HttpNotFound | Proporciona un modo para devolver un resultado de la acción con un código de estado de respuesta HTTP. |

# Tipos derivados de ActionResult

| Resultado | Método auxiliar | Descripción |
|---|---|---|
| ContentResult | Content | Devuelve un tipo de contenido definido por el usuario. |
| JsonResult | Json | Devuelve un objeto JSON serializado. |
| JavaScriptResult | JavaScript | Devuelve un script que se puede ejecutar en el cliente. |
| FileResult | File | Devuelve la salida binaria para escribir en la respuesta. |
| EmptyResult | | Representa un valor devuelto que se utiliza si el método de acción debe devolver un resultado null (vacío). |



# Pasar datos a la vista

- El marco de ASP.NET MVC proporciona contenedores de nivel de página que pueden pasar datos entre controladores y vistas, débilmente tipados (sin tipo) o fuertemente tipados (modelo).
- La propiedad ViewData es un diccionario (claves/valor) para datos de vista.
- La propiedad ViewBag gestiona el diccionario de datos de vista dinámicos (declaración al vuelo).

  ViewBag.p = 1; → ViewData["p"] = 1;
- Vista y controlador comparten las dos propiedades (mismo nombre y al llamar al método View del controlador se asignan a la vista)

# Pasar datos fuertemente tipados

- La vista debe heredar de ViewPage<TModel> y establecer le propiedad Model (empezar por):
  ```
  @model Domain.Entity
  ```
- El modelo se pasa como parámetro al llamar al método View del controlador.
- La propiedad ModelState obtiene el objeto de diccionario de estados del modelo que contiene el estado del modelo y la validación de enlace del modelo.
- El modelo se puede recibir automáticamente:
  ```
  [HttpPost][ValidateAntiForgeryToken]
  public ActionResult Edit(Entity ent) {
    if (ModelState.IsValid) {
  ```
- O manualmente utilizando los métodos del controlador:
  - UpdateModel, TryUpdateModel: Actualiza la instancia de modelo especificada con los valores del proveedor de valores actual del controlador.
  - ValidateModel, TryValidateModel : Valida la instancia de modelo especificada.



# Pasar el estado entre métodos de acción

- Los métodos de acción puede que tengan que pasar datos a otra acción, cuando se produce un error al exponer un formulario, o si el método debe redirigir a métodos adicionales.
- La propiedad TempData es un diccionario (claves/valor) para datos temporales.
- El valor de la propiedad TempData se almacena en el estado de la sesión (proveedor de datos temporales predeterminado) y se conserva hasta que se lea o hasta que se agote el tiempo de espera de la sesión.
- El proveedor de datos temporales se establece en Controller.TempDataProvider.
- Cualquier método de acción al que se llame después de establecer su valor puede obtener valores del objeto y, a continuación, procesarlos o mostrarlos.
- Conservar TempData de esta manera, habilita escenarios como la redirección, porque los valores de TempData están disponibles para más de una única solicitud.

# Errores

- Para agregar el mensaje de error especificado a la colección de errores para el diccionario de modelo-estado asociado a la clave especificada.
  - ModelState.AddModelError()
- Se pueden interceptar las excepciones sobrescribiendo el método OnException y generar una vista especifica pare el error.
- Para excepciones no tratada se pueden crear filtros de excepciones.



# Contexto de ejecución

| Propiedad | Descripción |
|---|---|
| ActionInvoker | Obtiene el invocador de acción para el controlador. |
| ControllerContext | Obtiene o establece el contexto del controlador. |
| HttpContext | Obtiene la información específica de HTTP sobre una solicitud HTTP individual. |
| ModelState | Obtiene el objeto de diccionario de estados del modelo que contiene el estado del modelo y la validación de enlace del modelo. |
| Request | Obtiene el objeto HttpRequestBase de la solicitud HTTP actual. |
| Response | Obtiene el objeto HttpResponseBase de la respuesta HTTP actual. |
| RouteData | Obtiene los datos de ruta de la solicitud actual. |
| Server | Obtiene el objeto HttpServerUtilityBase que proporciona los métodos que se utilizan durante el procesamiento de solicitudes web. |
| Session | Obtiene el objeto HttpSessionStateBase de la solicitud HTTP actual. |
| TempData | Obtiene o establece el diccionario para datos temporales. |
| TempDataProvider | Obtiene el objeto de proveedor de datos temporales que se utiliza para almacenar los datos para la solicitud siguiente. |
| Url | Obtiene el objeto auxiliar de direcciones URL que se usa para generar las direcciones URL mediante el enrutamiento. |
| User | Obtiene la información de seguridad del usuario para la solicitud HTTP actual. |
| ValidateRequest | Obtiene o establece un valor que indica si está habilitada la validación de solicitudes para esta solicitud. |

# Enlazado

- Las capacidades de binding de ASP MVC conforman un potente mecanismo mediante el cual, de manera automática, se obtienen valores para los parámetros de las acciones que se ejecutan buscándolos en la petición que llega del cliente, así como las propiedades del modelo.
- El binding es el proceso que trata de emparejar los valores y parámetros enviados desde el cliente con los parámetros de las acciones que se ejecutan en el servidor.
- Una vez selecciona una acción, se analizan los nombres de los parámetros de la acción y trata de encontrar parejas entre ellos y los campos de formularios, parámetros de ruta, parámetros GET o archivos adjuntos (siguiendo el orden indicado).



# Enlazado

- Por defecto, convenio de nombres:
  - Nombre GET → parámetro del método de acción
  - Nombre POST → parámetro del método de acción
- Tipos complejos:
  - Nombre POST → propiedad del parámetro
- Tipos complejos que contienen propiedades complejas:
  - Nombre POST → propiedad de la propiedad del parámetro
- Tipo lista
  - Nombre POST[n] → parámetro[n]
- Ficheros
  - Nombre POST → parámetro HttpPostedFileBase

# Personalizar el enlazado

- El decorador Bind se usa para proporcionar detalles cómo debe producir el enlace del modelo a un parámetro.
  - Prefix: Prefijo que se va a usa cuando se represente el marcado para enlazar a un argumento de acción o a una propiedad compleja de modelo.
  - Exclude: Lista delimitada por comas de nombres de propiedades para las que no se permite el enlace.
  - Include: Lista delimitada por comas de nombres de propiedades para las que se permite el enlace (el resto se excluye).
- La creación de Binders personalizados (clase que herede de DefaultModelBinder) proporciona un control total del enlazado. Se decora el parámetro con: [ModelBinder(typeof(MiBinder))]



# Model Binders

- Extensión del framework que permite crear instancias de clases en base a valores enviados por request.
  - Al action llega el objeto instanciado y no los valores del request
- Nos permite participar del ciclo de vida de creación de la instancia permitiéndonos por ejemplo validar los atributos y agregar mensajes de error invalidando el modelo

# Enrutamiento

- Las rutas mapean las URL a los métodos de acción de los controladores y sus parámetros.
- Separadas en fragmentos (/), compuestos por variables ({…}) y/o constantes.
- Ruta por defecto (RouteConfig.cs):

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



# Rutas personalizadas

- Soporta tantas rutas como sean necesarias pero deben estar registradas.
- El orden del mapeo importa, las mas especificas primero y las mas generales las últimas (la ruta por defecto la última)
- Toda ruta debe conocer su {controller} y su {action}, en caso de no aparecer en el patrón se deben definir sus valores por defecto.
- MVC enlaza por nombre: nombre variable → nombre parámetro.
- El resto de las variables pueden ser:
  - Obligatorias: sin valor por defecto, deben aparecer en la URL.
  - Opcionales con valor: tienen valor por defecto, si no aparecen en la URL el parámetro toma el valor por defecto.
  - Opcionales sin valor: tienen UrlParameter.Optional como valor por defecto, si no aparecen en la URL el parámetro toma el valor NULL.

# Rutas personalizadas

- Mediante expresiones regulares se pueden establecer restricciones a los valores de las variables y al mecanismo de recepción GET/POST (HttpMethodConstraint).

```
routes.MapRoute(
        name: "Informes",
        url: "informe/{locale}/{year}",
        defaults: new {
           controller = "Reports", action = "Show",
           locale = "es-es", year = DateTime.Now.Year.ToString() },
        constraints: new {
           locale = "[a-z]{2}-[a-z]{2}", year = @"\d{4}",
           httpMethod = "POST" });
```



# Atributos de Enrutado

- Activar en RouteConfig

  routes.MapMvcAttributeRoutes();
- Ruta particular para una acción:

  [Route(“reviews/{reviewId}/edit”)]

  public ActionResult Edit(int reviewId) { … }
- Parámetros opcionales:

  [Route(“books/{isbn?}”)]

  public ActionResult View(string isbn)
- Restricciones a los parámetros:
  - {x:alpha} {x:bool} {x:datetime} {x:decimal} {x:double} {x:float} {x:guid} {x:int} {x:length(6)} {x:length(1,20)} {x:long} {x:max(10)} {x:maxlength(10)} {x:min(10)} {x:minlength(10)} {x:range(10,50)} {x:regex(^\d{3}-\d{3}-\d{4}$)}

# Atributos de Enrutado

- Fijar prefijo para la clase y rutas relativas:

```
[RoutePrefix("reviews")]
[Route("{action=index}")]
public class ReviewsController : Controller {
  [Route] // ej.: /reviews
   public ActionResult Index() { … }
   [Route("{reviewId}")] // ej.: /reviews/5
   public ActionResult Show(int reviewId) { … }
```



# Ciclo de ejecución de una acción

| Método | Descripción |
|---|---|
| OnAuthorization | Se le llama cuando se produce la autorización. |
| OnActionExecuting | Se le llama antes de invocar al método de acción. |
| | Ejecución del método de acción |
| OnActionExecuted | Se le llama después de invocar al método de acción. |
| OnResultExecuting | Se le llama antes de ejecutar el resultado de la acción |
| OnResultExecuted | Se le llama después de ejecutar el resultado de la acción devuelto por un método de acción. |
| OnException | Se le llama cuando se produce una excepción no controlada en la acción. |

# Filtrado

- Los controladores definen métodos de acción que normalmente tienen una relación uno a uno con las posibles interacciones del usuario, como hacer clic en un vínculo o enviar un formulario.
- A veces se desea ejecutar la lógica antes de llamar a un método de acción o después de ejecutar un método de acción.
- Los filtros son clases personalizadas que proporcionan un método declarativo y de programación para agregar el comportamiento previo y posterior a la acción a los métodos de acción del controlador.



# Tipos de filtros de acción

- Filtros de autorización (IAuthorizationFilter): Toman decisiones de seguridad sobre si se debe ejecutar un método de acción, por ejemplo realizar la autenticación o validar las propiedades de la solicitud. Los filtros de autorización se ejecutan antes que cualquier otro filtro.
- Filtros de acción (IActionFilter): Ajustan la ejecución del método de acción y pueden llevar a cabo procesos adicionales, como proporcionar datos extra al método de acción, conversiones, inspeccionar el valor devuelto o cancelar la ejecución del método de acción.
- Filtros de resultados (IResultFilter) Ajustan la ejecución del objeto ActionResult y puede llevar a cabo procesos adicionales del resultado, como modificar la respuesta HTTP.
- Filtros de excepciones (IExceptionFilter) Se ejecutan si se produce una excepción no controlada durante la ejecución de la canalización de ASP.NET MVC. Los filtros de excepciones se pueden usar para tareas como registrar o mostrar una página de error.

# Filtros proporcionados en ASP.NET MVC

- **Authorize**: Restringe el acceso mediante autenticación y opcionalmente mediante autorización.
- **AllowAnonymous**: Marca controladores y acciones para omitir Authorize durante la autorización.
- **RequireHttps**: Obliga a reenviar las solicitudes HTTP no seguras sobre HTTPS.
- **AcceptVerbs**: Especifica a qué verbos HTTP responderá un método de acción (Con **HttpGet, HttpPost, HttpPut, HttpHead, HttpPatch, HttpOptions** y **HttpDelete** el método solo administra las solicitudes HTTP del verbo).
- **ValidateAntiForgeryToken**: Impide la falsificación de una solicitud.
- **ActionMethodSelector**: Se usa para influir en la selección de un método de acción.
- **ActionNameSelector**: Afecta a la selección de un método de acción.
- **AllowHtml**: Permite que una solicitud incluya el formato HTML durante el enlace del modelo al omitir la validación de solicitudes de la propiedad.
- **ValidateInput**: Marca métodos de acción cuya entrada se debe validar.
- **OutputCache**: Proporciona almacenamiento en caché de resultados.
- **HandleError**: Especifica cómo controlar una excepción generada por un método de acción.

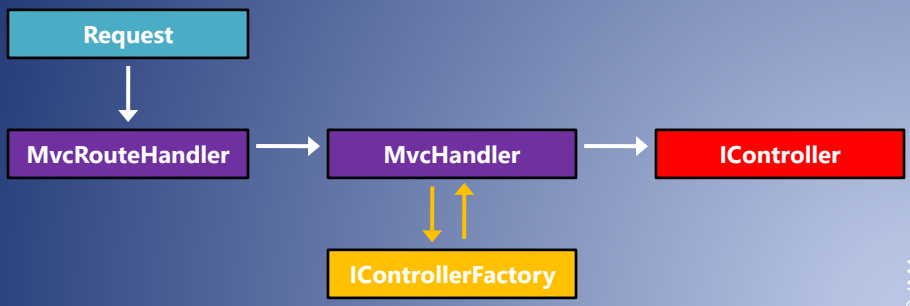


# Filtros de acción

- Se crean como clases derivadas de ActionFilterAttribute que implementan los correspondientes interfaces.
- Se deben registrar con el proveedor de filtros (FilterProviders).
- Se utilizan como decoradores del controlador o de los correspondientes métodos de acción.
- Permiten la reutilización de la lógica de presentación
- Los filtros se ejecutan en el siguiente orden:
  1. Filtros de autorización
  2. Filtros de acción
  3. Filtros de la respuesta
  4. Filtros de excepciones
- Utilizados en
  - Validaciones y carga de datos en el viewData
  - "Anti-leeching " para evitar que las imágenes o paginas se carguen o enlacen en páginas que no se hallan en su sitio web.
  - Localización para establecer la configuración regional.

# Controller Factory

- Necesitamos cambiar la manera de instanciar controllers
  - Por ejemplo: Para integrar la solución con un IoC container y poder inyectar dependencias






Ampliación

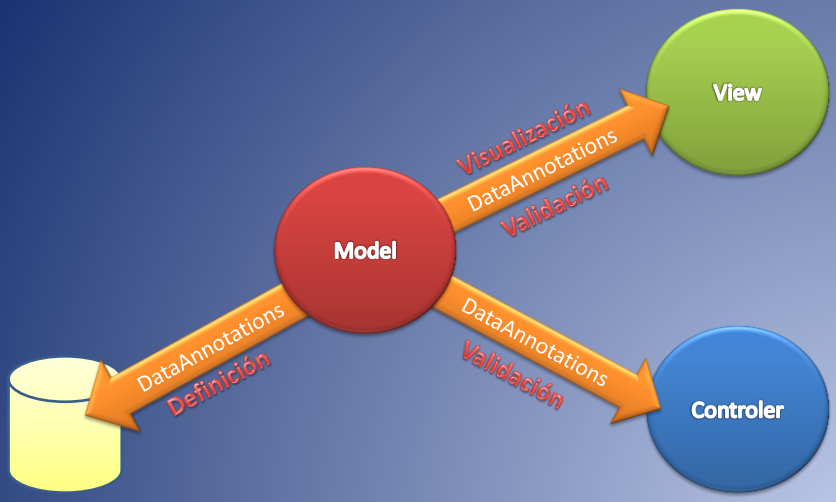
# MODELOS

# DataAnnotations

- System.ComponentModel.DataAnnotations;
- Permiten definir los metadatos de los modelos de datos para su uso por entidades externas.
- Cuenta con decoradores para:
  - Definición del modelo de datos: claves (PK), asociaciones (FK), simultaneidad, …
  - Interfaz de usuario y localización: Título, descripción, formato, solo lectura, …
  - Validaciones y errores personalizados: Obligatoriedad, longitudes, formatos, …



# Contexto Declarativo



# Visualización

| Decorador | Descripción |
|---|---|
| DataType | Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos. |
| Display | Permite especificar las cadenas traducibles de los tipos y miembros de las clases parciales de entidad. |
| DisplayFormat | Especifica el modo en que los datos dinámicos de ASP.NET muestran y dan formato a los campos de datos. |
| ScaffoldColumn | Especifica si una clase o columna de datos usa la técnica scaffolding. |
| ScaffoldTable | Especifica si una clase o tabla de datos usa la técnica scaffolding. |
| UIHint | Especifica la plantilla o el control de usuario que los datos dinámicos usan para mostrar un campo de datos. |
| **System.Web.Mvc** | |
| HiddenInput | Indica que una propiedad se debería presentar como un elemento input oculto. |



# [Display]

- **Name**: valor que se usa para mostrarlo en la interfaz de usuario (Título, Etiqueta, …).
- **Description**: valor que se usa para mostrar una descripción en la interfaz de usuario.
- **Prompt**: valor que se usará para establecer la marca de agua para los avisos en la interfaz de usuario.
- **ShortName**: valor que se usa para la etiqueta de columna de la cuadrícula.
- **GroupName**: valor que se usa para agrupar campos en la interfaz de usuario.
- **Order**: número del orden de la columna.

# Tipos asociados

| Asociado | Descripción |
|---|---|
| DateTime | Representa un instante de tiempo, expresado en forma de fecha y hora del día. |
| Date | Representa un valor de fecha. |
| Time | Representa un valor de hora. |
| Duration | Representa una cantidad de tiempo continua durante la que existe un objeto. |
| PhoneNumber | Representa un valor de número de teléfono. |
| Currency | Representa un valor de divisa. |
| Text | Representa texto que se muestra. |
| Html | Representa un archivo HTML. |
| MultilineText | Representa texto multilínea. |
| EmailAddress | Representa una dirección de correo electrónico. |
| Password | Represente un valor de contraseña. |
| Url | Representa un valor de dirección URL. |
| ImageUrl | Representa una URL en una imagen. |
| CreditCard | Representa un número de tarjeta de crédito. |
| PostalCode | Representa un código postal. |
| Upload | Representa el tipo de datos de la carga de archivos. |



# Validación

| Decorador | Descripción |
|---|---|
| Required | Especifica que un campo de datos necesita un valor. |
| DataType | Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos. Decoradores especializados: CreditCard, EmailAddress, EnumDataType, FileExtensions, Phone, Url |
| Compare | Compara dos propiedades. |
| Range | Especifica las restricciones de intervalo numérico para el valor de un campo de datos. |
| StringLength | Especifica la longitud mínima y máxima de caracteres que se permiten en un campo de datos. Decoradores especializados: MinLength, MaxLength |
| RegularExpression | Especifica que un valor de campo de datos debe coincidir con la expresión regular especificada. |
| CustomValidation | Especifica un método de validación personalizado que se utiliza para validar una propiedad o una instancia de clase. |

# Validación manual de objetos

- En System.ComponentModel.DataAnnotations, la clase estática Validator ofrece métodos que permiten realizar las comprobaciones de forma directa sobre objetos o propiedades concretas.

```
IEnumerable<ValidationResult> getValidationErrors(object obj)    {
    var validationResults = new List<ValidationResult>();
    var context = new ValidationContext(obj, null, null);
    Validator.TryValidateObject(obj,
            context,
            validationResults,
            true);
    return validationResults;
}
```



# IValidatableObject

- Obliga a implementar un único método, llamado Validate(), que determina si el objeto especificado es válido.
- Será invocado automáticamente por TryValidateObject() siempre que no encuentre errores al comprobar las restricciones especificadas mediante anotaciones.
- Devolverá una lista de objetos ValidationResult con los resultados de las comprobaciones.
- En las clases  prescriptivas (EF) se aplica con una partial class.
- Interfaces relacionados: IDataErrorInfo, INotifyDataErrorInfo

# Anotar clases ya existentes

- Se requiere una clase auxiliar con las propiedades a decorar decoradas.
- Declarativa: [MetadataType(typeof(tipo))]
  - Se aplica con una partial class (en el mismo ensamblado que la clase a anotar).
- Imperativa: System.ComponentModel.TypeDescriptor y AssociatedMetadataTypeTypeDescriptionProvider:

```
var descriptionProvider = new
    AssociatedMetadataTypeTypeDescriptionProvider(
      typeof(Friend),  typeof(FriendMetadata));
TypeDescriptor.AddProviderTransparent(
    descriptionProvider, typeof(Friend));
```



# En la vista

- Validación en cliente:

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

- Mostrar errores:
  - @Html.ValidationSummary(true)
  - @Html.ValidationMessageFor(model => model.id)

# ModelView

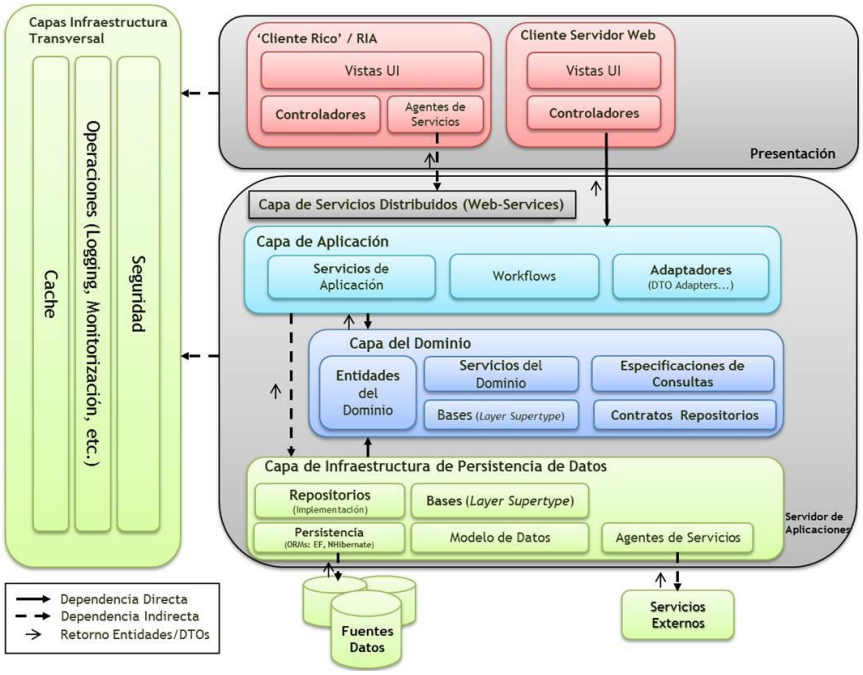
- Los modelos representan entidades de dominio (capa de dominio), que pueden requerir ampliaciones para su uso en capa de presentación.
- Los modelos de la capa de presentación se denominan comúnmente como ModelViews, modelos de las vistas.
- No se deben confundir con los ViewModels del patrón MVVM.
- Los ModelViews son clases (con el sufijo ModelView) que suelen encapsular al modelo de dominio a través de una propiedad.
- Los controladores pasan los ModelViews a las vistas en sustitución de los modelos.



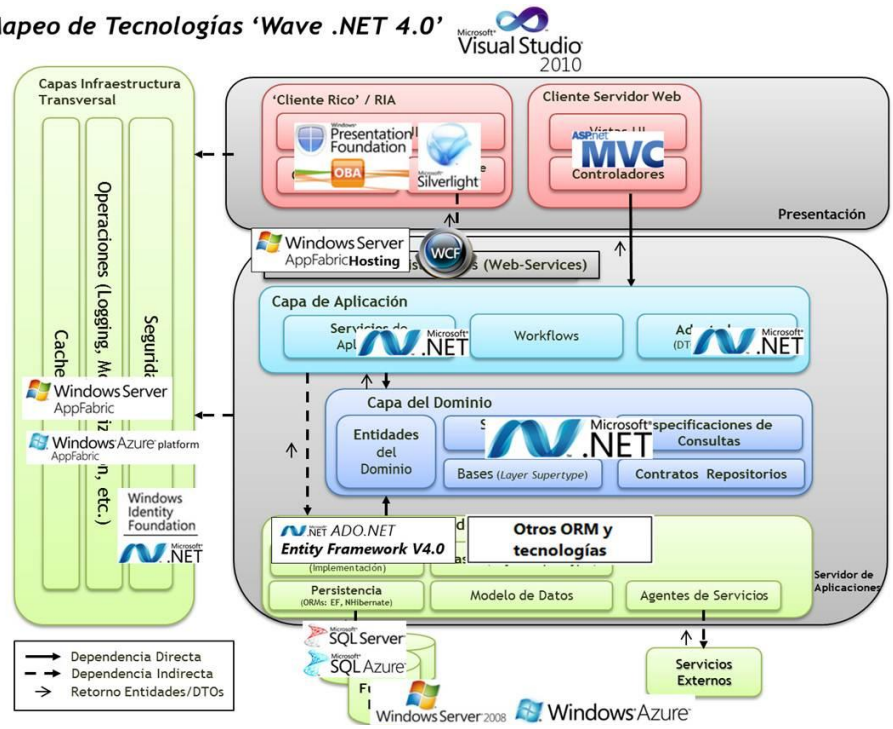
# Repositorio

- Un repositorio separa la lógica empresarial de las interacciones con la base de datos subyacente y centra el acceso a datos en un área, lo que facilita su creación y mantenimiento.
- El repositorio pertenecen a la capa de infraestructura y devuelve los objetos del modelo de dominio.
- Deberían implementar el patrón de doble herencia.
- Forma parte de los Domain Driven Design patterns: Domain Entity, Value-Object, Aggregates, Repository, Unit of Work, Specification, Dependency Injection, Inversion of Control (IoC).

## *Arquitectura N-Capas con Orientación al Dominio*




## *Mapeo de Tecnologías 'Wave .NET 4.0'*



# Ventajas del Repositorio

- Proporciona un punto de sustitución para las pruebas unitarias. Es fácil probar la lógica empresarial sin una base de datos y otras dependencias externas.
- Las consultas y los modelos de acceso a datos duplicados se pueden quitar y refactorizar en el repositorio.
- Los métodos del controlador pueden usar parámetros fuertemente tipados, lo que significa que el compilador encontrará errores de tipado de datos en cada compilación en lugar de realizar la búsqueda de errores de tipado de datos en tiempo de ejecución durante las pruebas.
- El acceso a datos está centralizado, lo que brinda las siguientes ventajas:
  - Mayor separación de intereses (SoC), uno de los principios de MVC, lo que facilita más el mantenimiento y la legibilidad.
  - Implementación simplificada de un almacenamiento en caché de datos centralizado.
  - Arquitectura más flexible y con menor acoplamiento, que se puede adaptar a medida que el diseño global de la aplicación evoluciona.
- El comportamiento se puede asociar a los datos relacionados (calcular campos, aplicar relaciones, reglas de negocios complejas entre los elementos de datos de una entidad, …).
- Un modelo de dominio se puede aplicar para simplificar una lógica empresarial compleja.



# Ampliación

- Domain Oriented N-Layered .NET 4.0 Sample App
  - http://microsoftnlayerapp.codeplex.com/

# PÁGINAS MAESTRAS, PLANTILLAS Y CSS



## Página Maestra

- Contiene las partes comunes de todas las paginas (concepto de layout).
- La vista es parte del cuerpo y se introduce a través de @RenderBody()
- Por defecto esta establecido en /Views/_ViewStart.cshtml

```
@{ Layout = "~/Views/Shared/_Layout.cshtml"; }
```

- Para establecerlo a nivel de vista:
  - @{ this.Layout = "…URL…"; }
  - @{ this.Layout = null; }

# Partes de Vistas

- Comúnmente conocidas como vistas parciales.
- Contienen un fragmento reutilizable de la vista: Etiquetas Razor y marcas HTML.
- Se almacenan en ficheros separados.
- Para facilitar su identificación, habitualmente su nombre va precedido por _
- No deben gestionar la página maestra (layout)
- Se insertan en la vista con:
  - @Html.Partial("MyPartial")
  - @Html.Action("MetodoAcción")
    - Marcado con [ChildActionOnly]
    - return PartialView(…);



# Secciones

- Las secciones establecen puntos de representación en la página maestra que se diseñan en las vistas.
- Pueden ser opcionales u obligatorias
- En la página maestra:

```
@RenderSection("featured", required: false)
@if (IsSectionDefined("featured"))
```

- En la vista:

```
@section featured {
...
}
```

- O la página maestra puede incluir vistas parciales:

```
@Html.Partial("_LoginPartial")
```

# Plantillas

- Representación visual (HTML) de los tipos de datos.
- Plantillas especificas:
  - <ControllerName>/DisplayTemplates/<TemplateName>.cshtml
    Utilizadas por los extensores Display de HtmlHelper
  - <ControllerName>/EditorTemplates/<TemplateName>.cshtml
    Utilizadas por los extensores Editor de HtmlHelper
- Plantillas comunes:
  - Shared/DisplayTemplates/<TemplateName>.cshtml
  - Shared/EditorTemplates/<TemplateName>.cshtml
- La plantilla utilizada viene determinada por el TemplateName, que representa el nombre de un:
  - Tipo especifico: Tipo de datos de la propiedad.
  - Tipo asociado: Modifica con [DataType(…)] el tipo de la propiedad.
  - Plantilla: Se establece con [UIHint("…")] en la propiedad.
  - Modelo: Tipo de la clase Model (DataTemplate).



# Tipos asociados

| Asociado | Descripción |
|---|---|
| Custom | Representa un tipo de datos personalizado. |
| DateTime | Representa un instante de tiempo, expresado en forma de fecha y hora del día. |
| Date | Representa un valor de fecha. |
| Time | Representa un valor de hora. |
| Duration | Representa una cantidad de tiempo continua durante la que existe un objeto. |
| PhoneNumber | Representa un valor de número de teléfono. |
| Currency | Representa un valor de divisa. |
| Text | Representa texto que se muestra. |
| Html | Representa un archivo HTML. |
| MultilineText | Representa texto multilínea. |
| EmailAddress | Representa una dirección de correo electrónico. |
| Password | Represente un valor de contraseña. |
| Url | Representa un valor de dirección URL. |
| ImageUrl | Representa una URL en una imagen. |
| CreditCard | Representa un número de tarjeta de crédito. |
| PostalCode | Representa un código postal. |
| Upload | Representa el tipo de datos de la carga de archivos. |

# EditorTemplates

```
@model DateTime
@{
   List<SelectListItem> days = new List<SelectListItem>();
   for (int i = 1; i <= 31; i++) {
      days.Add(new SelectListItem() { Text = i.ToString(), Value = i.ToString(), Selected = (i == Model.Day ?
true : false) });
   }
   List < SelectListItem > months = new List<SelectListItem>();
   for (int i = 1; i <= 12; i++) {
      months.Add(new SelectListItem() { Text = i.ToString(), Value = i.ToString(), Selected = (i ==
Model.Month ? true : false) });
   }
   List < SelectListItem > years = new List<SelectListItem>();
   int prevYearCount = ViewBag.PreviousYearCount ?? 100;
   int nextYearCount = ViewBag.NextYearCount ?? 0;
   for (int i = Model.Year - prevYearCount; i <= Model.Year + nextYearCount; i++) {
      years.Add(new SelectListItem() { Text = i.ToString(), Value = i.ToString(), Selected = (i == Model.Year ?
true : false) });
   }
}
@Html.DropDownList("days", days) @Html.DropDownList("months", months)
@Html.DropDownList("years", years)
```



RWD – Responsive Web Design

# DISEÑO WEB ADAPTATIVO

# Diseño Adaptativo

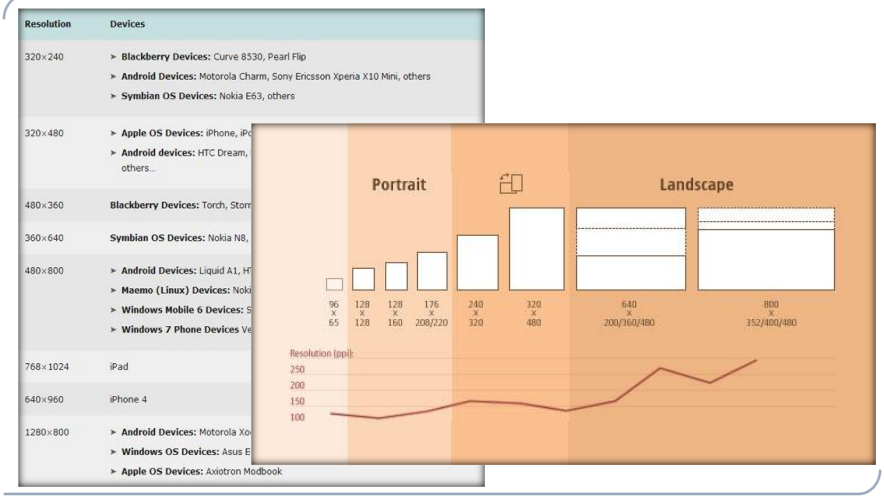
- Es un enfoque de diseño destinado a la elaboración de sitios/aplicaciones para proporcionar un entorno óptimo de:
  - Lectura Fácil
  - Navegación correcta con un número mínimo de cambio de tamaño
  - Planificaciones y desplazamientos
- Con la irrupción multitud de nuevos dispositivos y el que el acceso a internet se realiza ya mayoritariamente desde dispositivos diferentes a los tradicionales ordenadores ha obligado a seguir dicho enfoque en las aplicaciones WEB.
- Contempla la definición de múltiples elementos antes de la realización de la programación real.
  - Elementos de página en las unidades de medidas correctas
  - Imágenes flexibles
  - Utilización de CSS dependiendo de la aplicación



# Resolución

- Los dispositivos móviles tienen una característica distintiva, y es su resolución de pantalla.
- Es necesario conocer cuales son las resoluciones más comunes en este tipo de dispositivos móviles, de los gadgets más utilizados, etc.
- Las resoluciones van cambiando de forma muy rápida y en dispositivos nuevos

# Resolución






# Resolución

- También deberemos tener en cuenta la resoluciones de otros dispositivos como:
  - Tablets
  - TV SmartTV
  - Pizarras electrónicas, etc

| | |
|---|---|
| **Pizarras** | **10.1" y 11.6" (2560x1440, 1920x1080, 1366x768), 17" (1920x1080)** |
| **PC** | **12" (1280x800), 14" (1920x1080, 1366x768), 15.6" (1920x1080)** |
| **Family hub** | **23" (1920x1080), 27" (2560x1440)** |

# Orientación de Página

- La orientación del papel es la forma en la que una página rectangular está orientada y es visualizada.
- Los dos tipos más comunes son:
  - Landscape (Horizontal)
  - Portrait (Vertical)



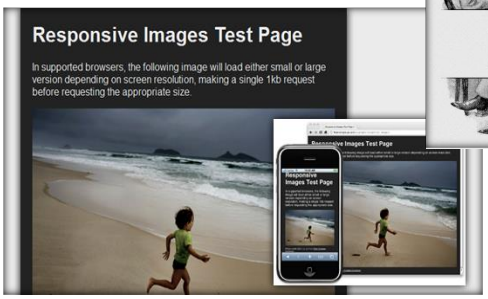
# Recomendaciones de Diseño

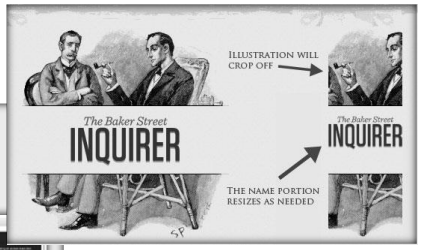
1. Utilizar porcentajes y “ems" como unidad de medida en lugar de utilizar los valores determinados como definición de pixel.

**Las ems son unidades relativas**, así que más exactamente 1 em equivale al cien por cien del tamaño inicial de fuente.

# Recomendaciones de Diseño

2. Determinar el tamaño y definición de las imágenes a utilizar
   - Recortar, Ajustar, etc






# Recomendaciones de Diseño

3. El contenido y funcionalidad BÁSICA debe de ser accesible por todos los navegadores

# Recomendaciones de Diseño

4. Definición correcta de contenidos en función del dispositivo




# Ventajas

- Soporte de dispositivos móviles.
- Con una sola versión en HTML y CSS se cubren todas las resoluciones de pantalla.
- Mejora la experiencia de usuario.
- Se reducen los costos de creación y mantenimiento cuando el diseño de las pantallas es similar entre dispositivos de distintos tamaños.
- Evita tener que desarrollar aplicaciones especificas para cada sistema operativo móvil.
- Facilita la referenciación y posicionamiento en buscadores, versión única contenido/página.

# DESARROLLO PARA LOS DISPOSITIVOS MÓVILES



## Diseño Adaptativo

- Uno de los objetivos de cualquier desarrollo que se realice es que sea usado por la mayor cantidad de personas posibles.
- Este objetivo se complica cuando se desea que se pueda ver en navegadores de escritorio/tableta y en navegadores de los móviles.
- Por ello surge el concepto de *diseño Adaptativo* que permite, siguiendo unas pautas, que un mismo contenido se ajuste al tamaño de pantalla que lo visualice.

# Representación adaptable

- Etiqueta meta de viewport
  <meta name="viewport" content="width=device-width" />
- Posicionamiento relativo (flujo) frente a absoluto.
- Unidades relativas frente a fijas.
- Consultas de medios de CSS 3

```
@media only screen and (max-width: 850px) {
    header { float: none; }
}
```

- Filtros de propiedades por navegador en CSS3:

```
-webkit-box-sizing:border-box;
```



# Vistas específicas

- MVC 4 permite reemplazar las vistas generales por vistas para dispositivos móviles con el uso de una convención de nombres.
  [view].mobile.cshtml
- Permite la entrega de vistas específicas para exploradores mediante los Modos de presentación. Es necesario registrarlos en Application_Start de Global.asax

```
DisplayModeProvider.Instance.Modes.Insert(0,
  new DefaultDisplayMode("iPhone")  {
      ContextCondition = (context =>
        context.GetOverriddenUserAgent().IndexOf(
          "iPhone", StringComparison.OrdinalIgnoreCase) >= 0)});
```

  [view].iPhone.cshtml
- jQuery Mobile es una biblioteca de código abierto basada en jQuery Core para construir interfaces de usuario para dispositivos móviles.
- La plantilla de proyecto de aplicación móvil de ASP.NET MVC 4 es para sitios que se enfocan únicamente en los exploradores móviles.

# Empezando con la versión para móvil.

- Agregar jQuery Mobile con NuGET
- Agregar una nueva plantilla maestra (Layout) para la versión móvil.
- Según la estructura:
  - Por áreas:
    - Añadir área
    - Configurar la tabla de rutas (mAreaRegistration)
    - Añadir cada controlador con sus vistas
  - Por modos:
    - Registrar los Modos en Application_Start de Global.asax
    - Añadir las versiones especificas de las vistas cuando sea necesario.



# Área para móviles

- Imperativo:

```
if(Request.Browser.IsMobileDevice)
```

- Registro del área:

```
context.MapRoute(
    "Mobile_default",
    "Mobile/{controller}/{action}/{id}",
    new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
  );
```

- Filtro de acción:

```
[RedirectMobileDevicesToMobileArea] // Applies just to this action
  public ActionResult Index()  {
```

- Activar filtro de área:

```
protected void Application_Start() {
GlobalFilters.Filters.Add(new
RedirectMobileDevicesToMobileAreaAttribute(), 1);
```

# Referencias

- Patrones y guías: Building Modern Mobile Web Apps
  - http://msdn.microsoft.com/en-us/library/hh994907.aspx



REpresentational State Transfer

# SERVICIOS RESTFUL

# REST (REpresentational State Transfer)

- Un **estilo de arquitectura** para desarrollar aplicaciones web distribuidas que se basa en el uso del protocolo HTTP e Hypermedia.
- Definido en el 2000 por Roy Fielding, para no reinventar la rueda, se basa en aprovechar lo que ya estaba definido en el HTTP pero que no se utilizaba.
- El HTTP ya define 8 métodos (algunas veces referidos como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado:
  - HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT
- El HTTP permite en el encabezado transmitir la información de comportamiento:
  - Accept, Content-type, Response (códigos de estado), Authorization, Cache-control, …



# Uso de la cabecera

- **Request**: Método /uri?parámetros
  - GET: Recupera el recurso
    - Todos: Sin parámetros
    - Uno: Con parámetros
  - POST: Crea un nuevo recurso
  - PUT: Edita el recurso
  - DELETE: Elimina el recurso
- **Accept**: Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- **Content-type**: Indica en que formato está codificado el cuerpo, utilizando MIME
- **Response**: Código de estado con el que el servidor informa del resultado de la petición.

# Peticiones

Request: GET /users
  Response: 200
  content-type:application/json
Request: GET /users/11
  Response: 200
  content-type:application/json
Request: POST /users
  Response: 201 Created
  content-type:application/json
  body
Request: PUT /users/11
  Response: 200
  content-type:application/json
  body
Request: DELETE /users/11
  Response: 204 No Content

*Fake Online REST API*
*for Testing and Prototyping*
https://jsonplaceholder.typicode.com/



# GraphQL

http://graphql.org/

- GraphQL es un lenguaje para ejecutar consultas en un API de servidor mediante el cual se provee al cliente de una descripción detallada de las estructuras de datos que ofrece.
- Este lenguaje fue creado por Facebook en 2012, cuya estandarización se comenzó a desarrollar en 2015, por lo que es un lenguaje en continuo desarrollo.
- Las principales características y funcionalidades que ofrece GraphQL son las siguientes:
  - Describe la estructura/organización de los datos ofrecidos por el API mediante un sistema de fuertemente tipado en el que se detallan los recursos y las relaciones existentes entre estos.
  - Permite solicitar al servidor tan sólo la información que necesitamos de un recurso en una sola petición incluyendo los datos de otros recursos que se relacionen con el recurso raíz.
  - Ofrece una única URI en la que se realizan todas las peticiones, por lo que no debemos preocuparnos de la semántica implementada por cada API.
  - Es un lenguaje independiente de la fuente de datos. Los datos pueden obtenerse de un servicio web, una base de datos, un fichero, etc.
  - Permite la evolución de un API sin crear nuevas versiones
  - Existen múltiples librerías para implementar en distintos lenguajes (JavaScript, .NET, Ruby, Java …)

# GraphQL

**Definición de datos**

```
type Human implements Character {
 id: ID!
 name: String!
 friends: [Character]
 appearsIn: [Episode]!
 starships: [Starship]
 totalCredits: Int
}
enum Episode {
 NEWHOPE
 EMPIRE
 JEDI
}
type Starship {
 id: ID!
 name: String!
 length(unit: LengthUnit = METER): Float
}
```

**Consulta**

```
{
 human(id: 1002) {
  name
  appearsIn
  starships {
   name
  }
 }
}
```



# GraphQL

**Resultado**

```
{
 "data": {
  "human": {
   "name": "Han Solo",
   "appearsIn": [
    "NEWHOPE",
    "EMPIRE",
    "JEDI"
   ],
   "starships": [
    {
     "name": "Millenium Falcon"
    },
    {
     "name": "Imperial shuttle"
    }
   ]
  }
 }
}
```

# Patrón Agregado (Aggregate)

- Una Agregación es un grupo de objetos asociados que deben tratarse como una unidad a la hora de manipular sus datos.
- El patrón Agregado es ampliamente utilizado en los modelos de datos basados en Diseños Orientados al Dominio (DDD).
- Proporciona un forma de encapsular nuestras entidades y los accesos y relaciones que se establecen entre las mismas de manera que se simplifique la complejidad del sistema en la medida de lo posible.
- Cada Agregación cuenta con una Entidad Raíz (root) y una Frontera (boundary):
  - La Entidad Raíz es una Entidad contenida en la Agregación de la que colgarán el resto de entidades del agregado y será el único punto de entrada a la Agregación.
  - La Frontera define qué está dentro de la Agregación y qué no.
- La Agregación es la unidad de persistencia, se recupera toda y se almacena toda.



## SERVICIOS WEB API

# Introducción

- Una de las principales novedades de la nueva versión de ASP MVC 4 es la Web API, mediante la cual podemos crear una auténtica capa de servicios REST de manera muy similar a como veníamos desarrollando los controladores hasta ahora.
- Es posible construir una capa de servicios simplemente retornando un JsonResult en las distintas acciones de los controladores.
- La Web API se encarga de realizar este trabajo de forma específica, ofreciendo unas funcionalidades más avanzadas y apropiadas para crear este tipo de capas de servicios de tipo REST.



# Arquitectura

- La Web API está pensada para el trabajo con HTTP y aunque su funcionamiento es muy similar al de cualquier controlador, es cierto que hay también bastantes diferencias.
- Los servicios Web API deben heredar de la clase ApiController de System.Web.Http.
- Los controladores Web API se mapean de una forma similar a los controladores MVC.
- Ruta por defecto (WebApiConfig.cs):

```
config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
```

# Verbos HTTP

- No se basan en los métodos de acción, se basa en los verbos HTTP (8 verbos):
  - GET: Pide una representación del recurso especificado.
  - HEAD: Pide una representación del recurso especificado.
  - POST: Envía los datos en el cuerpo de la petición para que sean procesados.
  - PUT: Sube, carga o realiza un upload a un recurso especificado.
  - DELETE: Borra el recurso especificado.
  - TRACE: Solicita al servidor que envíe de vuelta en el mensaje completado el cuerpo de entidad con la traza de la solicitud.
  - OPTIONS: Devuelve los métodos HTTP que el servidor soporta para un URL específico.
  - CONNECT: Se utiliza para saber si se tiene acceso a un host.



# RESTful - CRUD

- Acrónimo de Create, Read, Update, Delete (altas, bajas, modificaciones y consultas)
- Se debe crear un método de acción por cada verbo salvo la consulta que requiere dos métodos: uno sin parámetros para obtener todos y otro con parámetro para obtener solo uno.
- La correspondencia entre verbos HTTP y métodos se puede realizar de dos formas según se usen:
  - Prefijos: El nombre del método comienza con el nombre del verbo que implementa.
  - Decoradores: Hay libertad en la elección del nombre del método pero deben ir precedidos por el decorador del correspondiente verbo.

# Verbos y métodos

| Acción | Verbo | Prefijo | Decorador | Parámetros | Retorno |
|---|---|---|---|---|---|
| Consulta todos | GET | Get | HttpGet | Ninguno | IEnumerable<Entidad> ó HttpResponseException |
| Consulta uno | GET | Get | HttpGet | Id(Clave) | Entidad ó HttpResponseException |
| Añadir | POST | Post | HttpPost | Entidad | HttpResponseMessage con HttpStatusCode |
| Modificar | PUT | Put | HttpPut | Entidad | HttpResponseMessage con HttpStatusCode |
| Borrar | DELETE | Delete | HttpDelete | Id(Clave) | HttpResponseMessage con HttpStatusCode |



# Uso con JQuery AJAX

```
$.ajax({
        type: "PUT",
        url: "/api/miwebapi/" + $('#editId').val(),
        data: $('#editForm').serialize(),
        success: function (result) {
                if (result) {
                        …
                }
        }
});
```

# Seguridad

- La ejecución de aplicaciones JavaScript puede suponer un riesgo para el usuario que permite su ejecución.
- Por este motivo, los navegadores restringen la ejecución de todo código JavaScript a un entorno de ejecución limitado.
- Las aplicaciones JavaScript no pueden establecer conexiones de red con dominios distintos al dominio en el que se aloja la aplicación JavaScript.
- Los navegadores emplean un método estricto para diferenciar entre dos dominios ya que no permiten ni subdominios ni otros protocolos ni otros puertos.
- Si el código JavaScript se descarga desde la siguiente URL: http://www.ejemplo.com
- Las funciones y métodos incluidos en ese código no pueden acceder a:
  - **https**://www.ejemplo.com/scripts/codigo2.js
  - http://www.ejemplo.com**:8080**/scripts/codigo2.js
  - http://**scripts**.ejemplo.com/codigo2.js
  - http://**192.168.0.1**/scripts/codigo2.js
- La propiedad document.domain se emplea para permitir el acceso entre subdominios del dominio principal de la aplicación.



# CORS

- Un recurso hace una solicitud HTTP de origen cruzado cuando solicita otro recurso de un dominio distinto al que pertenece y, por razones de seguridad, los exploradores restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script.
- XMLHttpRequest sigue la política de mismo-origen, por lo que, una aplicación usando XMLHttpRequest solo puede hacer solicitudes HTTP a su propio dominio. Para mejorar las aplicaciones web, los desarrolladores pidieron a los proveedores de navegadores que permitieran a XMLHttpRequest realizar solicitudes de dominio cruzado.
- El Grupo de Trabajo de Aplicaciones Web del W3C recomienda el nuevo mecanismo de Intercambio de Recursos de Origen Cruzado (CORS, Cross-origin resource sharing). Los servidores deben indicar al navegador mediante cabeceras si aceptan peticiones cruzadas y con que características:
  - "Access-Control-Allow-Origin", "*"
  - "Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept"
  - "Access-Control-Allow-Methods", "GET, POST, PUT, DELETE“
- Soporte: Chrome 3+ Firefox 3.5+ Opera 12+ Safari 4+ Internet Explorer 8+

# Desactivar la seguridad de Chrome

- Pasos para Windows:
  - Localizar el acceso directo al navegador (icono) y crear una copia como “Chrome Desarrollo”.
  - Botón derecho -> Propiedades -> Destino
  - Editar el destino añadiendo el parámetro al final. ej: "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-web-security
  - Aceptar el cambio y lanzar Chrome
- Para desactivar parcialmente la seguridad:

  --allow-file-access

  --allow-file-access-from-files

  --allow-cross-origin-auth-prompt
- Referencia a otros parametros:
  - http://peter.sh/experiments/chromium-command-line-switches/



# Habilitar solicitudes entre orígenes cruzados en ASP.NET Web API 2

- https://docs.microsoft.com/es-es/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api
- Instalación:
  - Install-Package Microsoft.AspNet.WebApi.Cors
- En el archivo de App_Start/WebApiConfig.vb

```
Public Module WebApiConfig
    Public Sub Register(ByVal config As HttpConfiguration)
        config.EnableCors
```

- Para habilitar la CORS por cada acción o por controlador

```
<System.Web.Http.Cors.EnableCors("*", "Origin, Content-Type, Accept", "GET,
        POST, PUT, DELETE")>
```

- Puede habilitar la CORS por cada acción, por controlador o globalmente para todos los controladores

```
config.EnableCors(New System.Web.Http.Cors.EnableCorsAttribute("*", "Origin,
        Content-Type, Accept", "GET, POST, PUT, DELETE"))
```

AJAX

# CÓDIGO EN EL LADO DEL CLIENTE



## JavaScript no obstrusivo

- Todas las aplicaciones que se desarrollen deberían cumplir la normativa de **accesibilidad** vigente.
- Una de las normas indica que no se debe introducir eventos dentro de los elementos HTML.
- Los validadores lo detectan y avisan que no se podrá interactuar con determinados elementos que el usuario que accede desconoce, que no puede visualizar correctamente, etc.
- Alternativa: Usar atributos personalizados (no estándar) dentro de la etiqueta HTML que serán sustituidos al cargar el documento por el código JavaScript correspondiente (jquery.unobtrusive):
  - Validaciones: data-val-number, data-val-required, data-valmsg-for, data-valmsg-replace, …
  - AJAX: data-ajax, data-ajax-begin, data-ajax-complete, data-ajax-failure, data-ajax-loading, data-ajax-loading-duration, data-ajax-mode, data-ajax-update, data-ajax-url, …

# Infraestructura

- En el <appSettings> del Web.config:

  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
- En el <head> del _Layout.cshtml
  - <script src="~/Scripts/jquery-1.7.1.min.js" type="text/javascript"></script>
  - <script src="~/Scripts/jquery.unobtrusive-ajax.min.js" type="text/javascript"></script>
- En la vista:

```
@section Scripts {
   <script src="~/Scripts/jquery.unobtrusive-ajax.min.js"
type="text/javascript"></script>
}
```



# AjaxHelper

- La clase AjaxHelper incluye métodos que proporcionan funcionalidad de cliente en ASP.NET AJAX en aplicaciones de MVC, tal como la creación de formularios asíncronos y la representación de vínculos.
- La clase AjaxHelper admite la actualización asíncrona de páginas parciales.
- Las extensiones a la clase AjaxHelper se encuentran en el espacio de nombres System.Web.Mvc.Ajax.
- Los métodos y las extensiones auxiliares se llaman mediante la propiedad Ajax de la vista, que es una instancia de la clase AjaxHelper.

# Configuración: AjaxOptions

- **Url**: dirección URL en la que se va a hacer la solicitud.
- **HttpMethod**: método de solicitud HTTP ("Get" o "***Post***").
- **UpdateTargetId**: identificador del elemento DOM que se va a actualizar utilizando la respuesta del servidor.
- **InsertionMode**: modo que especifica cómo insertar la respuesta en el elemento DOM de destino ("InsertAfter", "InsertBefore" o "***Replace***").
- **Confirm**: mensaje que se mostrará en una ventana de confirmación antes de que se envíe una solicitud.
- **LoadingElementId**: el atributo id de un elemento HTML que se muestra mientras se está cargando la función Ajax.
- **LoadingElementDuration**: valor, en milisegundos, que controla la duración de la animación cuando se muestra u oculta el elemento que se está cargando.



# AjaxOptions (Eventos)

- **OnBegin**: nombre de la función JavaScript a la que se llama inmediatamente antes de que se actualice la página.
- **OnComplete**: nombre de la función JavaScript a la que se llama cuando se ha creado una instancia de datos de la respuesta pero antes de que se actualice la página.
- **OnSuccess**: nombre de la función JavaScript a la que se llama una vez que la página se actualiza correctamente.
- **OnFailure**: nombre de la función JavaScript a la que se llama si no se puede actualizar la página.

# Métodos de extensión AjaxHelper

- El método ActionLink representa un elemento delimitador (a) que se vincula a un método de acción.
- El método RouteLink representa un elemento delimitador (a) que se vincula a una URL y que puede resolverse en un método de acción, un archivo, una carpeta o en otro recurso.
- Los métodos BeginForm y BeginRouteForm pueden ayudar en la creación de formularios HTML compatibles con las funciones AJAX.



# Estrategias en el uso de Ajax

- Para realizar la invocación las estrategias son:
  - Uso de hipervínculos
  - Uso de formularios
  - Uso del JQuery Ajax.
- Para el tratamiento de la respuesta se pueden emplear también dos estrategias:
  - Vistas parciales: La invocación devuelve una vista parcial que sustituye a la actual.
  - JSON: La invocación devuelve JSON que debe ser tratado en JavaScript para actualizar el contenido de la página actual.

# Formularios Ajax (PartialView)

- En el controlador:
  - Crear un método de acción
  - Debe devolver PartialViewResult (se puede declarar como ActionResult )
  - Para devolver la vista se utiliza el método PartialView del controlador:

```
public PartialViewResult metodoAccion(
… ) {
    …
    return PartialView(data);
}
```



# Formularios Ajax (PartialView)

- En la vista:
  - Identificar la etiqueta que va a contener el resultado y asignar contenido inicial:

```
<div id="rsltAjax">@Html.Action(…)</div>
```

  - Preparar opciones Ajax:

```
AjaxOptions ajaxOpts = new AjaxOptions {
      UpdateTargetId="rsltAjax",
      Url=Url.Action(…), … };
```

  - Crear el formulario:

```
@using (Ajax.BeginForm(ajaxOpts)) {
<div>
    …
    <button type="submit">Enviar</button>
</div>
}
```

# Enlaces Ajax

- Crear el método de acción en el controlador
- En la vista:
  - Preparar e inicializar el contenedor del resultado.
  - Configurar la opciones Ajax.
  - Crear el enlace:

```
@Ajax.ActionLink(
      "Texto interno del hipervinculo.",
      "Nombre del método de la acción.",
      ajaxOpts, … )
```



# JSON

- JSON, acrónimo de JavaScript Object Notation, es un formato ligero de texto para el intercambio de datos (http://www.json.org/).
- JSON es un subconjunto de la notación literal de objetos de JavaScript (pero independiente) que no requiere el uso de XML (de mayor potencia y complejidad).
- Las implementaciones de XMLHttpRequest permiten cualquier codificación basada en texto, incluyendo: texto plano, XML, JSON y HTML.
- En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval().

# Sintaxis JSON

- JSON permite definir dos tipos de estructuras básicas:
  - Colecciones de pares nombre/valor: para representar objetos, registros, estructuras, diccionarios, tablas hash, listas con clave, o matrices asociativa.
    { nombre : valor, … }
  - Listas ordenadas de valores: para representar tablas, vectores, listas o secuencias.
    [ valor, … ]
- Un valor puede ser una cadena (entre comillas dobles y pueden secuencias de escape), un número (el punto como separador decimal), ***true***, ***false***, ***null*** u otras estructuras (estructuras anidadas).
- En JavaScript:

```
miVariable = eval('(' + datosJSON + ')');
```



# Controlador JSON

- Crear un método de acción
- Debe devolver JsonResult (se puede declarar como ActionResult )
- Para devolver el valor serializado en JSON se utiliza el método Json del controlador (por defecto POST, para peticiones GET hay que habilitarlas utilizando el parámetro JsonRequestBehavior.AllowGet ):

```
public JsonResult metodoAccion( … ) {
    …
    return Json(data,
       JsonRequestBehavior.AllowGet);
}
```

# Vista AJAX con JSON

- Preparar e inicializar el contenedor del resultado.
- Crear una función Java que procese el resultado (convertir JSON en HTML):

```
<script type="text/javascript">
function procesaJSON(data) {
    var target = $("#rsltAjax");
    target.empty();
    for (var i = 0; i < data.length; i++) {
            var modelo = data[i];
            target.append( ... );
    }
}
</script>
```

- Configurar la opciones Ajax, referenciando la función que procesa el resultado con la propiedad OnSuccess.

```
AjaxOptions ajaxOpts = new AjaxOptions  { … , OnSuccess =
"procesaJSON", … };
```

- Crear el formulario o enlace de invocación.



# Controladores adaptativos

- Devuelven HTML o JSON en función del tipo de invocción:

```
public ActionResult metodoAccion(…) {
    …
    if (Request.IsAjaxRequest()) {
        return Json(data,
        JsonRequestBehavior.AllowGet);
    } else {
        return PartialView(data);
    }
}
```

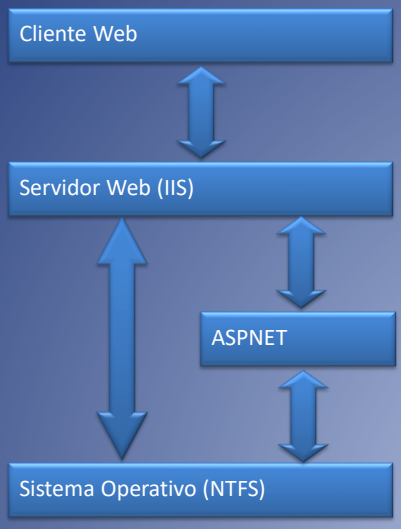
# Aplicaciones en página

- Patrón: Single Page Application (SPA)
- Knockout.js es una biblioteca JavaScript, que ayuda a implementar el modelo MVVM. Permite:
  - Realizar enlazados declarativos
  - Refresco automático de los elementos del UI, cuando se actualiza el modelo-vista se actualiza automáticamente
  - Seguimiento de Dependencias, detecta los cambios realizados en la vista o en el modelo y es capaz de propagarlos .
  - Permite generar rápidamente plantillas en función de los datos del modelo-vista.
- Patrones y guías: Project Silk: Client-Side Web Development for Modern Browsers
  - http://msdn.microsoft.com/en-us/library/hh396380.aspx



# SEGURIDAD DE UNA APLICACIÓN WEB

# Arquitectura ASP.NET






# Seguridad ASP.NET

- ASP.NET, conjuntamente con Microsoft Internet Information Services (IIS), puede autenticar las credenciales del usuario como nombres y contraseñas mediante los métodos de autenticación siguientes:
  - Windows: básica, implícita, y Autenticación de Windows integrada (NTLM o Kerberos).
  - Autenticación mediante formularios, con la que crea una página de inicio de sesión y se administra la autenticación en la aplicación.
  - Autenticación mediante certificados de cliente
- EN ASP.NET existen dos formas de autorizar el acceso a un recurso dado:
  - Autorización de archivos: Realiza una comprobación de la lista de control de acceso (ACL) del archivo.
  - Autorización de URL : Realiza la autorización de URL, que asigna usuarios y funciones a direcciones URL en aplicaciones ASP.NET.

# Autenticación Windows

- Ventajas:
  - Usa la infraestructura de Windows existente
  - Controla el acceso a información confidencial y se integra con el sistema operativo
  - Mayor nivel de seguridad
- Desventajas:
  - No admite todos los tipos de clientes
  - Requiere relaciones de confianza
- Habitualmente se restringe a Intranet.
- A nivel de aplicación MVC solo requiere su configuración en el Web.config de la raíz:
  - <authentication mode="Windows"/>



# Autenticación de formularios

- Ventajas:
  - Usa su propia infraestructura que se puede personalizar
  - Admite todos los tipos de clientes (cualquier S.O. y dominio, incluyendo la autenticación delegada)
- Desventajas:
  - Menor nivel de seguridad
  - Se basa en cookies
  - Requiere comunicaciones seguras dado que el usuario y la contraseña se transmiten en abierto ([RequireHttps])
- Recomendable para su uso en Internet.
- Las plantillas de MVC suministran el controlador, los modelos y las vistas para su gestión y personalización.

# Autenticación de formularios

- Controlador: AccountController
  - Login, LogOff
  - Register, Disassociate, Manage
- Modelos:
  - UserProfile, LoginModel, LocalPasswordModel, RegisterModel, ExternalLogin, RegisterExternalLoginModel
- Vistas:
  - Login, Register, Manage, ExternalLoginFailure, ExternalLoginConfirmation
  - _ChangePasswordPartial, _ExternalLoginsListPartial, _RemoveExternalLoginsPartial, _SetPasswordPartial



# Infraestructura

- Suministrada por .NET (System.Security.Principal):
  - WindowsIdentity y WindowsPrincipal
  - GenericIdentity y GenericPrincipal
- Suministrada por MVC (WebMatrix.WebData):
  - SimpleMembershipProvider y SimpleRoleProvider
  - WebSecurity
  - Filters/InitializeSimpleMembershipAttribute
- Suministrada por ASP.NET (System.Web.Security)
  - MembershipProvider y RoleProvider
  - FormsAuthentication, FormsAuthenticationTicket, FormsIdentity

# OAuth/OpenID

- ASP.NET permite la delegación de la autentificación en credenciales de Microsoft Live, Facebook, Twitter y Google.
- Como paso previo, salvo en Google, es necesario ir a los diferentes proveedores a registrar la aplicación/URL para obtener un identificador y una contraseña/código secreto (según el proveedor).
- El identificador/contraseña deben registrarse en AuthConfig para que estén disponibles en la creación de una cuenta local.



# Configuración

- Preparación de la base de datos:
  - %WINROOT%\Microsoft.NET\Framework\vX.X.XXXXX\aspnet_regsql.exe
- En el <system.web> del Web.config:
  - <authentication mode="Forms" >
    - <forms loginUrl="~/Account/Login" timeout="2880" />
  - <membership ><providers>
- En el <connectionStrings> del Web.config:
  - <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;Initial Catalog=aspnet-MvcMovie-20130424112038;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\aspnet-MvcMovie-20130424112038.mdf" providerName="System.Data.SqlClient" />

# Autorización

- NO se debe usar los archivos web.config o el direccionamiento para asegurar una aplicación MVC.
- En MVC se deben proteger los controladores y los métodos de acción.

- Las posibilidades disponibles son:
  - Utilizar el atributo [Authorize] en los Controllers.
    - Indica que método de acción o controlador (todos sus métodos de acción) requiere autenticación y a que usuarios (no recomendado) o roles se le permite el acceso (todos por defecto).
  - Utilizar un Filtro de autorización global.
    - Sí en RegisterGlobalFilters se añade AuthorizeAttribute como filtro global, todos los métodos de acción de la aplicación requieren autenticación .
- Con el decorador [AllowAnonymous] se indica que un método de acción o controlador no requiere autenticación.



# Personalización de Vistas y Controladores

- Request.IsAuthenticated: indica si la solicitud se ha autenticado.
- User: Obtiene la información de seguridad del usuario para la solicitud HTTP actual.
  - Identity: Obtiene la identidad de la entidad de seguridad actual.
    - Name: Obtiene el nombre del usuario actual.
    - AuthenticationType: Obtiene el tipo de autenticación utilizado.
  - IsInRole(): Determina si la entidad de seguridad actual pertenece al rol especificado.

Disponibles a través de las propiedades de contexto de las vistas y los controladores

# XSS: Cross-Site Scripting

- Técnica utilizada para inyectar código malicioso ejecutable en las aplicaciones, utiliza las entradas de texto de los formularios que posteriormente se mostrarán en las páginas.
- ASP.NET protege por defecto del XSS, validando de entrada de datos para impedir que contenga HTML (y scripting) y formateando la salida como texto (HTML Encoding).

- Para que los usuarios puedan insertar texto enriquecido se debe utilizar los siguientes recursos:
  - Decorar las propiedades de las clases del modelo con [AllowHtml]
  - Utilizar en las vistas el Helper @Html.Raw(Propiedad)
- Se recomienda el uso de la biblioteca AntiXSS (en Nuget Packages) para "sanear" los datos de entrada que puedan contener código malicioso (validación positiva).



# CSRF: Cross Site Request Forgery

- Al contrario del XSS que explota la confianza del usuario en un sitio en particular, explota la confianza del sitio en un usuario en particular.
- El CSRF utiliza a un usuario ya autenticado en un sitio para, a través de este, introducir solicitudes "válidas" al sitio en su nombre sin su conocimiento (siempre que mantenga abierta la sesión) mediante formularios y campos ocultos.
- Mitigación:
  - Habilitar cierre de sesión por el usuario
  - Control estricto del Session.TimeOut
  - Supervisión del atributo Referer de la cabecera HTTP.
  - Usar un Token dinámico:
    - En la vista (Helper): @Html.AntiForgeryToken()
    - En el método de acción (Decorador): [ValidateAntiForgeryToken]

## Inyección SQL

- Se produce al generar consultas SQL que se concatenan con entradas de usuario no validadas y su posterior ejecución en el motor de la base de datos.
- No se producen cuando se usa EF y LINQ
- Se evitan al utilizar consultas parametrizádas.
- Se debe establecer una política de privilegios mínimos en la base de datos.



## DEPURACIÓN, PRUEBAS UNITARIAS Y REFACTORIZACIÓN

# Depuración

- El depurador permite observar el comportamiento del programa en tiempo de ejecución y encontrar errores lógicos.
- El depurador permite:
  - Control de ejecución: Iniciar, continuar, interrumpir o detener la ejecución. Avanzar paso a paso por la aplicación. Ejecutar un proceso hasta una ubicación especificada. Establecer el punto de ejecución.
  - Puntos de interrupción, puntos de seguimiento, pila de llamada, …
  - Inspección de variables, registros, memoria, …
  - Editar y continuar
- La depuración debe habilitarse en el entorno y en el web.config (<system.web><compilation debug="true">).



# Refactorización

- La refactorización es el proceso que consiste en mejorar el código una vez escrito cambiando su estructura interna sin modificar su comportamiento externo.
- C# proporciona los siguientes comandos de refactorización en el menú Refactorización:
  - Extraer método
  - Cambiar nombre
  - Encapsular campo
  - Extraer interfaz
  - Promocionar una variable local a parámetro
  - Quitar parámetros
  - Reordenar parámetros

# Unit testing

- Probar partes del sistema de manera individual asegurando que funcionan correctamente
- Provee un contrato escrito y estricto que una porción de código debe cumplir
- Como resultado podemos encontrar problemas de manera temprana y de manera instantánea



# Microsoft.VisualStudio.TestTools .UnitTesting

- Atributos que identifican las clases y métodos de prueba ([TestClass] y [TestMethod]).
- Atributos de inicialización y limpieza para ejecutar código antes o después de ejecutar las pruebas unitarias, a fin de asegurarse un estado inicial y final concretos ([TestInitialize] y [TestCleanup]).
- Clases Assert que se pueden utilizar para comprobar las condiciones en las pruebas unitarias.
- El atributo ExpectedException para comprobar si se inicia determinado tipo de excepción durante la ejecución de la prueba unitaria.
- La clase TestContext que almacena la información que se proporciona a las pruebas unitarias, como la conexión de datos para las pruebas controladas por datos y la información necesaria para ejecutar las pruebas unitarias para los servicios Web ASP.NET.

# Anatomía de las pruebas unitarias

- Clase decorada con [TestClass] (Sin el se omiten los métodos de prueba).
- Conjunto de métodos de prueba ([TestMethod()]), uno por prueba.
- Opcionalmente:
  - Método con el código a ejecutar antes de hacer cada prueba [TestInitialize ()].
  - Método con el código a ejecutar cuando cada prueba se haya ejecutado [TestCleanUp ()] .
- La estructura de un método de prueba siempre sigue una secuencia de tres pasos (Patrón AAA: Arrange, Act, Assert)
  - Arrange: Configurar los objetos que intervienen en la prueba.
  - Act: Ejecutar el método a probar.
  - Assert: Verificar el correcto comportamiento.
- Nomenclatura: Utilizar el sufijo "Test" (fichero, clase y métodos) y nombres significativos.



# Estructura de un método de prueba

- La estructura de una prueba unitaria siempre sigue una secuencia de tres pasos (Patrón AAA: Arrange, Act, Assert)
  - Arrange: Configurar los objetos que intervienen en la prueba.
  - Act: Ejecutar el método a probar.
  - Assert: Verificar el correcto comportamiento .
- Es necesario utilizar clases Assert:
  - Assert: cuenta con numerosos métodos para comprobar los resultados.
  - CollectionAssert :para comparar colecciones de objetos y para comprobar el estado de una o más colecciones.
  - StringAssert: para comparar cadenas (con métodos específicos Contains, Matches , StartsWith ,…).
- O lanzar excepciones AssertFailedException.

# Probando la lógica de presentación

- Aseguremos que todo lo que llega a la vista esta testeado
- Probamos lógica de navegación
- Probamos las validaciones
- La vista la podemos probar navegando el sitio manualmente o con alguna herramienta automatizada



# Simulación de objetos

- Las dependencias con sistemas externos afectan a la complejidad de la estrategia de pruebas, ya que es necesario contar con sustitutos de estos servicios externos durante el desarrollo. Ejemplos típicos de estas dependencias son Servicios Web, Sistemas de envío de correo, Fuentes de Datos o simplemente dispositivos hardware.
- Estos sustitutos, muchas veces son exactamente iguales que el servicio original, pero en otro entorno o son simuladores que exponen el mismo interfaz pero realmente no realizan las mismas tareas que el sistema real, o las realizan contra un entorno controlado.
- Para poder emplear la técnica de simulación de objetos se debe diseñar el código a probar de forma que sea posible trabajar con los objetos reales o con los objetos simulados:
  - IoC: Inversión de Control (Inversion Of Control)
  - DI: Inyección de Dependencias (Dependency Injection)
  - Objetos Mock

# MockObjects

- La forma de establecer los valores esperados y "memorizar" el valor con el que se ha llamado al simulador para posteriormente verificarlo se ha generalizado dando lugar a un marco de trabajo que permite definir objetos simulados sin necesidad de crear explícitamente el código que verifica cada uno de los valores.
- Los MockObjects son objetos que siempre realizan las mismas tareas:
  - Implementan un interfaz dado
  - Permiten establecer los valores esperados (tanto de entrada como de salida)
  - Permiten establecer el comportamiento (para lanzar excepciones en casos concretos)
  - Memorizan los valores con los que se llama a cada uno de sus miembros
  - Permiten verificar si los valores esperados coinciden con los recibidos



# Patrón Inversión de Control (IoC)

- La Inversión de Control es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así el reuso de los mismos.
- Técnicas de implementación:
  - Service Locator: es un componente (contenedor) que contiene referencias a los servicios y encapsula la lógica que los localiza dichos servicios.
  - Inyección de dependencias: las dependencias son expresadas en términos de interfaces en lugar de clases concretas y se resuelven dinámicamente en tiempo de ejecución.

# Unity

- El ciclo de vida de la inyección de dependencias:
  - registrar, resolver y eliminar
- El contenedor de Unity permite administrar el ciclo mediante el registro, resolución y eliminación de dependencia, facilitando el uso de la inyección de dependencia en las aplicaciones.
- Permite tres tipos de inyección
  - inyección del constructor (es la más frecuente)
  - inyección del setter de propiedad
  - inyección de la llamada a un método



# Registro

- Utilizando el contenedor de Unity, se puede registrar un conjunto de asignaciones que determinan qué tipo concreto requiere cuando un constructor (o propiedad o método) identifica el tipo que se inyectará por un tipo de interfaz o tipo de clase base.
- Para la dependencia de ITenantStore en la clase ManagementController:

```
public ManagementController(ITenantStore tenantStore) {
  this.tenantStore = tenantStore;
}
```

- El registro sería:

```
var container = new UnityContainer();
container.RegisterType<ITenantStore, TenantStore>();
```

- También es posible el registro de instancias donde el contenedor es responsable de mantener una referencia a una instancia única de un tipo.

```
StorageAccount account =   ApplicationConfiguration
        .GetStorageAccount("DataConnectionString");
container.RegisterInstance(account);
```

# Registro

- Unity permite cargar una colección de registros desde un archivo de configuración a un contenedor, por ejemplo, desde app.config o web.config:

```
<configuration>
  <configSections>
    <section name="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection,
Microsoft.Practices.Unity.Configuration" />
  </configSections>
  <unity xmlns="http://schemas.microsoft.com/practices/2010/unity">
    <namespace name="Tailspin.Web.Survey.Shared.Stores" />
    <containers>
      <container name="RealAppContext">
        <register type="ITenantStore" mapTo="TenantStore" />
      </container>
      <container name="FakeAppContext">
        <register type="ITenantStore" mapTo="TenantStoreMock" />
      </container>    <containers>
  </unity>
</configuration>
```

- El método de extensión LoadConfiguration se define en el espacio de nombres Microsoft.Practices.Unity.Configuration.

```
IUnityContainer container = new UnityContainer();
container.LoadConfiguration("RealAppContext");
```



# Resolver

- El uso del método RegisterType define la asignación entre el tipo de interfaz utilizado en la clase de cliente y el tipo concreto que desea usar en la aplicación.
- A partir de este momento no se instancia directamente la clase, a través del método Resolve del contenedor se solicita la intancia plenamente formada con las dependencias resueltas.

```
var controller = container
        .Resolve<ManagementController>();
```

# Desarrollo Guiado por Pruebas (TDD)

- El Desarrollo Guiado por Pruebas, es una técnica de programación (definida por KentBeck); consistente en desarrollar primero el código que pruebe una característica o funcionalidad deseada antes del código que implementa dicha funcionalidad.
- El objetivo a lograr es que no exista ninguna funcionalidad que no esté avalada por una prueba.
- Lo primero que hay que aprender de TDD son sus reglas básicas:
  - No añadir código sin escribir antes una prueba que falle
  - Eliminar el Código Duplicado empleando Refactorización



# Ritmo TDD

- TDD invita a seguir una serie de tareas ordenadas, que a menudo se denomina ritmo TDD, y que se basa en los siguientes pasos:
  1. Escribir una prueba que demuestre la necesidad de escribir código.
  2. Escribir el mínimo código para que el código de pruebas compile
  3. Implementar exclusivamente la funcionalidad demandada por las pruebas
  4. Mejorar el código (Refactoring) sin añadir funcionalidad
  5. Volver al primer paso
- Este ritmo permite formalizar las tareas que se han de realizar para conseguir un código fácil de mantener, bien diseñado y que se puede probar automáticamente.

# Beneficios

- Reducen el número de errores y bugs ya que éstos, aplicando TDD, se detectan antes incluso de crearlos.
- Facilidad para entender el código, eligiendo una buena nomenclatura sirven de documentación.
- Facilidad para mantener el código:
  - Protege ante cambios, los errores que surgen al aplicar un cambio se detectan (y corrigen) antes de subir ese cambio.
  - Protegen ante errores de regresión (rollbacks a versiones anteriores).
  - Dan confianza.
- Facilidad para desarrollar ciñéndose a los requisitos.
- Ayudan a encontrar inconsistencias en los requisitos
- Ayudan a especificar comportamientos
- Ayudan a refactorizar para mejorar la calidad delcódigo (Clean code!)
- A medio/largo plazo aumenta (y mucho) la productividad.



# Referencias

- Enterprise Library
  - http://msdn.microsoft.com/en-us/library/ff648951.aspx
- Unity Container:
  - http://msdn.microsoft.com/en-us/library/ff647202.aspx
- Moq
  - http://code.google.com/p/moq
- PEX (Stubs and Moles)
  - http://research.microsoft.com/en-us/projects/pex/

# DESPLIEGUE Y PUESTA EN PRODUCCIÓN DE APLICACIONES WEB



## Tareas de la publicación

- Copiar la aplicación Web
- Cambiar los valores de Web.config que deben ser diferentes en el entorno del destino.
- Propagar los datos o estructuras de datos en las bases de datos que se usan en la aplicación web.
- Configurar valores de IIS en el equipo de destino, como el grupo de aplicaciones, el método de autenticación, si se permite el examen de directorios y el control de errores.
- Instalar certificados de seguridad.
- Establecer valores en el Registro del equipo de destino.
- Instalar ensamblados de aplicación en la GAC en el equipo de destino.

# Metodología

- Pasos (requieren una checklist):
  - Preparación
  - Despliegue
    - Paquetes de implementación web
    - Publicación con un solo clic
  - Verificación
- Existe una extensión de IIS, denominada Web Deploy, que puede automatizar la mayor parte de las tareas de implementación:
  - Compilación y copia de los archivos de la aplicación.
  - Configuración de valores de IIS, como el grupo de aplicaciones, el método de autenticación, si se permite el examen de directorios y el control de errores.
  - Generación y ejecución scripts de bases de datos que se usan para propagar los cambios a los datos o estructuras de la base de datos.
  - Transformación de valores de la configuración como la configuración de depuración, o cadenas de conexión.



# Preparación del archivo Web.config

- Para adaptar el fichero de configuración al entorno de destino es necesario:
  - Modificar:
    - Cadenas de conexión.
    - Valores de autenticación, pertenencia …
    - Configuración de depuración.
    - Configuración de traza.
    - Errores personalizados.
  - Quitar información confidencial.

# Transformación del archivo Web.config

- Para los proyectos de aplicación web, Visual Studio proporciona herramientas que automatizan el proceso de cambio (transformación) de archivos Web.config cuando se implementan.
- Para cada entorno en el que desee realizar una implementación, se debe crear un archivo de transformación que especifique solo las diferencias en el archivo Web.config para ese entorno.
- Los nombres de archivos de transformación incluyen el nombre del entorno de destino (el nombre de la configuración de compilación) como un nodo adicional entre "Web" y "config". Por defecto, el archivo de transformación para la configuración de compilación:
  - Debug se denomina Web.Debug.Config
  - Release se denomina Web.Release.Config
- Se pueden crear configuraciones de compilación personalizadas con sus correspondientes archivos de transformación.



# Preparación de la base de datos

- Al implementar una aplicación web que usa una base de datos de SQL Server, es posible que se tenga que propagar igualmente estructuras de datos, datos o ambas cosas.
- Para ello, Visual Studio puede crear automáticamente scripts (archivos .sql) desde la base de datos de origen para la base de datos de destino y estos scripts se pueden incluir en el paquete web.
- Para la actualización de la base de datos, también puede incluir scripts de SQL Server personalizados y especificar el orden en el que se deben ejecutar los scripts.
- Web Deploy ejecuta los scripts en el servidor de destino cuando se instala el paquete.

# Preparación Web Deploy

- Especificar opciones de implementación de la aplicación web (pestaña Empaquetar/publicar web de la página Propiedades del proyecto).
  - Si de desea propagar la configuración de IIS del proyecto al entorno del destino, hay que marcar la casilla Incluir la misma configuración de IIS existente en el Administrador de IIS.
- Especificar opciones de implementación de base de datos (pestaña Empaquetar/publicar SQL de la página Propiedades del proyecto).
- Especificar transformaciones del archivo Web.config (archivos Web.Debug.Config, Web.Release.Config, …).

Web Deploy debe estar instalado en el equipo de desarrollo (se instala en el equipo al instalar Visual Studio) y la misma versión de Web Deploy debe estar instalada en el servidor web de destino.



# Publicación con un solo clic

- Para poder publicar la aplicación web, debe crear un perfil de publicación que especifique cómo se publica. Se pueden crear tantos perfiles como sean necesarios para los diferentes escenarios (producción, preproducción, …).
- Los posibles métodos de publicación son: Web Deploy, FTP, Sistema de archivos o FPSE (Extensiones de servidor de Front Page).
- Salvo Web Deploy, el resto de los métodos de publicación solo copian archivos. No propagan configuraciones de IIS ni bases de datos, ni realizan otras tareas que podrían ser necesarias para implementar una aplicación web.
- La publicación con un solo clic está diseñada para simplificar la publicación repetitiva.