

# **Desarrollador de Aplicaciones Web**

## **Programación Web III**

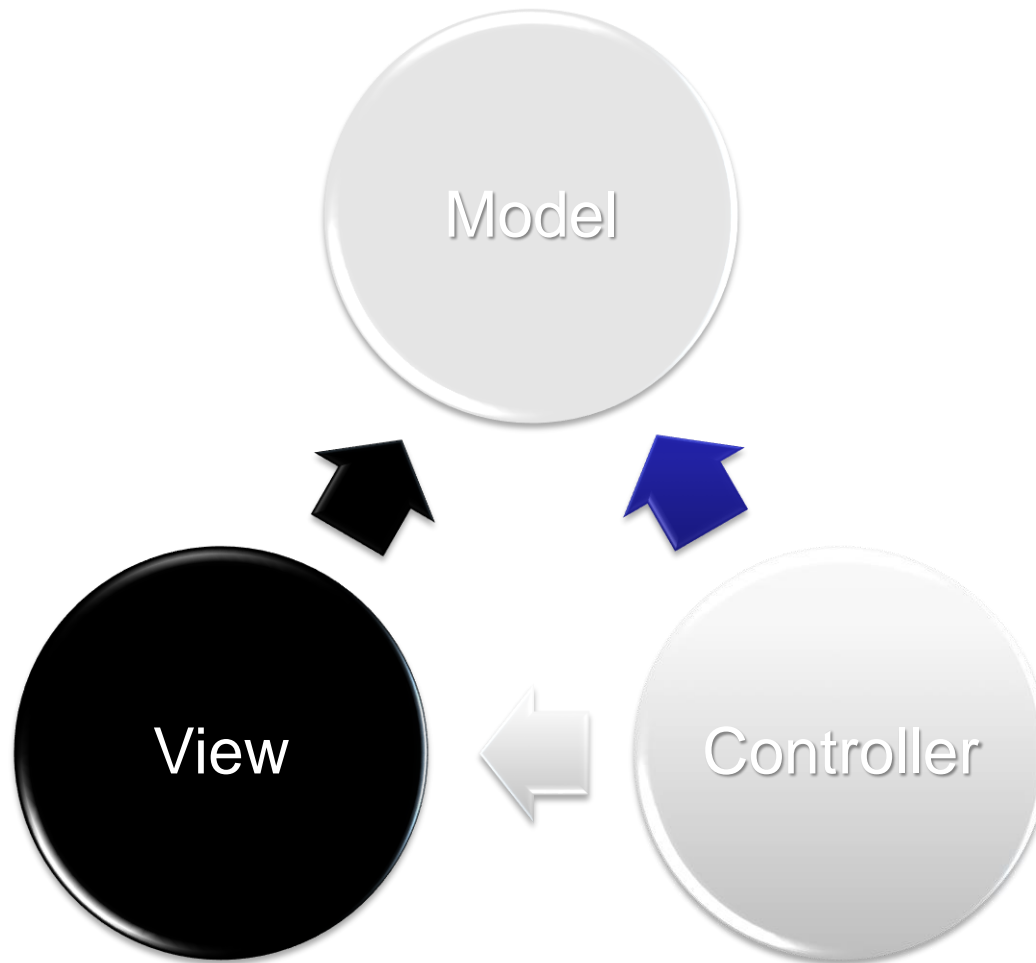


**Departamento de Ingeniería e Investigaciones Tecnológicas**

## **Razor & Helpers**

**Ing. Mariano Juiz**  
**Ing. Matias Paz Wasiuchnik**  
**Ing. Pablo Nicolás Sanchez**

# Modelo Vista Controlador



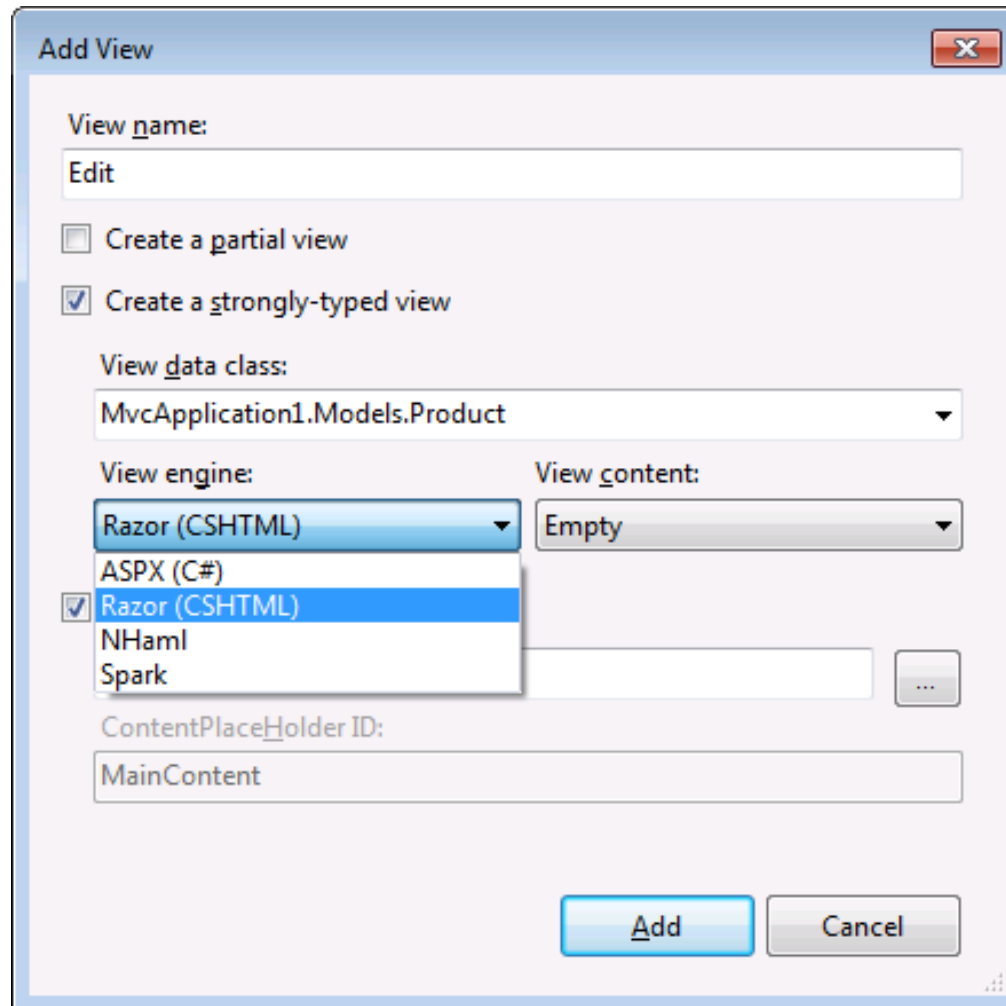
# View Engines

- ASP.NET MVC posee el concepto de “View Engines” desde sus inicios
- View Engines:
  - Templates Formularios Web “.aspx/.ascx/.master”
  - Spark (<http://sparkviewengine.com/>)
  - Nhaml (<http://code.google.com/p/nhaml/>)
  - Bellevue (<http://www.ope.ag/Bellevue/Page/intro>)
  - Razor
  - Otros...

Comparación: <https://stackoverflow.com/questions/1451319/asp-net-mvc-view-engine-comparison>

# View Engines

- View Engines:



The screenshot shows the 'Add View' dialog box with the following configuration:

- View name:** Edit
- ☐ Create a partial view
- ☒ Create a strongly-typed view
- View data class:** MvcApplication1.Models.Product
- View engine:** A dropdown menu is open, showing options: ASPX (C#), **Razor (CSHTML)** (selected), NHaml, and Spark. There is a checkmark next to 'Razor (CSHTML)'.
- View content:** Empty
- ContentPlaceHolder ID:** MainContent
- Buttons:** Add, Cancel

# Razor

- Surge en ASP.NET MVC 3. En MVC 4 aparece la 2da versión
- Se utiliza como motor de programación en las vistas o plantillas de nuestros controladores.
- Para C#, las páginas tiene extensión CSHTML
- Las clases de las vistas derivan de `System.Web.Mvc.ViewPage`
- `ViewPage` deriva de [System.Web.UI.Page](#) (tiene muchas propiedades y métodos que sólo sirven para Formularios Web)

# Razor

**Compacto, expresivo, y fluído:** No es necesario indicar de forma explícita los bloques de servidor dentro de su HTML. El analizador lo deduce del código

**Fácil de aprender:** Solo es necesario utilizar un lenguaje de programación (ej: C#) y conocimientos en HTML.

**No es un nuevo lenguaje:** Se utilizan los existentes (C# / VB.NET) y código HTML

**Compatible con Intellisense:** No se requiere una herramienta específica. Se puede utilizar cualquier editor de texto

**Compatible con Test Unitarios:** Se puede usar perfectamente Razor en entornos de pruebas o test unitarios.

# Sintaxis

## Comparación

### Formularios Web

6 transitions

```
<ul>  
  <% for (int i = 0; i < 10; i++) {%>  
    <li><%=i %></li>  
  <%} %>  
</ul>
```

### Razor

2 transitions

```
<ul>  
  @for (int i = 0; i < 10; i++)  
  {  
    <li>@i</li>  
  }  
</ul>
```

# Razor

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



# Sintaxis

## Carácter “@”

- Bloques con 1 sola sentencia

*@ {var mensaje = “Hola Mundo”;}*

- Expresiones “Inline”

*<span>La hora es: @DateTime.Now</span>*

- Bloques con varias sentencias

*@ {*

*var mensaje = “Hola Mundo”;*

*var diaDeLaSemana = DateTime.Now.DayOfWeek;*

*var mensaje = mensaje + “ Hoy es “ + diaDeLaSemana;*

*}*

- Todos las variables que mostremos con @ son parseadas con HTML Encode

# Sintaxis

## Carácter “@”

- Texto

*@: Hola Mundo*

- ¿Cómo agrego un “@” literal?

*unemail@@gmail.com*

- Todos las variables que mostremos con @ son parseadas con HTML Encode

# Sintáxis

- Comentarios

@{

@\* *Título de la página* \*@

// *Titulo de la página*

/\* *Titulo de la página* \*/

....

}

# Sintaxis

## Carácter “@”

### Ejemplo

```
@{  
    var nombre = "Juan Perez";  
    var rol="Desarrollador ASP.NET MVC";  
    <div>  
        Nombre: @nombre  
        Rol: @rol  
    </div>  
}
```

# Vistas

Razor procesa Vistas

Una vista es un archivo de texto que Razor reconoce que tiene trabajar en el

La extensión indica en que lenguaje Razor tiene que procesar el código que encuentra

Un Layout es una vista

Una Vista Parcial también es una Vista

# Sintaxis

## Layout

```
<html>
  <head>
    <title>Title</title>
  </head>
  <body>
    @RenderSection("Menu")
    @RenderBody()
  </body>
</html>
```

## Vista

```
@{
  Layout="~/Views/Shared/_Layout.cshtml";
}
@section Menu {
  <ul id="pageMenu">
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
}
```

# Sintaxis

Add View

View name:  
Edit

☐ Create a partial view  
☒ Create a strongly-typed view

View data class:  
MvcApplication1.Models.Product

View engine:  
Razor (CSHTML)  
ASPX (C#)  
☒ Razor (CSHTML)  
NHaml  
Spark

View content:  
Empty

ContentPlaceHolder ID:  
MainContent

Add Cancel

```
protected void Application_Start()  
{  
    ViewEngines.Engines.Add(new SparkViewFactory());  
    ...  
}
```

## Retornar el Modelo desde un Controlador

- Al momento de resolver la vista se debe retornar una instancia de **ViewResult** para poder enviar el contenido a la vista
- Se puede devolver por defecto una Vista (mediante el método Helper “View()”)
- Se envían los datos de un Modelo a la vista (mediante el método Helper “View(modelo)”)



## Retornar el Modelo desde un Controlador

- En el Controlador debe existir un ActionResult que devuelve el modelo a la vista

```
public ActionResult Index()
{
    var libro = new Libro {Isbn = "1122", Titulo = "El principito", TipoLibro = "Novela"};
    return View(libro);
}
```

- En la vista tenemos que indicar que tipo de datos vamos a recibir usando la sintáxis

@model "namespace.Clase"

@model EjemploMVC.Models.Libro

# Vistas

- Para mostrar el valor tenemos que usar la propiedad  
*El título del libro es <strong>@Model.Titulo</strong>*



# Vistas

- Si quiero mostrar un listado...

```
public ActionResult Index()  
{  
    var libros = new List<Libro>  
    {  
        new Libro {Isbn = "1122", Titulo = "El principito", TipoLibro = "Novela"},  
        new Libro {Isbn = "1122", Titulo = "Steve Jobs", TipoLibro = "Biografía"}  
    };  
    return View(libros);  
}
```

- En la vista tenemos que indicar que tipo de datos vamos a recibir usando la sintáxis

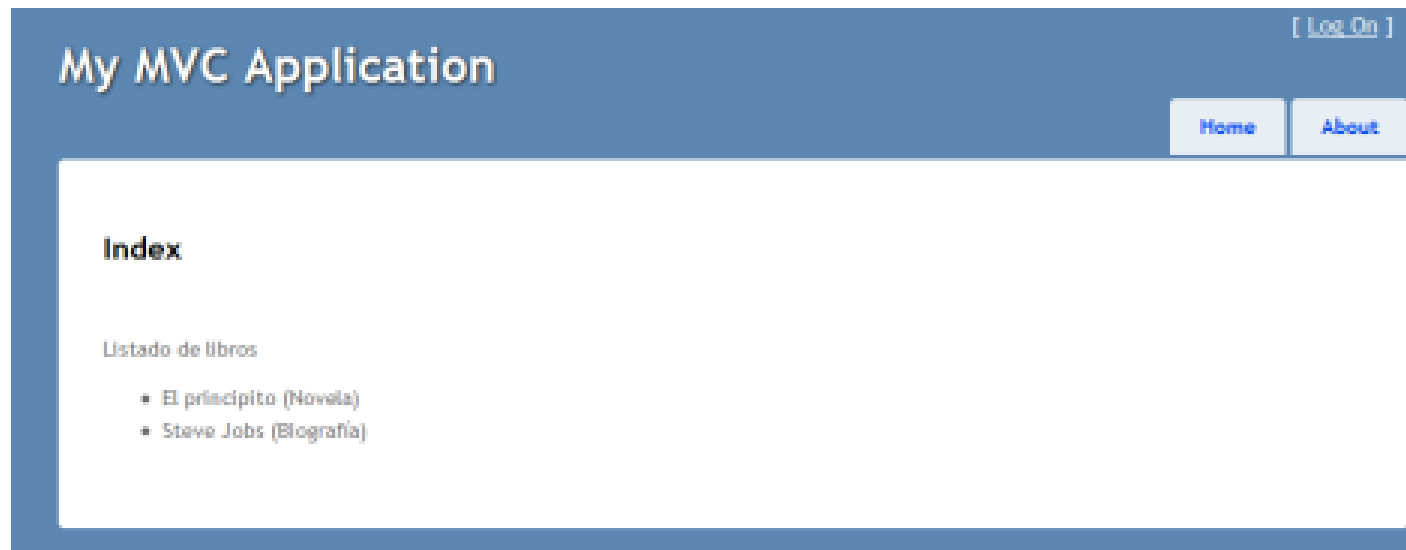
**@model IEnumerable** <"namespace.Clase">

*@model IEnumerable<EjemploMVC.Models.Libro>*

# Vistas

- Recorreremos los datos con “foreach”

```
<ul>
  @foreach(var libro in Model)
  {
    <li> @libro.Titulo ( @libro.TipoLibro)</li>
  }
</ul>
```



# Helpers

Los helpers son una herramienta muy potente para generar nuestro propio código HTML dentro de las vistas.

Podemos crear formularios (hay que tener cuidado con los literales de cadena) y campos dentro de el.

Se sugiere utilizar Helpers para todos los elementos involucrados, para evitar que Razor se “confunda”

Los helpers nos van a permitir crear campos en un formulario de una manera muy cómoda.

# Helpers

Hay un helper por cada control HTML que habitualmente usamos en formularios, como por ejemplo `Html.TextAreaFor`, `Html.LabelFor`, `Html.DropDownListFor`, etc.

Para crear un formulario usamos:

```
@Html.BeginForm([Action], [Controller]) {  
    <!-- Codigo del formulario -->  
}
```

**@Html.TextBoxFor** que muestra un campo de texto para la propiedad indicada

**@Html.CheckBoxFor** que muestra una casilla de verificación para la propiedad indicada

¿Cómo se indica a cada helper que propiedad debe renderizar el control?

**@Html.TextBoxFor(x=>x.Nombre)**

El parámetro en forma `x=>x.Nombre` es una **expresión lambda**. Son evaluadas en tiempo de compilación y no de ejecución

Esto requiere que la vista sea tipada con la declaración del `@Model` (como vimos anteriormente)

La ventaja es el manejo de errores

# Helpers

También podemos agregar Helpers creados en el proyecto (u otros)

## Creación

```
namespace MiHelpers.Helpers
```

```
{
```

```
    public static class CustomHelpers
```

```
    {
```

```
        public static MvcHtmlString EliminarHTML(this HtmlHelper html,
```

```
string input)
```

```
        {
```

```
            return new
```

```
MvcHtmlString(Regex.Replace(input, "<.*?>", string.Empty));
```

```
        }
```

```
    }
```

```
}
```

## Utilización

```
@using MyCustomHelper.Helpers
```

```
@Html.EliminarHTML("<div><p>Test</p></div>")
```

# **Desarrollador de Aplicaciones Web**

## **Programación Web III**



**Departamento de Ingeniería e Investigaciones Tecnológicas**

# **Muchas gracias**

**Ing. Mariano Juiz**  
**Ing. Matias Paz Wasiuchnik**  
**Ing. Pablo Nicolás Sanchez**