ASP.NET

Victor Herrero Cazurro

Contenidos

1. Introduccion

ASP.NET, es una plataforma para la creación de aplicaciones Web con .NET framework, a partir de la última versión, se ha adoptado el nombre de One ASP.NET, para agrupar todas las tecnologias web .NET, así pues no hay plantillas para crear proyectos con MVC o con Web APIs, sino que se ha modularizado y se pueden añadir sobre el mismo proyecto.

1.1. Visual Studio

La nueva version de la herramienta para los desarrollos de .NET, es ya la 2017.

Herramienta para el desarrollo de aplicaciones con **.Net Framework**, **.Net Standard** y **.Net Core**, que en su version comunity dispone de dos formas de instalación,

- Con un instalador online que se puede descargar de aquí.
- Para instalaciones sin Internet, aquí hay mas información.

1.2. ASP.NET WebForms

Las aplicacion ASP.NET WebForms permiten crear aplicaciones web basadas en paginas dinamicas.

Para la creacion de las paginas, la herramienta Visual Studio proporciona un potente editor visual que permite formar las paginas arrastrando y soltando una gran cantidad de componentes visuales disponibles.

El framework, proporciona una arquitectura basada en el patón observer, eventos que se generan en el cliente, que tienen su procesamiento en el servidor.

1.3. Tecnologías Web

Los desarrollos Web actuales implican no solo a una tecnologia de servidor como puede ser ASP.NET, java o PHP, sino tambien a una tecnologia de cliente como es javascript.

Esta tecnologia esta imponiendo un cambio en el paradigma del desarrollo, dado que parte de la logica de la aplicación, ya no es necesario que se ejecute en el servidor, sino que se puede ejecutar en el propio browser con su motor javascript.

Esto hace que las aplicaciones se hayan transformado en la parte servidora, dado que ahora no se preocupan de la Vista, sino unicamente de proporcionar datos, esto ha llevado a la adapción de los servicios REST como el paradigma para moldear la parte servidora (backend) y javascript para la parte cliente (frontend).

Una vez establecidas las responsibilidades, falta definir como se realizará la comunicación, y para ello se ha adptado AJAX, que es parte de javascript que permite realizar peticiones desde el browser al servidor, sin necesidad de realizar un cambio de pagina, se dice que las peticiones las realiza en background, una de las caracteristicas de AJAX, es que las peticiones las realiza de forma asincrona, es decir, se sabe cuando se lanzan, pero no se sabe cuando llega la respuesta. Otro punto importante en esta arquitectura es el formato de intercambio de datos entre cliente (browser) y sevidor (servicio REST), el formato mas estandar seria JSON.

1.3.1. JavaScript

Es un lenguaje de programación interpretado que parte del estandar ECMAScript, teniendo como principales caracteristicas

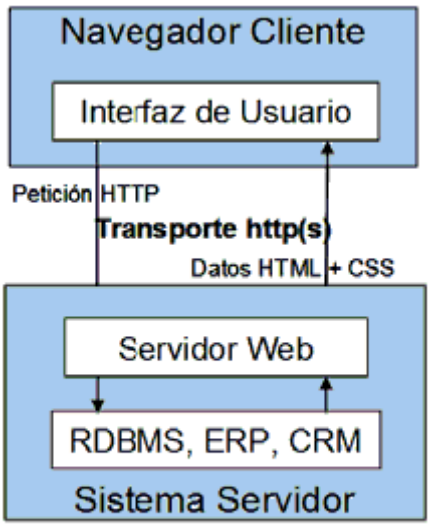
- Orientado a objetos basado en prototipos
- Débilmente tipado
- Dinámico.

Se incorpora en todos los navegadores web (browser) un motor capaz de interpretar javascript, permitiendo con el codigo interpertado modificar el comportamiento de los HTML estaticos descargados desde un servidor, dandoles dinamismo.

1.3.2. Ajax

En aplicaciones web tradicionales, las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenan llamadas al servidor, una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

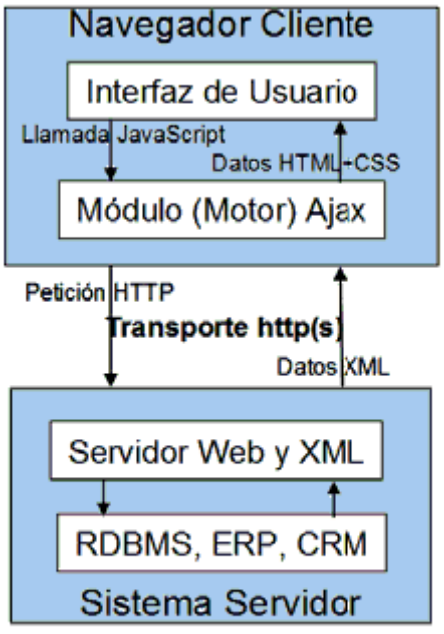
Esta técnica funciona correctamente, pero no crea una buena sensación al usuario, dado que al pedir paginas completas, el usuario debe esperar a que se cargue la página entera, cuando en la mayoria de los casos lo que necesita es un pequeño cambio en la misma pagina, en estos casos se pierde tiempo y ancho de banda en cargar información que ya se estaba visualizando antes de la petición.



Modelo clásico de aplicación web

La tecnologia AJAX permite mejorar la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano, esto proporciona dos ventajas:

- El usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor, efecto parpadeo.
- El servidor retorna solo información, no un información y como representarla.



Modelo Ajax de aplicación web

1.3.3. REST

Define una arquitectura de servicios orientada a recursos, esto es, permite realizar las tareas de CRUD sobre información.

Se basa en el protocolo de transporte HTTP, expriminedo todas sus caracteristicas Method, Headers, Status Code, Body, Url, …¬

Se emplea **Method** para indicar la acción a realizar, siendo estas las del CRUD

- Peticiones HTTP de GET, en el servicio se traducen en consultas (SELECT)
- Peticiones HTTP de POST, en el servicio se traducen en inserciones (INSERT)
- Peticiones HTTP de PUT, en el servicio se traducen en modificaciones (UPDATE)
- Peticiones HTTP de DELETE, en el servicio se traducen en borrado (DELETE)

Se emplea el **Body** para enviar información (el recurso) al servicio.

Se emplean las **Headers** como Content-Type y Accept para indicar el formato de los datos a intercambiar, normalmente JSON.

Ejemplo de Json

```
{
    //Identificador
    "id": "lisa",
    //Atributos
    "desc": "Lisa Simpson",
    "href": "http://simpsons/personajes/lisa",
    "name": "Lisa",
    "surname": "Simpson",
    //Relaciones
    "parent": { "id": "homer", "desc": "Homer Simpson" "href":
"/personajes/homer" },
    "mother": { "id": "marge", "desc": "Marge Simpson", "href" :
"/personajes/marge"},
    "brothers": [{ "/personajes/lisa/brothers" }]
}
```

Se emplea la URL para indicar: Tipo de información, Identificador, Relaciones, filtros…¬

Algunos ejemplos de peticiones

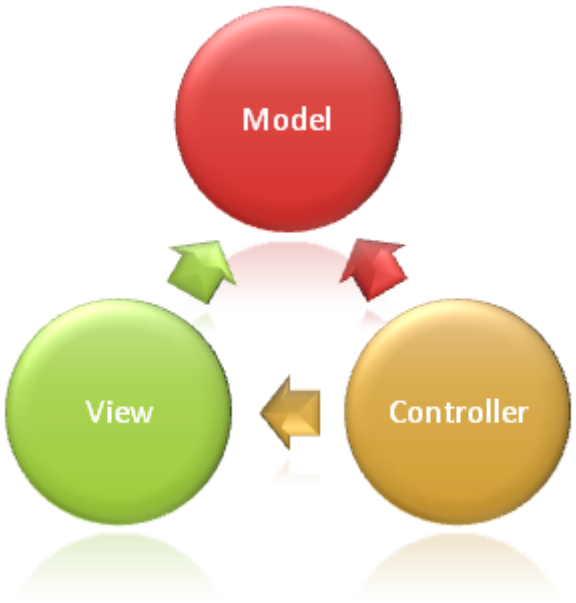
```
// Peticion que retorna el persona de Lisa Simpson,
// donde *personajes* es el tipo de dato a consultar (MODELO)
// y *lisa* es el identificador del dato.

http://simpsons/personajes/lisa (GET)

//Peticion que retorna los personajes de Bart y Maggie Simpson,
// donde *brothers* es una relación entre personajes.

http://simpsons/personajes/lisa/brothers (GET)

// Peticion que retorna los personajes Ned, Maude, Rod y Todd
http://simpsons/personajes?surname=Flanders
```

1.4. El Patrón MVC

Patrón de la capa de presentación, que identifica tres componentes de forma clara

- **Vista**: Es el encargado de representar la información, en el solo debe residir logica necesaria para la representación de la información. Por ejemplo un algortimo que permite recorrer un array de objetos personaje, para representarlo en una tabla. En cambio un ejemplo de lo que no debemos encontrar en la vista, seria la logica necesaria para obetener el array de objetos personaje.
- **Controlador**: Es el encargado de hacer accesible al usuario las acciones a la que es capaz de atender la aplicación, en una aplicación WEB, cada URL que publique la aplicación sera procesada por un Controlador. Además tiene como misiones:
  - ¬ Recopilar la información que envia el cliente.
  - ¬ Transformar dicha información en objetos entendibles por el resto de la aplicación, en aplicaciones WEB el intercambio de información entre cliente y servidor se realiza siempre con cadenas de caracteres, este paso puede incluir validaciones de datos.
  - ¬ Invocar la logica de negocio que realice la tarea que ofrece el controlador.
  - ¬ Recopilar la respuesta del la logica de negocio.
  - ¬ Elegir que Vista se va a emplear para representar la respuesta al cliente.
  - ¬ Trasladar la información necesaria a la vista y lanzarla.

- **Modelo**: Sería todo lo demas, lo que no es Controlador ni Vista, en general podiamos decir que es la lógica de negocio y los datos, en ocasiones se emplean Modelos anémicos, donde el Modelo únicamente representa información y no tiene ninguna lógica de negocio, en estos casos podemos distinguir dentro del Modelo, componentes de Logica de negocio y componentes Entidades.



1.5. Gestión de Dependencias

Para la gestion de dependencias que puedan presentar los proyectos con otras librerias, .NET ofrece compatibilidad con la herramienta NUGET.

1.6. Arquitectura MVC en ASP.NET

ASP.NET MVC es un framework con licencia open-source para crear aplicaciones WEB que apliquen el patrón MVC (Model- View-Controller) sobre el framework ASP.NET.

En los comienzos ASP.NET, era un framework que permitia la creación de aplicaciones WEB empleando WEB FORMS, un framework que poseia componentes visuales que ve y opera el cliente, que enlazan con manejadores de eventos que se ejecutan en el servidor.

Mas adelante aparece ASP.NET MVC, que aplica el patrón MVC a las aplicaciones Web con .NET, ahora ya está en su versión 6, aunque para desarrollos con Visual Basic, no se ha dado soporte para esta última versión, siendo la última soportada la

5.

Algunas de las caracteristicas que se han ido incluyendo a lo largo de las versiones son:

- UI Helpers
- Modelo de validaciones basado en atributos en cliente y servidor.
- Motor Razor View.
- Web API

Como se ve una de las caracteristicas es Web API, y es que ASP.NET MVC, aunque es un framework pensado para la creación de aplicaciones web que admitan peticiones HTTP y retornen HTML, tambien permite realizar de forma sencilla una capa de servicios que retornen XML o JSON.

Las novedades de MVC son:

- One ASP.NET: Un unico tipo de proyecto para todas las tecnologias ASP:NET.
- ASP.NET Identity: Nuevo framework de auteticación mas configurable
- Plantillas de Bootstrap 3: Uso de Bootstrap para el diseño de las paginas HTML.
- Attribute Routing: Nueva forma de definir las rutas directamente en los controladores
- ASP.NET scaffolding: Framework que crea el codigo necesario para realizar un CRUD WEB sobre el modelo de datos.
- Authentication filters: Nuevo filtro que permite personalizar la logica de autenticación.
- Filter overrides: Posibilidad de configurar la cadena de filtros a ejecutar previos a un Controlador.

A partir de Visual Studio 2012, NuGet es el encargado de resolver las dependecias de los proyectos, por lo que para las dependecias de proyectos MVC, tambien se empleará, de echo las plantillas proporcionadas por Visual Studio lo hacen.

1.6.1. Controladores, Modelos y Vistas

Los proyectos MVC, emplean el paradigma "Convention over Configuration", es decir establecen una pauta el el nombrado de los componentes y su situación, para evitar tener que configurar el componente, en este caso, se definen tres carpetas

- Controllers
- Models
- Views

Que estan destinadas a contener cada una de ellas los correspondientes componentes Controladores, Modelos y Vistas.

A mayores se añaden otras carpetas ya fuera del patrón MVC, como son:

- Scripts: para situar lo ficheros js.
- fonts: para las fuentes de texto web de bootstrap.
- Content: para los ficheros css, imagenes y otro contenido estatico.
- App_data: para ficheros de lectura/escritura desde la aplicación.
- App_start: para ficheros de configuracion como las rutas, filtros, …

Las convenciones no se quedan aqui, los nombres de las clases de los controladores, deberán llevar el sufijo Controller, además cuando un controlador referencie a una vista, esta por defecto se buscará en la siguiente ruta

```
/Views/<nombre de la clase del controlador sin el sufijo
Controller>/<nombre del m
étodo del controlador>
```

1.6.2. Enrutado

ASP.NET permite la definición de las rutas para acceder a los recursos de forma independiente a los mismos, esto se define en el fichero RouteConfig, en el método RegiterRoutes, el cual es invocado desde Global.asax.

En este método se definirán patrones de rutas genéricos, que permitirán conducir a la petición hasta la lógica deseada.

La configuración por defecto de las rutas es la siguiente

```
Public Module RouteConfig
    Public Sub RegisterRoutes(ByVal routes As RouteCollection)

        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

        routes.MapRoute(
            name:="Default",
            url:="{controller}/{action}/{id}",
            defaults:=New With {.controller = "Home", .action =
"Index", .id = UrlParameter.Optional}
        )
    End Sub
End Module
```

Donde se interpreta parte de la URL para localizar la clase de controlador y el método que se ha ejecutar ante una petición.

1.6.3. Acciones

Las acciones o métodos de acción, son los que albergan la lógica accesible por el cliente mediante la invocación de una URL, son los que contienen la logica de control.

1.6.4. El motor Razor View

Es el motor que permite definir vistas dinamicas de forma simplificada, ya que permite incustrar en paginas HTML contenido dinámico con una sintaxis limpia.

1.7. Estructura de proyecto

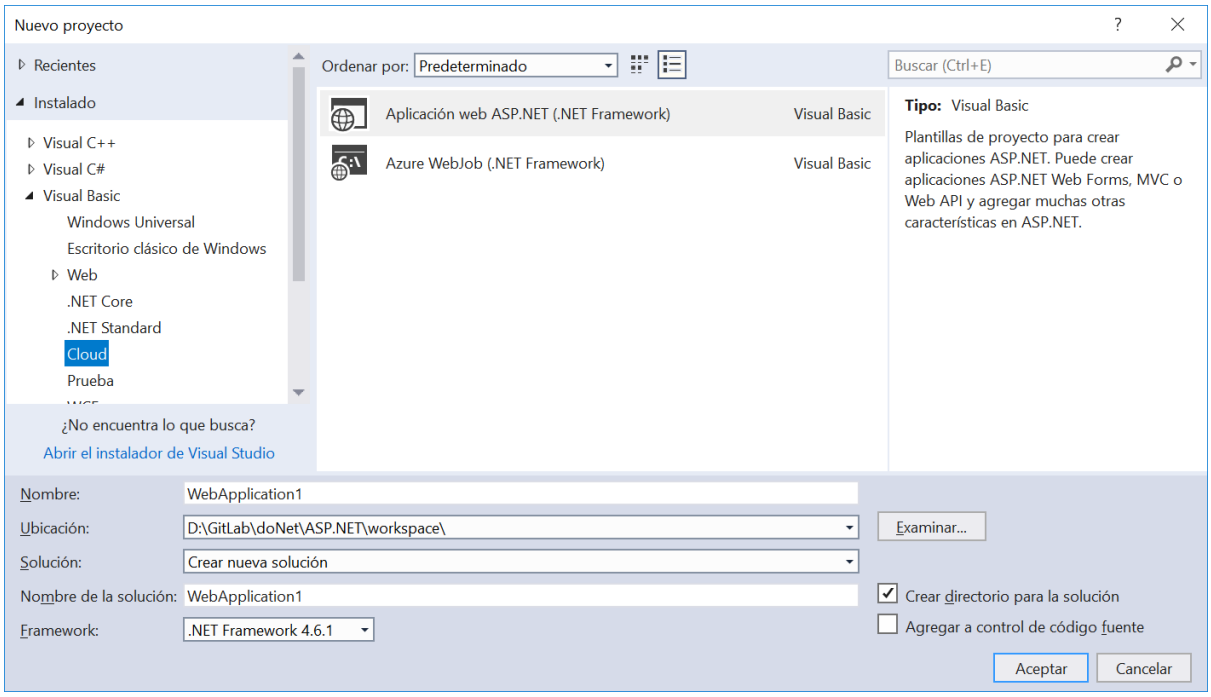
En la estructura de proyecto cabe destacar lo siguiente

- Se crea una carpeta para contener los **Controladores** (Controllers), otra para las **Vistas** (Views) y otra para el **Modelo** (Models).
- Se crea una carpeta para contener los **CSS**, imagenes y cualquier fichero estatico (Content) y otra para los **JS** (Scripts)
- Se crea una carpeta **App_Data** para contener los datos de la aplicacion.
- Se crea una carpeta **App_Start** para contener clases de configuracion de enrutado, filtros y JS.

- Se crea un fichero **Global.asax** para definir el enrutado, los filtros y la referencia a los JS de la aplicacion, que delegan en las clases definidas en **App_Start**.
- Se crea un fichero **Web.config** para la configuracion de la aplicacion.
- Se crea un fichero **packages.config** que describe las referencias a librerias del proyecto.

1.8. Primer proyecto VB

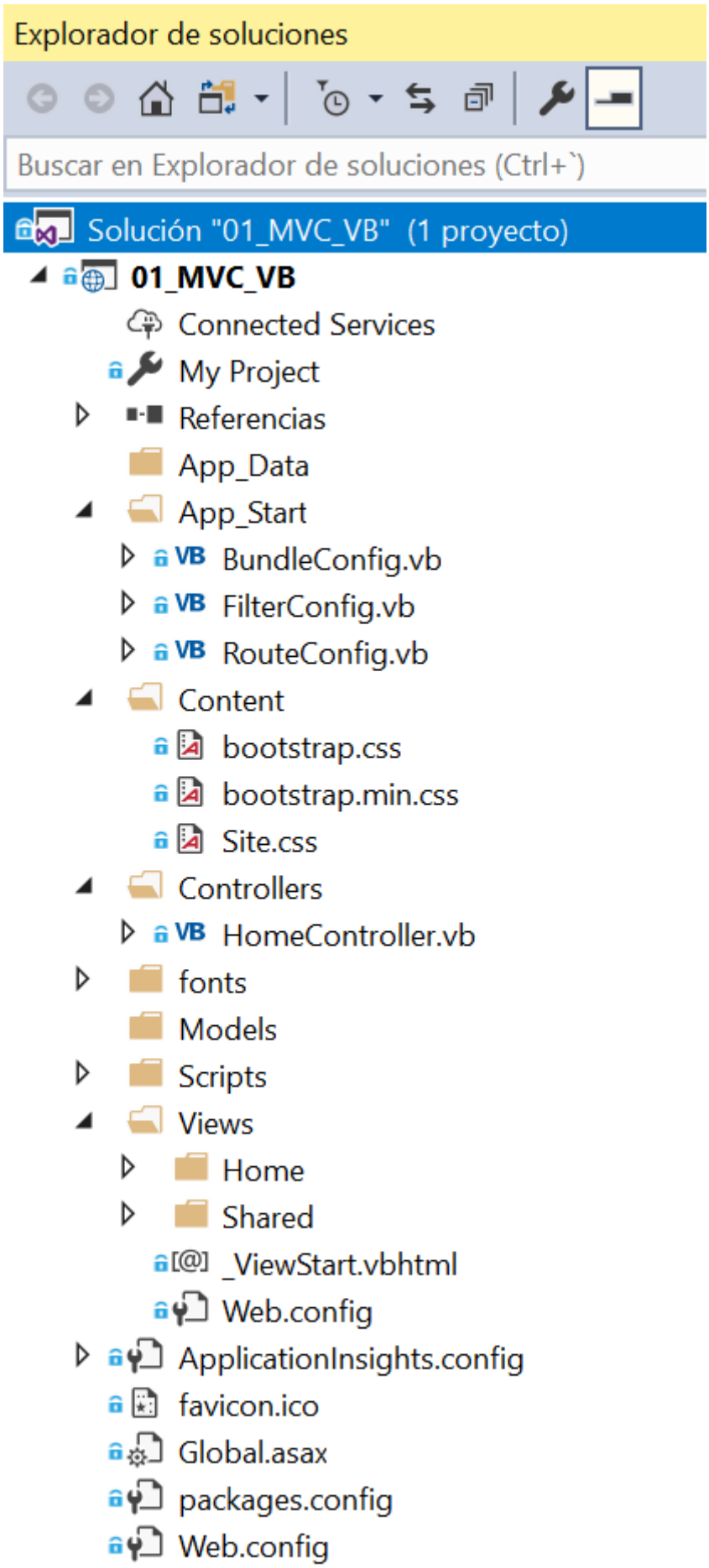
Para crear un proyecto MVC 5 con VB.NET, se ha de abrir el menu **Nuevo - Proyecto** de VS2017.



Lo primero que se ha de seleccionar es el lenguaje a emplear para codificar el proyecto, en este caso **Visual Basic**.

Y posteriormente el tipo de proyecto, en este caso, seleccionaremos la seccion **Cloud** y dentro **Aplicacion web ASP.NET (.NET Framework)**.

Por defecto se genera un proyecto con la estructura basica MVC, en la que se dispone de un **Controlador** y varias **Vistas**.

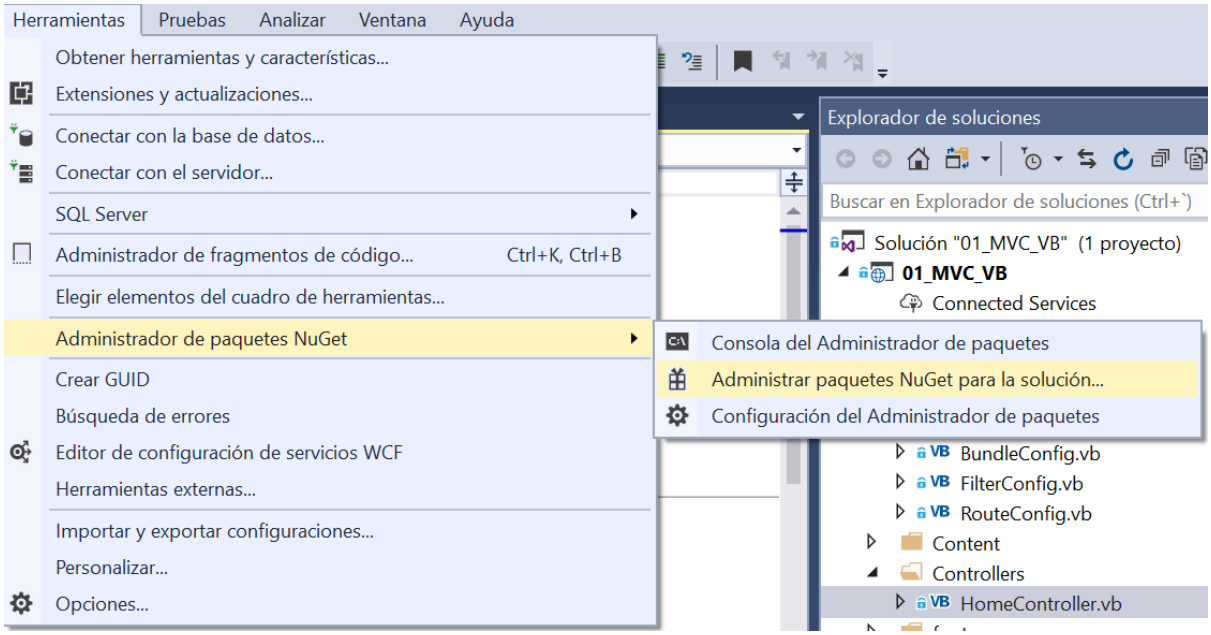

Explorador de soluciones
Buscar en Explorador de soluciones (Ctrl+`)
Solución "01_MVC_VB" (1 proyecto)
01_MVC_VB
Connected Services
My Project
Referencias
App_Data
App_Start
BundleConfig.vb
FilterConfig.vb
RouteConfig.vb
Content
bootstrap.css
bootstrap.min.css
Site.css
Controllers
HomeController.vb
fonts
Models
Scripts
Views
Home
Shared
_ViewStart.vbhtml
Web.config
ApplicationInsights.config
favicon.ico
Global.asax
packages.config
Web.config

2. Gestion de Dependencias

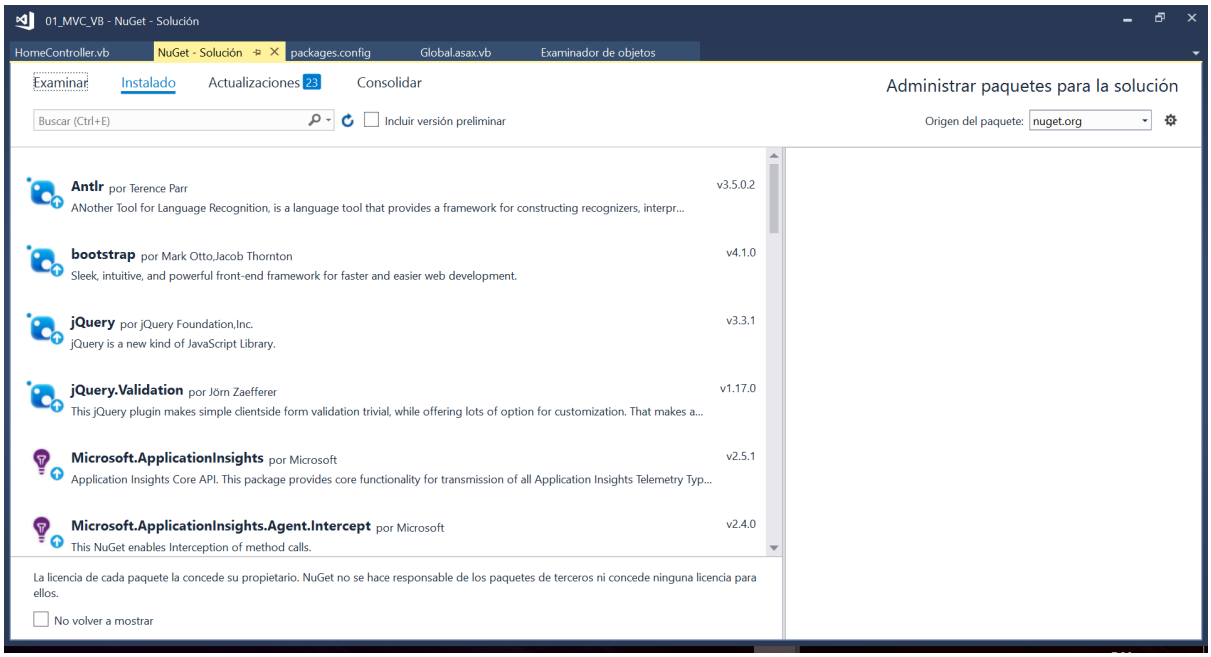
La gestión de dependencias se realiza con la herramienta llamada **Nuget**.

Nuget es un gestor de paquetes para proyectos Microsoft, es capaz de proporcionar librerias que enriquezcan los desarrollos.

Es un proyecto independiente, aunque en Visual Studio viene embebido un cliente para la descarga de paquetes, que se puede acceder desde el menu **Herramientas**



En dicho cliente, se pueden visualizar las dependencias actuales y buscar nuevas, asi como ver actualizaciones.



2.1. Crear dependencias

Para poder generar una nueva depedencia con un proyecto local, se ha de descargar la herramienta **Nuget.exe**, para ello se ha de acceder aquí

Se ha de incluir el comando en la variable de entorno PATH.

Se puede crear el fichero \.nuspec*, para ello se ha de ejecutar sobre el directorio que contiene el fichero \.vbproj* del proyecto .NET, el comando

```
> NuGet spec <fichero.csproj>
```

Y posteriormente rellenarlo con los datos correspondientes

Una vez definido el fichero descriptor, se a de crear el paquete NuGet con el comando

```
> NuGet pack <fichero.nuspec>
```

*Este puede ser un ejemplo de fichero *nuspec*

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns=
"http://schemas.microsoft.com/packaging/2013/05/nuspec.xsd">
    <metadata>
        <id>Simpson</id>
        <version>1.0.0</version>
        <title>Simpson Web App</title>
        <authors>Victor</authors>
        <owners>Victor</owners>
        <requireLicenseAcceptance>false</requireLicenseAcceptance>
        <description>Web App Simpsons</description>
        <copyright>Copyright © 2016</copyright>
        <dependencies>
            <dependency id="bootstrap" version="3.0.0" />
            <dependency id="Microsoft.ApplicationInsights.Web" version
="2.1.0" />
            <dependency id="Microsoft.AspNet.Mvc" version="5.2.3" />
            <dependency id="Microsoft.AspNet.Web.Optimization" version
="1.1.3" />
            <dependency id=
"Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version=
"1.0.0" />
            <dependency id="Microsoft.jQuery.Unobtrusive.Ajax" version
="3.2.3" />
            <dependency id="Microsoft.jQuery.Unobtrusive.Validation"
version="3.2.3" />
            <dependency id="Modernizr" version="2.6.2" />
            <dependency id="Respond" version="1.2.0" />
        </dependencies>
    </metadata>
</package>
```

En este fichero basicamente se indican campos relativos al proyecto que lo hacen unico, y las dependencias que tiene.

En el caso de proyectos Visual Studio, se puede ir más rapido generando el paquete de Nuget, saltandose la creación del fichero \.spec* que se autogenerará con el comando

```
> NuGet pack <fichero.csproj> -Build
```

2.2. Publicar dependencias

Para publicar un paquete en Nuget.org, es necesario registrarse o bien en Nuget o bien en Microsoft.

Los paquetes que se pueden publicar como ficheros \.nupkg*.

La publicación se hará dede la propia página de Nuget.org, desde la pestaña Upload Package

images::nuget_publicar_nuget.org.png[]

2.3. Control de Errores

Se puede incluir una pagina de descripcion detallada de errores para el desarrollo de las aplicaciones ASP.NET Core, siguiendo estos pasos:

- añadiendo el paquete NuGet **Microsoft.AspNetCore.Diagnostics**
- añadiendo el Midleware **DeveloperExceptionPage**

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
}
```

Esta funcionalidad deverá estar solo disponible en el entorno de **Development**.

Para el resto de entornos, se deberá generar un controlador para los errores, existiendo dos formas

- Asociandolo a un **Controlador** existente, tanto en el enrutado

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseExceptionHandler("/Home/Error");
}
```

Como en la implementacion

```
public class HomeController : Controller
{
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}
```

- O definiendo un **Controlador** particular

```
public class ErrorController : Controller
{
    [Route("/Error")]
    public IActionResult Index()
    {
        return View(new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}
```

3. Modelo

3.1. Patrón Dao

Patrón de diseño que permite aislar la capa de acceso a datos del resto de aplicación.

La idea de esta capa es que el resto de la aplicación, no debe verse afectada porque se cambe el motor de persistencia, así como no debe verse afectada por la sintaxis y los APIs necesarios para acceder a los datos, alogo como: "Lo que pasa en Las Vegas (La Persistencia), se queda en Las Vegas (La Persistencia)".

Generalmente este patrón hace que definan distintas clases para cada entidad, en la que se dispone de métodos abstractos, que operan con el almacen de datos.

3.2. Patrón Specification

Este Patrón de diseño, indica que las reglas de negocio pueden crearse y combinarse por el encadenamiento de otras reglas mas simples, usando lógica booleana.

La idea es que cada regla tendrá un método que compruebe unicamente dicha regla y al concatenarla se obtenga la aplicacion de una regla mas compleja.

3.3. Patrón Repositorio

Patrón de mas alto nivel de la capa de persistencia, que modela un cache de las tablas a traves de algun tipo de coleccion de datos, sobre la que se puede añadir, modificar, borrar y consultar registros directamente.

Las consultas se podrán resolver empleado el patrón **Specification**

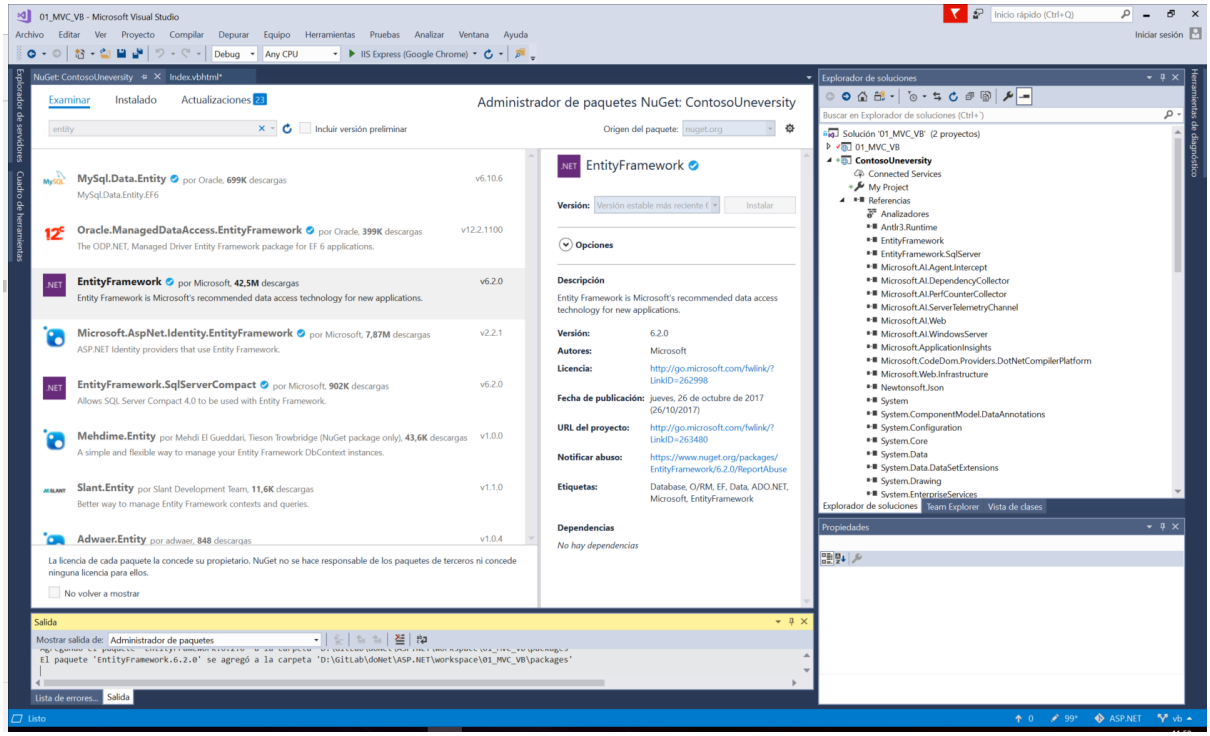
3.4. Object Relational Mapping (ORM)

Técnica de programación que permite realizar un mapeo (relacion) entre Objetos de un programa y registros de una Base de datos relacional.

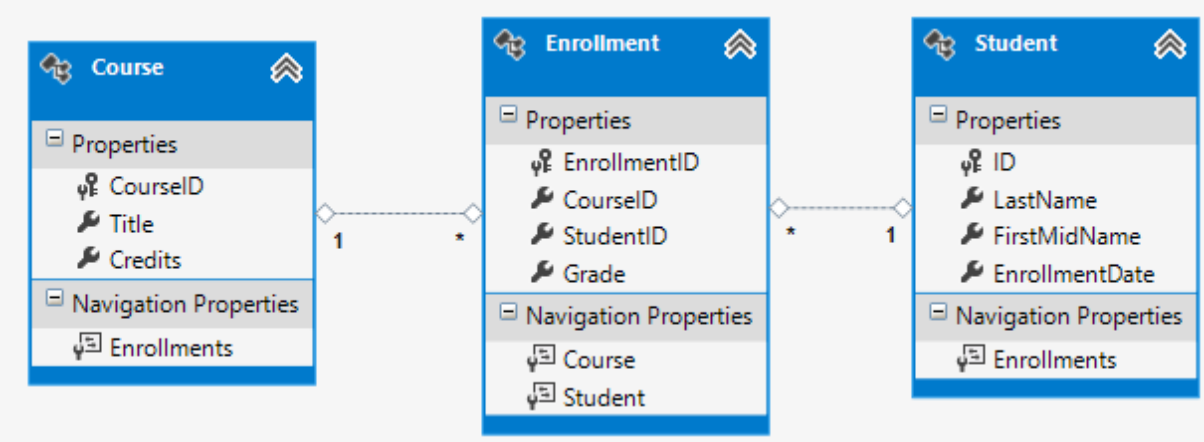
3.4.1. Entity Framework

Para poder emplear Entity Framework en un proyecto, lo primero será añadir la

referencia con Nuget, para ello acceder al menú **Proyecto - Administrar Paquetes Nuget**



A lo largo de este documento, se trabaja con el siguiente modelo de datos



La clase que coordina la funcionalidad de Entity Framework es una que extiende **System.Data.Entity.DbContext**.

```
Namespace ContosoUniversity.DAL
    Public Class SchoolContext
        Inherits DbContext

        Public Sub New()
            MyBase.New("SchoolContext")
        End Sub

        Public Property Students As DbSet(Of Student)
        Public Property Enrollments As DbSet(Of Enrollment)
        Public Property Courses As DbSet(Of Course)

        Protected Overrides Sub OnModelCreating(modelBuilder As
DbModelBuilder)
            modelBuilder.Conventions.Remove(Of
PluralizingTableNameConvention)()
        End Sub

    End Class
End Namespace
```

NOTE

Al invocar al contructor del padre indicando **SchoolContext**, se esta estableciendo el nombre de la cadena de conexion que se define en el ficher **Web.config**

De no especificarse, se supondra, que la cadena se llamará igual que la clase, en este caso **SchoolContext**

La tipologia **System.Data.Entity.DbSet** representa un conjunto de entidades, es decir por regla genral una tabla de la base de datos.

```
Public Property Students As DbSet(Of Student)
```

NOTE

Se podríam omitir tanto **Students As DbSet(Of Enrollment)**, como **Students As DbSet(Of Course)**, ya que Entity Framework las incluiría implícitamente dado que la entidad **Student** hace referencia a la entidad **Enrollment** y esta a la entidad **Course**.

NOTE

El método **OnModelCreating**, permite configurar el comportamiento de creacion del modelo en la base de datos a partir del modelo de clases, en este caso, se esta indicando que los nombres de las tablas se generen en singular aunque los DbSet sea plurales.

Se pueden establecer otros comportamientos.

Creacion del Modelo

Entity Framework puede automáticamente crear el Modelo de la base de datos partiendo del Modelo de Objetos Entidad de la aplicación. Para ello simplemente hay que definir las entidades y emplearlas en un contexto de Entity Framework.

En el siguiente ejemplo se puede ver un modelo de objetos Entidad

```
Public Class Enrollment
    Public Property EnrollmentID() As Integer
    Public Property Grade() As Grade
    'FK
    Public Property CourseID() As Integer
    Public Property StudentID() As Integer
    'Navigation Property
    Public Overridable Property Course() As Course
    Public Overridable Property Student() As Student
End Class
Public Enum Grade
    A
    B
    C
    D
    E
    F
End Enum
Public Class Course
    Public Property CourseID() As Integer
    Public Property Title() As String
    Public Property Credits() As Integer
    Public Overridable Property Enrollments() As ICollection(Of
Enrollment)
End Class
Public Class Student
    Public Property StudentID() As Integer
    Public Property LastName() As String
    Public Property FirstMidName() As String
    Public Property EnrollementDate() As DateTime
    Public Overridable Property Enrollments() As ICollection(Of
Enrollment)
End Class
```

Y a continuacion el Modelo de la base de datos que EntityFramework basandose en unas configuraciones por defecto, generará

```
CREATE TABLE [dbo].[Enrollment] (
    [EnrollmentID] INT IDENTITY (1, 1) NOT NULL,
    [Grade]        INT NOT NULL,
    [CourseID]     INT NOT NULL,
    [StudentID]    INT NOT NULL,
    CONSTRAINT [PK_dbo.Enrollment] PRIMARY KEY CLUSTERED ([
EnrollmentID] ASC),
    CONSTRAINT [FK_dbo.Enrollment_dbo.Course_CourseID] FOREIGN KEY (
[CourseID]) REFERENCES [dbo].[Course] ([CourseID]) ON DELETE CASCADE,
    CONSTRAINT [FK_dbo.Enrollment_dbo.Student_StudentID] FOREIGN KEY (
[StudentID]) REFERENCES [dbo].[Student] ([StudentID]) ON DELETE CASCADE
);
CREATE TABLE [dbo].[Course] (
    [CourseID] INT            IDENTITY (1, 1) NOT NULL,
    [Title]    NVARCHAR (MAX) NULL,
    [Credits]  INT            NOT NULL,
    CONSTRAINT [PK_dbo.Course] PRIMARY KEY CLUSTERED ([CourseID] ASC)
);
CREATE TABLE [dbo].[Student] (
    [StudentID]       INT            IDENTITY (1, 1) NOT NULL,
    [LastName]        NVARCHAR (MAX) NULL,
    [FirstMidName]    NVARCHAR (MAX) NULL,
    [EnrollementDate] DATETIME       NOT NULL,
    CONSTRAINT [PK_dbo.Student] PRIMARY KEY CLUSTERED ([StudentID] ASC)
);
```

Se puede observar como para la Entidad Enrollment, se definen dos pares de propiedades que aparentemente describen una relacion con Course y con Student, la idea de duplicar la relacion, es por que unas propiedades reflejan las relaciones reales **CourseID** y **StudentID**, y las otras reflejan la relacion a nivel de objetos.

Inicializacion de la base de datos

Para inicializar la base de datos, se puede escribir une **Seed** que previa configuracion será invocado automáticamente después de crear la base de datos para rellenarla con datos de prueba.

El Seed esta representado por una clase que hereda de **System.Data.Entity.DropCreateDatabaseIfModelChanges**

```
Namespace ContosoUniversity.DAL
    Public Class SchoolInitializer
```

```
        Inherits DropCreateDatabaseIfModelChanges(Of SchoolContext)

        Protected Overrides Sub Seed(context As SchoolContext)
            MyBase.Seed(context)

            Dim students As List(Of Student) = New List(Of Student)()
            students.Add(New Student With {.FirstMidName = "Carson",
.LastName = "Alexander", .EnrollementDate = DateTime.Parse("2005-09-
01")})
            students.Add(New Student With {.FirstMidName = "Arturo",
.LastName = "Anand", .EnrollementDate = DateTime.Parse("2003-09-01")})
            students.Add(New Student With {.FirstMidName = "Gytis",
.LastName = "Barzdukas", .EnrollementDate = DateTime.Parse("2002-09-
01")})
            students.Add(New Student With {.FirstMidName = "Yan",
.LastName = "Li", .EnrollementDate = DateTime.Parse("2002-09-01")})
            students.Add(New Student With {.FirstMidName = "Peggy",
.LastName = "Justice", .EnrollementDate = DateTime.Parse("2001-09-
01")})
            students.Add(New Student With {.FirstMidName = "Laura",
.LastName = "Norman", .EnrollementDate = DateTime.Parse("2003-09-01")})
            students.Add(New Student With {.FirstMidName = "Nino",
.LastName = "Olivetto", .EnrollementDate = DateTime.Parse("2005-09-
01")})

            For Each student As Student In students
                context.Students.Add(student)
            Next

            context.SaveChanges()

            Dim courses As List(Of Course) = New List(Of Course)()
            courses.Add(New Course With {.CourseID = 1050, .Title =
"Chemistry", .Credits = 3})
            courses.Add(New Course With {.CourseID = 4022, .Title =
"Microeconomics", .Credits = 3})
            courses.Add(New Course With {.CourseID = 4041, .Title =
"Macroeconomics", .Credits = 3})
            courses.Add(New Course With {.CourseID = 1045, .Title =
"Calculus", .Credits = 4})
            courses.Add(New Course With {.CourseID = 3141, .Title =
"Trigonometry", .Credits = 4})
            courses.Add(New Course With {.CourseID = 2021, .Title =
"Composition", .Credits = 3})
            courses.Add(New Course With {.CourseID = 2042, .Title =
```

```
"Literature", .Credits = 4})

            For Each course As Course In courses
                context.Courses.Add(course)
            Next

            context.SaveChanges()

            Dim enrollments As List(Of Enrollment) = New List(Of
Enrollment)()
            enrollments.Add(New Enrollment With {.StudentID = 1,
.CourceID = 1050, .Grade = Grade.A})
            enrollments.Add(New Enrollment With {.StudentID = 1,
.CourceID = 4022, .Grade = Grade.C})
            enrollments.Add(New Enrollment With {.StudentID = 1,
.CourceID = 4041, .Grade = Grade.B})
            enrollments.Add(New Enrollment With {.StudentID = 2,
.CourceID = 1045, .Grade = Grade.B})
            enrollments.Add(New Enrollment With {.StudentID = 2,
.CourceID = 3141, .Grade = Grade.F})
            enrollments.Add(New Enrollment With {.StudentID = 2,
.CourceID = 2021, .Grade = Grade.F})
            enrollments.Add(New Enrollment With {.StudentID = 3,
.CourceID = 1050})
            enrollments.Add(New Enrollment With {.StudentID = 4,
.CourceID = 1050})
            enrollments.Add(New Enrollment With {.StudentID = 4,
.CourceID = 4022, .Grade = Grade.F})
            enrollments.Add(New Enrollment With {.StudentID = 5,
.CourceID = 4041, .Grade = Grade.C})
            enrollments.Add(New Enrollment With {.StudentID = 6,
.CourceID = 1045})
            enrollments.Add(New Enrollment With {.StudentID = 7,
.CourceID = 3141, .Grade = Grade.A})

            For Each enrollment As Enrollment In enrollments
                context.Enrollments.Add(enrollment)
            Next

            context.SaveChanges()

        End Sub
    End Class
End Namespace
```

La configuracion consiste en añadir al **Web.config** lo siguiente en la seccion ya existente de **entityFramework**

```
<entityFramework>
  <contexts>
    <context type="ContosoUniversity.DAL.SchoolContext,
ContosoUniversity">
      <databaseInitializer type=
"ContosoUniversity.DAL.SchoolInitializer, ContosoUniversity" />
    </context>
  </contexts>
  <defaultConnectionFactory type=
"System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
EntityFramework">
    <parameters>
      <parameter value="v11.0" />
    </parameters>
  </defaultConnectionFactory>
  <providers>
    <provider invariantName="System.Data.SqlClient" type=
"System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
  </providers>
</entityFramework>
```

En la configuracion, se define a nivel de **context**

- El nombre completo de la clase que implementa el contexto
- El ensamblado donde se encuentra

Y a nivel de **databaseInitializer**

- El nombre completo de la clase que implementa el inicializador del contexto
- El ensamblado donde se encuentra

Para desactivar la inicializacion, se puede indicar con la propiedad **disableDatabaseInitialization** de **context**

Personalizacion del Modelo

Se pueden personalizar los tipos de datos y las constraints del modelo empleando **Atributos** (Anotaciones) que encontraremos en los Namespaces

System.ComponentModel.DataAnnotations y **System.ComponentModel.DataAnnotations.Schema**

La principales serian:

- Key: Permite definir que campos forman la clave primaria. De no especificarse, de forma predefinida, se asigna la PK a un campo denominado **<Entidad>ID** o **ID**. Pueden formarla mas de una propiedad

```
<Key>
Public Property CourseID As Integer
```

- DatabaseGenerated: permte indicar el procedimiento de generacion de una clave. Los posibles valores son: None, Identity y Computed

```
<DatabaseGenerated(DatabaseGeneratedOption.None)>
```

- ForeignKey: Permite identificar quien es el extremo que alberga la relacion.

```
<ForeignKey("Instructor")]>
```

- Table: Permite definir el nombre de la tabla de Base de datos con la que se asociará la entidad

```
<Table("Student")>
Public Class Student
End Class
```

- Column: Permite definir las caracteristicas del campo de la tabla con el que se asocia la propiedad de la entidad

```
<Column(TypeName = "money")>
```

- DataType: Indica el tipo de dato que alberga el campo de forma mas especifica, en el siguiente ejemplo se esta limitando el campo a represntar unicamente la fecha y no la hora.

```
<DataType(DataType.Date)>
Public Property Fecha As Date
    Get
        Return _Fecha
    End Get
    Set(value As Date)
        _Fecha = value
    End Set
End Property
```

- DisplayFormat: Indica e formato valido de representacion

```
<DataType(DataType.Date)>
<DisplayFormat(DataFormatString:= "{0:yyyy-MM-dd}",
ApplyFormatInEditMode:= true)>
Public Property Fecha As Date
    Get
        Return _Fecha
    End Get
    Set(value As Date)
        _Fecha = value
    End Set
End Property
```

3.4.2. LinQ

Es una implementacion del patrón **Specification**.

Aparece en la versión 3.5 de .NetFramework.

Permite escribir sentencias cercanas al SQL en programas .net

```
' Obtain a list of customers.
Dim customers As List(Of Customer) = GetCustomers()

' Return customers that are grouped based on country.
Dim queryResults = From cust In customers
                   Where cust.Country = "Canada"
                   Select cust.CompanyName, cust.Country
```

La idea es que LINQ es capaz de consultar cualquier objeto que implemente el

patrón **Iterator**.

En realidad lo que hay por debajo de LINQ, son unas extensiones del lenguaje que transforman las sentencias LINQ en llamadas a funciones.

Operadores de Consulta

Se disponen de los siguientes operadores que pueden ser empleados en exresiones LINQ

- From: Especifica una colección de origen y una variable de iteración de una consulta.

```
' Returns the company name for all customers for which
' the Country is equal to "Canada".
Dim names = From cust In customers
            Where cust.Country = "Canada"
            Select cust.CompanyName
```

- Select: Declara la proyeccion:

```
' Returns the company name and ID value for each
' customer as a collection of a new anonymous type.
Dim customerList = From cust In customers
                   Select cust.CompanyName, cust.CustomerID
```

- Where: Especifica una condición de filtrado para una consulta.

```
' Returns all product names for which the Category of
' the product is "Beverages".
Dim names = From product In products
            Where product.Category = "Beverages"
            Select product.Name
```

- Order By: Especifica el criterio de ordenación de columnas en una consulta.

```
' Returns a list of books sorted by price in
' ascending order.
Dim titlesAscendingPrice = From b In books
                           Order By b.price
```

- Join: Combina dos colecciones en una sola colección. Por ejemplo:

```
' Returns a combined collection of all of the
' processes currently running and a descriptive
' name for the process taken from a list of
' descriptive names.
Dim processes = From proc In Process.GetProcesses
                Join desc In processDescriptions
                  On proc.ProcessName Equals desc.ProcessName
                Select proc.ProcessName, proc.Id, desc.Description
```

- Group By: Agrupa los elementos de los resultados de una consulta. Se puede utilizar para aplicar funciones de agregado a cada grupo.

```
' Returns a list of orders grouped by the order date
' and sorted in ascending order by the order date.
Dim orderList = From order In orders
                Order By order.OrderDate
                Group By OrderDate = order.OrderDate
                Into OrdersByDate = Group
```

- Group Join: Combina dos colecciones en una sola colección jerárquica.

```
' Returns a combined collection of customers and
' customer orders.
Dim customerList = From cust In customers
                   Group Join ord In orders On
                     cust.CustomerID Equals ord.CustomerID
                   Into CustomerOrders = Group,
                        TotalOfOrders = Sum(ord.Amount)
                   Select cust.CompanyName, cust.CustomerID,
                          CustomerOrders, TotalOfOrders
```

- Aggregate: Aplica una o más funciones agregadas a una colección. Por ejemplo, puede utilizar la cláusula Aggregate para calcular una suma de todos los elementos devueltos por una consulta.

```
' Returns the sum of all order amounts.
Dim orderTotal = Aggregate order In orders
                 Into Sum(order.Amount)
```

También puede utilizar la cláusula Aggregate para modificar una consulta. Por ejemplo, puede utilizar la cláusula Aggregate para realizar un cálculo en una colección de consultas relacionada.

```
' Returns the customer company name and largest
' order amount for each customer.
Dim customerMax = From cust In customers
                  Aggregate order In cust.Orders
                  Into MaxOrder = Max(order.Amount)
                  Select cust.CompanyName, MaxOrder
```

- Let: Calcula un valor y lo asigna a una nueva variable en la consulta.

```
' Returns a list of products with a calculation of
' a ten percent discount.
Dim discountedProducts = From prod In products
                         Let Discount = prod.UnitPrice * 0.1
                         Where Discount >= 50
                         Select prod.Name, prod.UnitPrice, Discount
```

- Distinct: Restringe los valores de la variable de iteración actual para eliminar los valores duplicados en los resultados de la consulta

```
' Returns a list of cities with no duplicate entries.
Dim cities = From item In customers
             Select item.City
             Distinct
```

- Skip: Omite un número especificado de elementos de una colección y, a continuación, devuelve los elementos restantes.

```
' Returns a list of customers. The first 10 customers
' are ignored and the remaining customers are
' returned.
Dim customerList = From cust In customers
                   Skip 10
```

- Skip While: Omite los elementos de una colección siempre que el valor de una condición especificada sea true y, a continuación, devuelve los elementos

restantes.

```
' Returns a list of customers. The query ignores all
' customers until the first customer for whom
' IsSubscriber returns false. That customer and all
' remaining customers are returned.
Dim customerList = From cust In customers
                   Skip While IsSubscriber(cust)
```

- Take: Devuelve un número especificado de elementos contiguos desde el principio de una colección.

```
' Returns the first 10 customers.
Dim customerList = From cust In customers
                   Take 10
```

- Take While: Incluye los elementos de una colección siempre que el valor de una condición especificada sea true y, a continuación, omite los elementos restantes.

```
' Returns a list of customers. The query returns
' customers until the first customer for whom
' HasOrders returns false. That customer and all
' remaining customers are ignored.
Dim customersWithOrders = From cust In customers
                          Order By cust.Orders.Count Descending
                          Take While HasOrders(cust)
```

4. Enrutado

Como ya hemos visto las rutas permiten asociar URL con métodos de acción de los controladores.

Esta visión de las URLs, permite facilitar su escritura y agrupar logicas.

4.1. Definición de rutas

Por defecto las rutas de definen en el fichero **/App_Start/RouteConfig.vb**, en el método **RegiterRoutes**, el cual es invocado desde **Global.asax**.

Por defecto se incluye la siguiente configuración

```
Public Module RouteConfig
    Public Sub RegisterRoutes(ByVal routes As RouteCollection)

        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

        routes.MapRoute(
            name:="Default",
            url:="{controller}/{action}/{id}",
            defaults:=New With {.controller = "Home", .action =
"Index", .id = UrlParameter.Optional}
        )
    End Sub
End Module
```

En ella se pueden ver varias cosas, si nos centramos en la sentencia

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

Se ve como se define un patrón de ruta {controller}/{action}/{id}, que esta parametrizado con

- {controller}: Representa la clase de Controller sin el Sufijo Controller.

- {action}: El metodo de acción
- {id}: El parametro id que le llegará al método de acción.

En los patrones se pueden definir los parametros que se quieran, siempre con la restricción de que no se pongan seguidos, debe haber algun caracter entre ellos, en este caso /.

Tambien se están definiendo los valores por defecto para los parametros, en este caso de no indicarse un controller, es decir el primer nivel del PATH, se tomará Home, esto sucede cuando se introduce la URL <a href="http://<host>:<port>" class="bare">http://<host>:<port&gt;

Si lo que no se incluye es el action, o el segundo nivel del PATH, se tomará como valor de action **Index**, esto sucederia con la URL **http://<host>:<port>/<cualquier controller>**

Si lo que no se incluye es el **id**, o el tercer nivel del PATH, no pasa nada ya que está declarado como opcional, por lo que si esta se considera y sino nada, luego son validas URL como **http://<host>:<port>/<cualquier controller>/<cualquier action>**, **http://<host>:<port>/Personajes/Consulta/1** o **http://<host>:<port>/Personajes/Consulta/Hommer**

Donde **1** o **Hommer** son considerados como **id**, siempre que el método de acción invocado reciba un parametro de tipo int o string denominado id

```
Function Consulta(id As Integer) As ActionResult
    Return View()
End Function

Function Consulta(id As String) As ActionResult
    Return View()
End Function
```

Si se quieren capturar todas las secciones de un PATH, independientemente del numero de ellas que haya, se puede emplear la siguiente sintaxis en el ultimo elemento de la ruta {*restopath}

```
routes.MapRoute(
    name:="Default",
    url:="{controller}/{action}/{id}"
)
```

4.1.1. Constraints

Se puede añadir un cuarto parametro al Mapeo de la Ruta, siendo este las restriciones (constraints) de los elementos que componen la ruta.

```
routes.MapRoute(
    name:="Blog",
    url:="{controller}/{action}/{year}/{month}/{day}",
    defaults:={},
    constraints:=New With {.controller = "Blog", .year = "\d{4}",
.month = "\d{2}", .day = "\d{2}"}
)
```

En el ejemplo, se estarán procesando URL del tipo
http://<host>:<port>/Blog/<Accion>/2012/02/05

Donde se ejecutará un método de acción de la clase controlador Blog, lo normal es que el método de acción tuviese como parametros year, month y day.

```
Function Index(year As Integer, month As Integer, day As Integer) As
ActionResult
   Return View()
End Function
```

Para la definición de las constraints, se emplean expresiones regulares.

4.2. Ignorar rutas

Por defecto el sistema de enrutado no se aplica sobre ficheros fisicos, es decir primero se busca el recurso pedido fisicamente y si se encuentra se retorna y sino, se busca un método de acción empleando el sistema de enrutado.

Como pueden existir ficheros fisicos que no se desee que sean accedidos directamente, se pueden incluir PATH a ignorar.

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}")
```

4.3. Routing Attributes

Es una nueva funcionalidad en ASP.NET MVC 5, que permite delegar la declaración de las rutas de los métodos de acción a los propios métodos de acción, en lugar de a la clase RouteConfig.

Lo primero que habrá que hacer es registrar esta nueva funcionalidad, para ello, en RouteConfig, se ha de sustituir lo que viene por defecto por

```
Public Module RouteConfig
    Public Sub RegisterRoutes(ByVal routes As RouteCollection)
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")

        routes.MapMvcAttributeRoutes()

    End Sub
End Module
```

NOTE	Se ha de tener en cuenta que el orden de resolucion de las rutas, es el orden de definición en **RegisterRoutes**, se va preguntando a las distintas rutas definidas, hasta que una responda afirmativamente y en ese momento se deja de preguntar.
NOTE	Se pueden mantener las dos aproximaciones, pero habrá que definir las Rutas con **AttributeRutes** primero, dado que son mas concretas y las tradicionales mas genericas.

Y luego habra que añadir a las clases Controller las siguientes anotaciones hasta conseguir el comportamiento deseado:

- Route: Permite indicar la ruta completa para acceder al recurso.

```
Public Class BlogController
    Inherits Controller

    ' GET: Blog
    <Route("index")>
    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

Se puede definir a nivel del controlador o a nivel del método de acción

```
<Route("index")>
Public Class BlogController
    Inherits Controller

    ' GET: Blog
    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

De definirse en el método de acción, cualquier definición en el controlador quedará sobreescrito y no será valido.

```
<Route("rutas")>
Public Class RutasController
    Inherits Controller

    ' GET: /rutas No es valido
    ' GET: /index
    <Route("index")>
    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

Se pueden definir tantas rutas como se quieran asociado a un método de acción.

```
Public Class RutasController
    Inherits Controller

    ' GET: /
    ' GET: /index
    <Route("")>
    <Route("index")>
    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

Las rutas son parametrizables.

```
<Route("home/{action}")>
```

Se pueden definir valores por defecto de los parametros.

```
<Route("home/{action=Index}")>
```

Se puede indicar que un parametro es opcional, normalmente se hará a nivel del controlador.

```
<Route("{action}/{id?}")>
```

- RoutePrefix: Permite definir a nivel del controlador, un prefijo para todas las rutas definidas en dicho controlador, es decir en el resto de **<Route>**.

```
<RoutePrefix("rutas")>
Public Class RutasController
    Inherits Controller

    ' GET: /rutas
    ' GET: /rutas/index
    <Route("")>
    <Route("index")>
    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

Si en algún caso, se quiere ignorar el [RoutePrefix], se puede emplear la virgulilla

```
<RoutePrefix("rutas")>
Public Class RutasController
    Inherits Controller

    ' GET: /
    <Route("~/")>
    ' GET: /rutas
    <Route("")>
    ' GET: /rutas/index
    <Route("index")>
    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

4.3.1. Route Constraints

Se pueden definir asociadas a los parametros de las rutas, una serie de constraints, que afectan en general al tipo de dato que represente al parametro, aunque se puede recurrir a una expresión regular, tamaños, …

- {n:bool} - Valor booleano
- {n:datetime} - Valor Fecha. Los formatos validos son **2016-12-31** y **2016-12-31 7:32pm**
- {n:decimal} - Valor Decimal

- {n:double} - Valor Double
- {n:float} - Valor Float
- {n:guid} - Valor Guid
- {n:int} - Valor Int32
- {n:long} - Valor Int64
- {n:minlength(2)} - Valor String de al menos 2 caracteres
- {n:maxlength(2)} - Valor String de maximo 2 caracteres
- {n:length(2)} - Valor String de 2 caracteres
- {n:length(2,4)} - Valor String con 2, 3 o 4 caracteres.
- {n:min(1)} - Valor Int64 mayor o igual a 1
- {n:max(3)} - Valor Int64 menor o igual que 1
- {n:range(1,3)} - Valor Int64 entre 1 y 3
- {n:alpha} - Valor String alfanumerico, con caracteres entre A–Z y a–z.
- {n:regex (^a+$)} - Valor String que contenga solo una o mas 'a'

```
<RoutePrefix("rutas")>
Public Class RutasController
    Inherits Controller

    ' GET /rutas/Madrid/2017-01-02
    <Route("{Ciudad}/{Fecha:datetime}")>
    Function Index(Ciudad As String, Fecha As DateTime) As ActionResult
        Return View()
    End Function
End Class
```

5. Controladores y Acciones

Los Controladores, son los encargados de responder a las peticiones HTTP del navegador y devolver la información al mismo.

5.1. IController, ControllerBase, y Controller

En ASP.NET MVC, los controladores heredan de la jerarquiaa de System.Web.Mvc.Controller, que a su vez lo hace de System.Web.Mvc.ControllerBase e implmentan System.Web.Mvc.IController.

La clase ControllerBase es la que se encarga de mantener los objetos ControllerContext, ViewBag y ViewData.

La clase Controller da acceso a los objetos Request y Response, que representan la petición y la respuesta.

Recordar que por convenio las clases de los Controladores deben tener como sufijo Controller y estar en la carpeta Controllers.

5.2. Definición de Acciones

Por defecto cada uno de los métodos publicos que componen el controlador será una acción, es decir será una funcionalidad que se podrá invocar desde el cliente indicando una URL.

La URL que se ha de escribir podra ser redefinida con las Rutas, pero por convenio se establece que la URL será

```
http://<host>:<port>/<Nombre de la clase Controlador sin el sufijo
Controller>/<nombre
de la acción>
```

Así pues si se desea ejecutar la acción Index de un Controlador HomeController, se ha de acceder a

```
http://<host>:<port>/Home/Index
```

Los métodos de acción podrán retornar alguno de los siguientes tipos

- String: Representa directamente la respuesta que le llega al Browser.
- ActionResult: Representa el tipo de salida, tiene varios subtipos.

5.3. Selectores de Acciones

Un selector de Acción es aquel que permite definir reglas para que una acción se ejecute solo si se cumplen una serie de requisitos, como el path invocado o el method HTTP con el que se realiza la petición.

El API proporciona las siguientes anotaciones:

- ActionName: Permite definir un path diferente para el método de acción

```
<ActionName("ListarMusicaPorGenero")>
Function MusicaPorGenero(Genero As String) As ActionResult
    Return View()
End Function
```

- NonAction: Permite definir un método publico que no represente una acción dentro de la clase del controlador.

```
<NonAction>
Function MusicaPorGenero(Genero As String) As ActionResult
    Return View()
End Function
```

- ActionVerbs: No existe una anotación llamada así, sino que se proporcionan un grupo de anotaciones, que representan los denominados verbos HTTP.
    - ¬ HttpGet: Representa que las peticiones deberan de llegar por HTTP GET. Es el verbo por defecto, por lo que no se suele emplear la anotación.
    - ¬ HttpPost: Representa que las peticiones deberan de llegar por HTTP POST.
    - ¬ HttpPut: Representa que las peticiones deberan de llegar por HTTP PUT.
    - ¬ HttpDelete: Representa que las peticiones deberan de llegar por HTTP DELETE.
    - ¬ HttpHead: Representa que las peticiones deberan de llegar por HTTP HEAD.
    - ¬ HttpOptions: Representa que las peticiones deberan de llegar por HTTP OPTIONS.

- ¬ HttpPatch: Representa que las peticiones deberan de llegar por HTTP PATCH.

```
<HttpPost()>
Function Create(ByVal collection As FormCollection) As ActionResult
    Try
        ' TODO: Add insert logic here

        Return RedirectToAction("Index")
    Catch
        Return View()
    End Try
End Function
```

Se pueden definir selectores de acción personalizados, sin más que heredar de la clase **System.Web.Mvc.ActionMethodSelectorAttribute**

```
Public Class CustomActionSelector
    Inherits ActionMethodSelectorAttribute

    Public Overrides Function IsValidForRequest(controllerContext As
ControllerContext, methodInfo As MethodInfo) As Boolean
        Throw New NotImplementedException()

        Return False

    End Function
End Class
```

En esa clase se tiene acceso a traves del ControllerContext a los datos de la petición, como: MethodHttp, Cookies, Parametros, …

```
Dim request = controllerContext.HttpContext.Request
Dim cookies = request.Cookies
Dim parametros = request.Params
```

Una vez definido el selector de acción, lo único que resta es emplearlo, y se hace de forma analoga a los proporcionados por el API, con una anotación, que en este caso es la clase que se acaba de definir.

```
<CustomActionSelector>
Function Index() As ActionResult
    Return View()
End Function
```

5.4. Filtros de Acciones

Los permiten encapsular funcionalidades transversales, como el control de excepciones o la autorización.

Permiten definir lógica a ejecutar antes y despues de los **métodos de acción**.

Los filtros pueden aplicarse a controladores o acciones (o pueden ejecutarse globalmente), esto es una diferencia con los selectores de acción, ya que los selectores de accion solo son aplicables a los métodos de acción.

Aplicacion del filtro Autorize sobre un controlador

```
<Authorize>
Public Class LibroController
    Inherits Controller
End Class
```

El API proporciona las siguientes anotaciones:

- OutputCache: Permite cachear la respuesta de la acción, durante un tiempo, siendo la respuesta de la invocación de la acción la misma en ese periodo de tiempo, ya que no se ejecuta la acción.

```
<OutputCache(Duration:=100)>
Function Index() As ActionResult
    Return View(DateTime.Now.ToString("T"))
End Function
```

NOTE	Ojo que una petición no se considera igual a otra si los parametros cambian, por lo que no se cachearia.

- HandleError: Permite manejar las execciones que se lanzan en la ejecución de la acción, indicando la vista que se encargará de mostrar la informacion del error.

```
<HandleError(ExceptionType:=GetType(Exception), View:="Error")>
Function Index() As ActionResult
    Throw New Exception
End Function
```

NOTE

Para activar esta caracteristica, hay que activar los errores personalizazos en el **Web.config**

```
<system.web>
    <customErrors mode="On"/>
</system.web>
```

Los posibles valores del atributo mode son On, Off y RemoteOnly.

- Authorize: Permite restringir el acceso a una acción a usuarios autentificados, o incluso a usuarios con un determinado Rol o a un Usuario concreto.

```
<Authorize>
Public Class LibroController
    Inherits Controller
End Class

<Authorize(Users:="Alice,Bob")>
Public Class LibroController
    Inherits Controller
End Class

// Restrict by role:
<Authorize(Roles:="Administrators")>
Public Class LibroController
    Inherits Controller
End Class
```

- AllowAnonymous: Permite el acceso a una acción sin estar autentificado, se emplea junto con Authorize cuando esta anota la clase entera para permitir el acceso a una acción de forma anonima.

```
<Authorize>
Public Class LibroController
    Inherits Controller

    ' GET: Libro
    <AllowAnonymous>
    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

El API proporciona la posibilidad de crear filtros personalizados, basados en la clase **System.Web.Mvc.ActionFilterAttribute**, en la cual se pueden sobreescribir los siguientes métodos:

- OnActionExecuting: Se invoca antes de que se ejecute el método de acción.
- OnActionExecuted: Se invoca despues de que se ejecute el método de acción y antes de que el resultado sea ejecutado, es decir antes de la renderización de la vista.
- OnResultExecuting: Se invoca antes de que el resultado sea ejecutado, es decir antes de la renderización de la vista.
- OnResultExecuted: Se invoca despues de que el resultado sea ejecutado, es decir despues de la renderización de la vista.

```
Public Class CustomActionFilter
    Inherits ActionFilterAttribute

    Public Overrides Sub OnActionExecuting(filterContext As
ActionExecutingContext)
        MyBase.OnActionExecuting(filterContext)

        Debug.WriteLine("En OnActionExecuting")

    End Sub

    Public Overrides Sub OnActionExecuted(filterContext As
ActionExecutedContext)
        MyBase.OnActionExecuted(filterContext)

        Debug.WriteLine("En OnActionExecuted")

    End Sub

    Public Overrides Sub OnResultExecuting(filterContext As
ResultExecutingContext)
        MyBase.OnResultExecuting(filterContext)

        Debug.WriteLine("En OnResultExecuting")

    End Sub

    Public Overrides Sub OnResultExecuted(filterContext As
ResultExecutedContext)
        MyBase.OnResultExecuted(filterContext)

        Debug.WriteLine("En OnResultExecuted")

    End Sub
End Class
```

Para emplear el filtro en un método de acción o controlador unicamente se ha de incluir la anotación de la siguiente forma

```
<CustomActionFilter>
Function Index() As ActionResult
    Return View(DateTime.Now.ToString("T"))
End Function
```

Si se desea que cualquiera de estos filtros se aplique a todos los métodos de acción de la aplicación, habra que añadir en el fichero FilterConfig que esta en App_Start el registro del Filtro en los Filtros globales.

```
Public Module FilterConfig
    Public Sub RegisterGlobalFilters(ByVal filters As
GlobalFilterCollection)
        filters.Add(New HandleErrorAttribute())
        filters.Add(New CustomActionFilterAttribute)
    End Sub
End Module
```

5.5. HttpContext

La propiedad **HttpContext** de la clase **Controller**, es una instancia de **System.Web.HttpContextBase**, la cual proporciona acceso a los objetos:

- Application: Permite el acceso a objetos compartidos por todas las sesiones, representa el contexto de la aplicacion web.

```
HttpContext.Application.Add("clave", "valor")
Dim valor = HttpContext.Application.Get("clave")
```

- Session: Permite el acceso a objetos dentro de una session, compartido para todas las peticiones dentro de la session.

```
HttpContext.Session.Add("clave", "valor")
Dim otro = HttpContext.Session.Item("clave")
```

- Request: Objeto que representa la petición.

```
Dim cookies = HttpContext.Request.Cookies
```

- Response: Objeto que representa la respuesta.

```
Dim contentType = HttpContext.Response.ContentType
```

- User: Objeto que representa el usuario autenticado en la aplicación.

```
If HttpContext.User.IsInRole("Admin") Then

End If
```

Esta información, tambien puede ser accedida por cualquier otra clase dentro de la aplicacion ASP.NET, a traves de la clase System.Web.HttpContext y mas concretamente con su propiedad Current.

```
Dim clave = HttpContext.Current.Session.Item("clave")
```

5.6. RouteData

La propiedad **RouteData** del **Controller**, es una instancia de la clase **System.Web.Routing.RouteData** que provee acceso a los valores que forman la ruta actual.

```
Dim ruta = RouteData.Route
```

5.7. ActionResult

Es el tipo de dato que se puede retornar por parte de los métodos de acción a parte de un String, es un tipo base, del cual heredan los siguientes tipos

- 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 dirección 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).

La clase Controller, proporciona métodos como:

- View(): Permite indicar la View a retornar como resultado de la ejecución de la acción. De no indicarse nada, buscará una View con el nombre del método de acción. Este tipo de resultado aplica el layout de pagina.

```
Function Index() As ActionResult
    Return View()
End Function
```

- PartialView(): Permite indicar la View a retornar como resultado de la ejecución de la acción. De no indicarse nada, buscará una View con el nombre del método de acción. Este tipo de resultado **no** aplica el layout de pagina.

```
Function Index() As ActionResult
    Return PartialView()
End Function
```

- Redirect(): Permite redirecionar a una URL cualquiera.

```
Function Index() As ActionResult
    Return Redirect("http://www.google.es")
End Function
```

- RedirectToRoute(): Permite redireccionar a una acción del proyecto, indicando la ruta para llegar a ella.

```
Function Index() As ActionResult
    Return RedirectToRoute(New With {.controller = "Controlador",
.action = "RetornoJson"})
End Function
```

- RedirectToAction(): Permite redireccionar a una acción del proyecto.

```
Function Index() As ActionResult
    Return RedirectToAction("RetornoJavascript")
End Function
```

- Javascript(): Permite retorna codigo javascript como respuesta.

```
Function Index() As ActionResult
    Return JavaScript("alert('Hola, Este es un mensaje del servidor')")
End Function
```

Este **ActionResult**, se emplea junto con la etiqueta HTML <script> en alguna de las vistas.

```
<script type="text/javascript" src="/Controlador/RetornoJavascript"
></script>
```

- Json(): Permite retornar un JSON como respuesta

```
Function Index() As ActionResult
    Return Json(New With{ .Name = "Hommer Simpson", .ID = 1,
.DateOfBirth = new DateTime(1955, 05, 12) },
JsonRequestBehavior.AllowGet)
End Function
```

NOTE

Para retornar un JSON hay que activar JsonRequestBehavior.AllowGet dado que por defecto no esta permitido obtener datos en formato JSON por HttpGet.

Esto es motivado porque en navegadores antiguos, se podia redefinir la funcion setter del tipo Array, por lo que al retornar el servidor un Array en formato JSON, este metodo se ejecuta y al estar sobreescrito, el comportamiento no es el esperado y por tanto es una vulnerabilidad, la vulnerabilidad afecta a los datos procesados, ya que pueden ser distribuidos a terceros no autorizados, pero no a la aplicacion.

- Content(): Permite retornar lo que se quiera partiendo de un String.

```
Function Index() As ActionResult
    Return Content("<h3>Mi HTML</h3>", MediaTypeNames.Text.Html)
End Function
```

NOTE

En la clase System.Net.Mime.MediaTypeNames, se pueden encontrar los MimeType mas habituales como constantes.

- File(): Permite indicar un fichero que se retornará como resultado de la acción.

```
Function Index() As ActionResult
    Return File(Url.Content("~/Files/testfile.txt"),
MediaTypeNames.Text.Plain)
End Function
```

NOTE

La virgulilla (~), permite referenciar la raiz de directorios del proyecto.

En Windows se obtiene con **Alt Gr + 4**

5.8. Parámetros

Lo parametros permiten a la aplicación recibir información desde el cliente.

```
http://<host>:<port>/<Nombre de la clase Controlador sin el sufijo
Controller>/<nombre de la acción>?<clave del parametro>=<valor del
parametro>
```

Por ejemplo

```
http://<host>:<port>/TiendaDeMusica/MusicaPorGenero?genero=Pop
```

Para recibir esta información en un método de acción, basta con incluir un parametro a la firma del método que se llame igual que el parametro que se envia en la petición

```
Function Index(Genero As String) As ActionResult
    Return View()
End Function
```

Otra forma de recibir información, es a traves de la propia URL, es el caso del enrutado por defecto de MVC que incluye un parametro **id** opcional como parte de la ruta, como el siguiente nivel despues del método de acción, algo como

```
http://<host>:<port>/TiendaDeMusica/DetalleDisco/1
```

Donde en el ejemplo, el valor 1, será el id del disco a visualizar, para recibir este dato en el método de acción se ha de proceder de forma analoga a como se hizo con los parametros de la peticion

```
Function Index(Id As Integer) As ActionResult
    Return View()
End Function
```

No se pueden definir dos métodos con el mismo nombre, sin cambiar el enrutado, ya que de lo contrario habrá una ruta ambigua que no se sabrá resolver, por no saber que metodo de acción ejecutar.

NOTE

Dado que puede ser peligroso emplear directamente datos de tipo String provenientes desde el cliente, ya que pueden contener codigo malicioso, es recomendable realizar un procesado de estos valores, para ello el API proporciona la clase **HttpUtility.HtmlEncode**

```
HttpUtility.HtmlEncode("<dato recibido desde el
cliente>")
```

5.9. Model Binder

Sucede que normalmente la información recibida no será útil como String, Int, … separados, sino que formarán juntos un dato mas complejo (generalmente un modelo), en estos casos el API, proporciona la funcionalidad Model Binder, que permite obtener directamente el objeto del modelo a partir de los parametros que se reciben en la petición.

Un ejemplo, dado un Controlador con un método de Acción

```
public class PersonasController : Controller
{
    public ActionResult Alta(Persona persona)
    {
    }
}
```

Y la siguiente clase Persona

```
public class Persona
{
    public int Id { get; set; }
    public String Nombre { get; set; }
    public String Apellido { get; set; }
    public DateTime FechaNacimiento{ get; set; }
    public Persona(int id, String nombre, String apellido, DateTime
fechaNacimiento)
    {
        Id = id;
        Nombre = nombre;
        Apellido = apellido;
        FechaNacimiento = fechaNacimiento;
    }
    public Persona()
    {
    }
}
```

Se puede hacer llegar la información de la persona con la siguiente URL

```
http://<host>:<port>/Personas/Alta?nombre=Hommer&apellido=Simpson&fecha
nacimiento=05/12/1955
```

NOTE El formato de las fechas (DateTime) por defecto será **MM/dd/yyyy**, por lo que salvo que se defina un Binding personalizado habrá que respetarlo.

NOTE Ojo con los constructores en las clases de Modelo, ya que deben tener siempre el constructor sin paremetros.

La información tambien podria llegar en el cuerpo de una petición POST, por ejemplo desde un formulario

```
<form action="Alta" method="POST">
    id <input id="id" name="id" type="text" />
    Nombre <input id="nombre" name="nombre" type="text" />
    Apellido <input id="apellido" name="apellido" type="text" />
    Fecha <input id="fechaNacimiento" name="fechaNacimiento" type="
date" />
    <input id="Submit" type="Submit" value="Submit" />
</form>
```

NOTE Al enviar los datos por POST, hay que asegurarse de que el Content-Type de la petición es **application/x-www-form-urlencoded**

Se puede definir el proceso de Binder de forma personalizada, para ello hay que definir una clase que implemente la interface System.Web.Mvc.IModelBinder, donde se definan las conversiones que haya que realizar entre lo que llega en la petición y el modelo que se debe conseguir.

```
public class PersonaModelBinder : IModelBinder
{
    public object BindModel(ControllerContext controllerContext,
ModelBindingContext bindingContext)
    {
        HttpRequestBase request =
controllerContext.HttpContext.Request;
        int id = 0;
        int.TryParse(request.Form.Get("Id"), out id);
        String nombre = request.Form.Get("Nombre");
        String apellido = request.Form.Get("Apellido");
        DateTime fechaNacimiento;
        DateTime.TryParse(request.Form.Get("FechaNacimiento"), out
fechaNacimiento);
        Persona persona = new Persona(id, nombre, apellido,
fechaNacimiento);
        return persona;
    }
}
```

Una vez definido el nuevo Binder, se puede decidir emplear el Binder en un método de acción concreto empleando la anotacion System.Web.Mvc.ModelBinder

```
public String
RecibiendoUnaPersona([ModelBinder(typeof(PersonaModelBinder))] Persona
persona)
{
}
```

O registrarlo como Binder de la aplicación para el tipo de dato correspondiente, esto se hace en el fichero Global.asax, en la clase de tipo System.Web.HttpApplication, en el método Application_Start.

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
        ModelBinders.Binders.Add(typeof(Persona), new
PersonaModelBinder());
    }
}
```

5.10. Acciones Asíncronas

Funcionalidad que permite la reutilización de los hilos de ejecucion del servidor, cuando como parte de una petición se realiza una tarea de larga duración, que evidentemente no debe estar haciendo uso del hilo, por ejemplo una consulta a la BD o una operacion I/O.

En estos casos si la acción no es asincrona, el hilo permanece parado, sin procesar ninguna sentencia más, en cambio con acciones asincronas ese hilo vuelve al pool de hilos del servidor donde si se necesita procede a atender a otras peticiones, hasta que la tarea asincrona lo vuelva a necesitar.

Una petición asincróna tarda el mismo tiempo en procesarse que una sincróna, lo que aporta la asincróna es que no bloquea la capacidad del servidor de responder a otras peticiones mientras espera a que se complete la primera, por tanto, las peticiones asincrónas evitan encolar las peticiones cuando se podrucen muchas al unisono y entre ellas hay de larga duración.

Para implementar un método de acción asincrono, se ha de:

- Añadir el modificador async al método de acción.
- Cambiar el tipo de dato retornado por System.Threading.Tasks<ActionResult>.
- Marcar la tarea (sentencia) de larga duración con el modificador await.

```
public async Task<ActionResult> TareaAsincrona(string city)
{
    int resultado = await TareaLargaDuracionAsincrona(1);
    return View("Generos");
}
```

NOTE Las tareas de larga duración asincronas, deben retornar información.

Un ejemplo de tarea de larga duracion podria ser

```
private Task<int> TareaLargaDuracionAsincrona(int num)
{
    return Task.Factory.StartNew(() =>
        {
            Thread.Sleep(10000); // Simulate a very hard task
            return 42;
        });
}
```

Se puede realizar la ejecución en paralelo de varias tareas de larga duración, para ello se emplea la clase **System.Threading.Tasks**

```
public async Task<ActionResult> Listado()
{
    await Task.WhenAll(TareaLargaDuracionAsincrona(1),
    TareaLargaDuracionAsincrona(2));
    return View();
}
```

La duración en este caso de la petición, será el de la tarea mas larga.

6. Vistas

Son las encargadas de ofrecer una interface de usuario (UI) al cliente, donde se mostrarán formateados los datos que el controlador obtuvo.

Por convención cuando un controlador retorna

```
Return View()
```

Se buscará un fichero cshtml dentro de la carpeta Views, en la ruta marcada por el controlador y el método de acción donde se encuentre, así pues, si el controlador es HomeController y el método de acción es Index

```
Public Class HomeController
    Inherits System.Web.Mvc.Controller

    Function Index() As ActionResult
        Return View()
    End Function
End Class
```

El fichero que se buscará será **/Views/Home/Index.vbhtml**

El nombre de la vista se puede seleccionar, omitiendo el comportamiento por defecto, tanto dentro de la misma carpeta del controlador

Sentencia qie buscara la el fichero **/Views/Home/NotIndex.vbhtml**

```
Function Index() As ActionResult
    Return View("NotIndex")
End Function
```

Como en cualquier ubicación

```
Return View("~/Views/Example/Index.vbhtml")
```

Desde Visual Studio, se pueden generar Vistas empleando plantillas.

images::vs2017_crear_vista_basada_en_plantillas.png[]

Las plantillas disponibles son:

- Create: Crea una Vista con un formulario para la generación de nuevas instancias del Modelo indicado. Genera una etiqueta y un campo de input para cada propiedad del Modelo.
- Delete: Crea una Vista con un formulario para borrar instancias existentes del Modelo indicado.
- Details: Crea una Vista para mostrar cada propiedad de la instancia del Modelo dado.
- Edit: Crea una Vista con un formulario para editar una instancia del Modelo dado.
- Empty: Crea una Vista vacia, solo se especifica la directiva @model.
- Empty (without model): Crea una Vista vacia.
- List: Crea una Vista con una tabla para representar instancias del Modelo dado, genera una columna por cada propiedad, se ha de pasar un IEnumerable del Modelo dado. La Vista tambien incluye enlaces para las funcionalidades de creacion, editado y borrado.

6.1. El motor Razor View

Las vistas ASP.NET MVC usan el motor de vistas Razor para representar vistas.

Razor es un lenguaje de marcado de plantillas, que permite definir vistas referenciando a código Visual Basic.

Razor se usa para generar dinámicamente contenido web en el servidor. Permite combinar código del servidor con código del cliente.

```
@Code
    ViewData("Title") = "Home Page"
End Code
<head>
<title>@ViewData("Title")</title>
</head>
<ul>
@For Index = 1 To 10
    @<li>List item @Index</li>
Next
</ul>
```

Cuando se habla de sintaxis Razor, se habla de @, es el caracter que precede a cualquier sentencia que deba ser interpretada por el Motor de Razor View.

6.2. Sintaxis Razor

Como se ve en el ejemplo anterior, existen dos posibilidades de definicion de codigo de servidor

- En bloque, empleando la estructura **@Code ...¬ End Code**

```
@Code
    ViewData("Title") = "Home Page"
End Code
```

- Monosentencia, precediendo a la sentencia Visual Basic de @

```
<title>@ViewData("Title")</title>

<ul>
@For Index = 1 To 10
    @<li>List item @Index</li>
Next
</ul>
```

NOTE

Dentro de los bloques de codigo de sevidor, si se desea añadir codigo de cliente (HTML, Texto plano o js), se ha de escapar de nuevo con la @

```
@Code
    For i As Integer = 0 To ViewBag.NumTimes
        @<h3> @ViewBag.Message @i.ToString </h3>
    Next i
End Code
```

Otros casos de uso de @, serian los Imports

```
@Imports Antlr.Runtime
```

Dado que @ es considerado como un caracter especial, si en algun momento se

necesita escribir un literal con @, habrá que escaparlo, para ello @@.

NOTE	En el caso de los correos electronicos normalmente no es necesario escaparlo, dado que el Motor Razor detecta expresiones de tipo e-mail y no lo considera como sentencia Razor.

Si se quiere concatenar un texto justo antes/despues de una sentencia de Razor, es conveniente acotar la sentencia Razor con ()

```
@For Index = 1 To 10
    @<li>List item @(Index)</li>
Next
```

El motor de Razor codifica automaticamente todo lo que representa, por lo que se evita problemas de XSS (Cross Site Scripting). Si se tiene la siguiente situación

```
@Code
    Dim message As String = "<script>alert('haacked!');</script>"
End Code
<span>@message</span>
```

El resultado mostrado por el motor Razor será

```
<span>&lt;script&gt;alert(&#39;haacked!&#39;);&lt;/script&gt;</span>
```

En los casos en los que se quiera considerar el contenido sin codificar, se dispone del siguiente Helper

```
@Code
    Dim message As String = "<strong>This is bold!</strong>"
End Code
<span>@Html.Raw(message)</span>
```

Siendo el resultado mostrado por el motor Razor

```
<span><strong>This is bold!</strong></span>
```

Se pueden comentar bloques de sentencias Razor con @* *@

La sintaxis de Razor tambien dispone de sentencias de bloques como

- @For..To..Next: Que permite iterar de forma analoga a For en VB.

```
@For Index = 1 To 10
    @<li>List item @(Index)</li>
Next
```

- @For..Each: Que permite iterar de forma analoga a For Each en VB.

```
@For Each item As String In lst
    @item
Next
```

- @If..Then..Else..End If: Analoga al If de VB

```
@If count = 0 Then
    message = "There are no items."
ElseIf count = 1 Then
    message = "There is 1 item."
Else
    message = "There are " & count & " items."
End If
```

6.3. ViewData y ViewBag

Ambos objetos permiten transferir información desde el controlador a la vista.

- ViewData: Es un Diccionario, y permite acceder a los valores empleando como claves strings

```
ViewData("Message") = "Your application description page."
```

Si en el controlador se define

```
Dim personajes As IList = New List(Of Personaje) From {
            New Personaje("Hommer"), New Personaje("Marge")}

ViewData("personajes") = personajes
```

En la vista habria que hacer

```
@For Each personaje As Personaje In ViewData("personajes")
    @<li>
        @personaje.Nombre
    </li>
Next
```

- ViewBag: Es un Tipo Dinamico, no tiene estructura en tiempo de compilación, sino que la adopta en tiempo de ejecución, pudiendo tener las propiedades que se necesiten en cada momento.

```
ViewBag.Name = "Hommer Simpson"
```

Si en el controlador se define

```
Dim personajes As IList = New List(Of Personaje) From {
            New Personaje("Hommer"), New Personaje("Marge")}

ViewBag.Personajes = personajes
```

En la vista habria que hacer

```
@For Each personaje As Personaje In ViewBag.Personajes
    @<li>
        @personaje.Nombre
    </li>
Next
```

NOTE	Internamente emplean el mismo diccionario, por lo que no se pueden duplicar las claves
NOTE	Ambos pierden los valores ante las redirecciones

6.4. Layouts

Permiten mantener homogeneas las vistas de una aplicación, sin tener que escribir y mantener todo el codigo en todas las vistas, tambien se las conoce como paginas

maestras.

La idea es definir una cshtml con el contenido que se repetira en todas las paginas, dejando huecos que se rellenarán en cada pagina con contenido diferente, estos huecos son tambien conocidos como PlaceHolder.

Un ejemplo de Layout podría ser

```
<!DOCTYPE html>
<html>
    <head><title>@ViewBag.Title</title></head>
    <body>
        <h1>@ViewBag.Title</h1>
        <div id="main-content">@RenderBody()</div>
        <footer>@RenderSection("Footer")</footer>
    </body>
</html>
```

En el ejemplo anterior, los **PlaceHolder**, se definen con las expresiones Razor

- @RenderBody(): Sera el contenido principal de la página.
- @RenderSection(): Será un contenido especialmente marcado con un Identificador.

Y una página que emplee dicho Layout podría ser

```
@Code
    Layout = "~/Views/Shared/_Layout.vbhtml"
End Code

<!-- Aqui empieza el bloque considerado principal -->
<p>This is the main content!</p>
<!-- Aqui termina el bloque considerado principal -->

<!-- Aqui empieza la seccion Footer -->

@section Footer
    This is the <strong>footer</strong>.
End section
```

Como se puede ver, hay que indicar al comienzo de la página en un bloque de codigo, la propiedad Layout, esto para cada vista puede ser un incordio, por eso se

puede definir el fichero _ViewStart.cshtml dentro del directorio Views, e indicar alli el Layout por defecto.

```
@Code
    Layout = "~/Views/Shared/_Layout.vbhtml"
End Code
```

6.5. Vistas parciales

Las Vistas partiales son un tipo de Vista especial pensada en devolver un trozo de HMTL, y no un HTML completo, esto es no aplicar el Layout.

```
Public Class HomeController
    Inherits System.Web.Mvc.Controller

    Function Index() As ActionResult
        return PartialView()
    End Function
End Class
```

De echo, es lo que hace en parte, dado que no muestra el Layout que se este estableciendo con _**ViewStart.vbhtml** como se haria con las **View**, salvo que se indicase explicitamente en el propio *.vbhtml, en este caso sí se aplicaria el Layout.

6.6. Objeto ViewModel

Además de pasar información con los Diccionarios **ViewData** y **ViewBag**, tambien se puede pasar como **ViewModel**, esto es, pasar el objeto con la información al método View().

```
Function Parcial() As ActionResult
    Dim personaje As Personaje = New Personaje("Hommer")
    Return View(personaje)
End Function
```

Una vez pasado el objeto de Modelo a la Vista, falta recogerlo en la Vista y precesarlo, para ello recurrimos a Razor

```
<h2>@Model.Nombre</h2>
```

El modelo tambien podría ser una coleccion

```
Dim personajes As IList = New List(Of Personaje) From {
            New Personaje("Hommer"), New Personaje("Marge")}
Return View(personajes)
```

Y se procesaría en la vista

```
@For Each personaje As Personaje In Model
    @<li>
        @personaje.Nombre
    </li>
Next
```

NOTE Antes de procesar el Modelo, hay que asegurarse de que no es null

7. Helpers

Son como su nombre indica Ayudantes, que facilitan la labor de crear determinados trozos de código en las Vistas.

7.1. Helpers básicos

7.1.1. HTML Helpers

Serán accesible a traves de la propiedad Html de la Vista, esta propiedad es de tipo System.Web.Mvc.HtmlHelper<T>, donde T representa al Modelo.

- Html.BeginForm(): Permite incluir la etiqueta <form> de HTML, tiene su contrapuesto Html.EndForm(), que permite cerrar la etiqueta </form>

Apoyandose en que el objeto que retorna la invocación de Html.BeginForm() implementa la interface IDisposable, se puede reducir un poco más el anterior codigo empleando la directiva using, que se encargará de invocar al método Dispose y este de introducir la etiqueta de fin del </form>.

```
@Using Html.BeginForm("Personajes", "Buscar", FormMethod.Get)
    @<input type = "text" name="Nombre" />
    @<input type = "submit" value="Buscar" />
End Using
```

El resultado de este código, sera una etiqueta de <form>, donde en la propiedad action, se establecerá la URL necesaria para llegar al método Buscar, del controlador Personajes y en la propiedad method se pondra GET.

```
<form action="/Personajes/Buscar" method="get">
    <input type="text" name="Nombre" />
    <input type="submit" value="Buscar" />
</form>
```

- Html.ValidationSummary(): Representa con una lista desordenada HTML, todos los elementos (errores) que se encuentre en el Dicionario ModelState.

```
@Html.ValidationSummary(True)
```

Siendo el resultado final para la anterior etiqueta

```
<div class="validation-summary-errors">
    <ul>
        <li>Esta todo mal</li>
    </ul>
</div>
```

Suponiendo que en el controlador se han incluido errores empleando ModelState

```
ModelState.AddModelError("", "Esta todo mal")
ModelState.AddModelError("Nombre", "El Nombre no es correcto")
```

NOTE

La propiedad excludePropertyErrors, indica que solo se representen los errores asociados al Modelo en general, y no a propiedades concretas, como en este caso **Nombre**

- Html.TextBox(): Muestra un elemento HTML de entrada de datos simple

```
@Html.TextBox("Nombre", Model.Nombre)
```

Siendo el resultado final para la anterior etiqueta

```
<input id="Nombre" name="Nombre" type="text" value="<Valor que tenga la
propiedad Nombre del Modelo actual>" />
```

NOTE

Los campos que precisan emplear el Modelo, necesitan validar que existe.

```
@If (Model <> Nothing) Then

End If
```

- Html.TextArea(): Muestra un elemento HTML de entrada de datos multilinea

```
@Html.TextArea("Descripcion", Model.Descripcion)
```

Siendo el resultado final para la anterior etiqueta

```
<textarea cols="20" id="text" name="Descripcion" rows="2">valor que
tenga en Modelo actual en su capo Descripcion</textarea>
```

- Html.Label(): Muestra un elemento HTML de etiqueta de un campo de entrada

```
@Html.Label("Nombre")
```

Siendo el resultado final para la anterior etiqueta

```
<label for="Nombre">Nombre</label>
```

NOTE

Si se desea que la propiedad del Modelo, no se use para pintar la Label, se puede anotar dicha propiedad con DisplayName y cambiar así el valor que usará Label

```
<DisplayName("Nombre completo")>
Public Property Nombre() As String
    Get
        Return _Nombre
    End Get

    Set(Nombre As String)
        Me._Nombre = Nombre
    End Set
End Property
```

En este caso el resultado HTML seria

```
<label for="Nombre">Nombre completo</label>
```

- Html.DropDownList(): Muestra un elemento HTML de seleccion simple.

```
@Html.DropDownList("Genero")
```

Donde se tiene que cumplir que exista un objeto de tipo

IEnumerable<SelectListItem> en ViewBag, de nombre Genero, siendo ese nombre el mismo que el del campo del Modelo que posteriormente se envie al controlador.

```
Dim items As IList(Of SelectListItem) = New List(Of SelectListItem)()
items.Add(New SelectListItem With {.Text = "Masculino", .Value = "0",
.Selected = True})
items.Add(New SelectListItem With {.Text = "Femenino", .Value = "1"})
ViewBag.Genero = items
```

Siendo el resultado final para la anterior etiqueta

```
<select id="Genero" name="Genero">
    <option selected="selected" value="0">Masculino</option>
    <option value="1">Femenino</option>
</select>
```

- Html.ListBox(): Muestra un elemento HTML de seleccion multiple.

```
@Html.ListBox("Ciudades", ViewBag.Ciudades)
```

Donde se tiene que cumplir que exista un objeto de tipo MultiSelectList en ViewBag, de nombre Ciudades

```
Dim Ciudades As IList = {
    New With {.Id = 0, .Nombre = "Madrid"},
    New With {.Id = 1, .Nombre = "Barcelona"}
}

Dim Seleccionados As IList = {
    1
}

ViewBag.Ciudades = New MultiSelectList(Ciudades, "Id", "Nombre",
Seleccionados)
```

Siendo el resultado final para la anterior etiqueta

```
<select id="Ciudades" multiple="multiple" name="Ciudades">
    <option value="0">Madrid</option>
    <option selected="selected" value="1">Barcelona</option>
</select>
```

- Html.CheckBox(): Muestra un elemento HTML de tipo checkbox, se aplica sobre propiedades booleanas.

```
@Html.CheckBox("Acepto", true)
```

Siendo el resultado final para la anterior etiqueta

```
<input checked="checked" id="Acepto" name="Acepto" type="checkbox"
value="true" />
```

- Html.RadioButton(): Muestra un elemento HTML RadioButton

```
@Html.RadioButton("Genero","Masculino")
@Html.RadioButton("Genero","Femenino")
```

Siendo el resultado final para la anterior etiqueta

```
<input checked="checked" id="Genero" name="Genero" type="radio" value=
"Masculino" />
<input checked="checked" id="Genero" name="Genero" type="radio" value=
"Femenino" />
```

- Html.ActionLink(): Genera un elemento de HTML <a>

```
@Html.ActionLink("pulsame", "Index", "TiendaDeMusica")
```

Siendo el resultado final para la anterior etiqueta

```
<a href="/TiendaDeMusica">pulsame</a>
```

7.1.2. URL Helpers

- Url.Action(): Genera un URL partiendo de un método de acción y un controlador

```
@Url.Action("Index", "TiendaDeMusica")
```

Siendo el resultado final para la anterior etiqueta **/TiendaDeMusica/Index**

- Url.Content(): Genera una URL de un recurso de la aplicación

```
<a href="@Url.Content("~/Content/Site.css")" >ver fichero de
estilos</a>
```

Siendo el resultado final para la anterior etiqueta **/Content/Site.css**

- Url.Encode(): Codifica una URL

```
@Url.Encode("http://www.google.es?script=<script></script>")
```

Siendo el resultado final para la anterior etiqueta <a href="http://www.google.es?script=<script></script&gt" class="bare">http://www.google.es?script=<script></script&gt;

7.1.3. Ajax Helpers

- @Ajax.JavaScriptStringEncode(): Evita la decodificación de caracteres reservados, evitando así ataques por XSS.
- Ajax.ActionLink(): Permite definir un elemento HTML <a>, que en vez de realizar una navegación, realiza una petición en background, con tecnologia Ajax, permitiendo insertar o reempleazar un determinado nodo del arbol DOM con el resultado obtenido con la petición.

```
@Ajax.ActionLink(
    "Ajax", 'Texto del enlace
    "Index", 'Método de Acción
    "TiendaDeMusica", 'Controlador
    new AjaxOptions With
    {
        .UpdateTargetId = "actualizado", 'Id del nodo DOM sobre el que
se aplicará el nuevo contenido
        .InsertionMode = InsertionMode.Replace, 'Acción a realizar
sobre el nodo DOM con los datos obtenidos
        .HttpMethod = "GET" 'Method HTTP
    }
)
```

Para que este Helper funcione, hay que realizar varias cosas, la primera es añadir/comprobar que se tiene instalado el paquete jquery.unobtrusive-ajax en el proyecto, esto se hace con NuGet, la segunda es incluir el js en nuestra página, lo podemos hacer así

```
@section scripts{
    @Scripts.Render("~/Scripts/jquery.unobtrusive-ajax.min.js")
}
End section
```

Dado que en el Layout por defecto, se ha creado una sección scripts, donde en principio se deberian de añadir todos los **js**

Por último para que funcione debe existir el nodo con id actualizado

```
<div id="actualizado"> Este texto desaparecera </div>
```

- @Ajax.BeginForm(): Permite configurar un formulario HTML, que en vez de realizar una navegación, realiza una petición en background, con tecnologia Ajax, permitiendo insertar o reempleazar un determinado nodo del arbol DOM con el resultado obtenido con la petición.

```
@Using (Ajax.BeginForm(
        "Ajax", 'Método de Acción
        "Home", 'Controlador
        New AjaxOptions With {
            .UpdateTargetId = "actualizado", 'Id del nodo DOM sobre el
que se aplicará el nuevo contenido
            .InsertionMode = InsertionMode.Replace,
            .HttpMethod = "GET"
        }
))

    @<input type="submit" value="Enviar" />

End Using
```

Funciona de forma analoga al anterior, con las mismas necesidades.

7.2. Helpers Tipados

Están asociados a la propiedad Model de la Vista, permitiendo asociar a propiedades del Modelo, campos del formulario HTML, por lo que al tener que acceder a las propiedades del Modelo, necesitan conocer su tipo, por lo tanto lo primer es establecerlo

```
@ModelType Personaje
```

- Html.HiddenFor(): Muestra un campo oculto HTML asociado a un atributo del modelo.

```
@Html.HiddenFor(Function(m) m.Nombre)
```

- Html.LabelFor(): Muestra un elemento HTML de etiqueta de un campo de entrada

```
@Html.LabelFor(Function(m) m.Nombre, "Nombre", New With { .@class =
"control-label col-md-2" })
```

- Html.ValidationMessageFor(): Muestra los errores de validación asociados a un atributo del Modelo

```
@Html.ValidationMessageFor(Function(m) m.Nombre)
```

- Html.TextBoxFor(): Muetra un campo de entrada HTML asociado a un atributo del Modelo

```
@Html.TextBoxFor(Function(m) m.Nombre)
```

- @Html.DropDownListFor(): Muestra una lista de seleccion simple

```
@Html.DropDownListFor(Function(m) m.Nombre, ViewBag.Generos as
SelectList)
```

- Html.EditorFor(): Permite seleccionar con anotaciones en los campos del modelo el tipo de HTML a renderizar

```
@Html.EditorFor(Function(m) m.Nombre)
```

Se puede elegir en la clase del Modelo, que tipo de representación se desea para el campo Nombre, empleando la anotacion DataType

```
<DataType(DataType.Text)>
Public Property Nombre() As String
    Get
        Return _Nombre
    End Get

    Set(Nombre As String)
        Me._Nombre = Nombre
    End Set
End Property
```

7.3. Helpers Personalizados

Para crear un nuevo **Helper**, basta con definir un método de extensión de la clase correspondiente, por ejemplo si se desea crear un HtmlHelper, basta con crear un método de extensión de la clase **System.Web.Mvc.HtmlHelper**

```
Public Module LabelExtensions
     <Extension()> _
     Public Function SimpleLabel(ByVal helper As HtmlHelper, ByVal
target As String, ByVal text As String) As MvcHtmlString
          Return new MvcHtmlString(String.Format("<label for='{0}'>
{1}</label>", target, text))
     End Function
End Module
```

Normalmente, el retorno del método sera de tipo **System.Web.Mvc.MvcHtmlString**, ya que aunque es un tipo que encapsula un String, la ventaja es que lo codifica para evitar XSS, aunque tambien se puede emplear String

Para invocar esté nuevo Helper

```
@Html.SimpleLabel("target", "text")
```

Si se precisa, a traves del objeto HtmlHelper, se puede obtener el acceso a los objetos

- Model
- ViewData
- ViewBag

```
helper.ViewData.Model
helper.ViewData
helper.ViewBag
```

7.4. Helpers Declarativos

Permiten definir un Helper directamente en la vista, se emplea **@helper**

```
@helper TextoResaltado(ByVal text As String)
    @If String.IsNullOrWhiteSpace(text) Then
        @<h2>Default name</h2>
    Else
        @<h2>@text</h2>
    End If
End helper

@TextoResaltado("Titulo")
```

8. Validaciones

8.1. Anotaciones

El API ofrece algunas anotaciones para incluir en las clases de Modelo, para indicar las restricciones que han de cumplir los atributos de dichos Modelos, estas anotaciones estan en el namespace **System.ComponentModel.DataAnnotations**

- Required: Indica que el campo es obligatorio

```
<Required>
Public Property Titulo As String
    Get
        Return _Titulo
    End Get
    Set(value As String)
        _Titulo = value
    End Set
End Property
```

- StringLength: Máximo y minimo tamaño del campo String.

```
<StringLength(150, MinimumLength:=10)>
Public Property Titulo As String
    Get
        Return _Titulo
    End Get
    Set(value As String)
        _Titulo = value
    End Set
End Property
```

- RegularExpression: Permite definir una expresión regular que deberá cumplir el campo

```
<RegularExpression("[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}")>
Public Property Email As String
    Get
        Return _Email
    End Get
    Set(value As String)
        _Email = value
    End Set
End Property
```

- Range: Rango de valores entre los que se ha de encontrar el campo numerico.

```
<Range(35,44)>
Public Property Id As Long
    Get
        Return _Id
    End Get
    Set(value As Long)
        _Id = value
    End Set
End Property
```

- Compare: Permite asegurarse que dos campos son iguales

```
<Compare("Autor")>
Public Property ConfirmacionAutor As String
    Get
        Return _ConfirmacionAutor
    End Get
    Set(value As String)
        _ConfirmacionAutor = value
    End Set
End Property
```

- Remote: Permite indicar un método de acción que se encargará de realizar una validación previa a su procesamiento

```
<Remote("ValidarAutor", "Libro")>
Public Property ConfirmacionAutor As String
    Get
        Return _ConfirmacionAutor
    End Get
    Set(value As String)
        _ConfirmacionAutor = value
    End Set
End Property
```

NOTE	Deberá existir el método de acción **ValidarAutor** en un Controlador **LibroController**

8.1.1. Mensajes personalizados

Estos validadores, tiene asociados unos mensajes por defecto, que pueden no ser los deseados, pero es posible cambiarlos con la propiedad ErrorMessage de las anotaciones.

```
<RegularExpression("[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}",
ErrorMessage:="Mensaje personalizado para Autor")>
Public Property Autor As String
    Get
        Return _Autor
    End Get
    Set(value As String)
        _Autor = value
    End Set
End Property
```

8.2. Helpers

- Html.ValidationSummary(): Representa con una lista desordenada HTML, todos los elementos (errores) que se encuentre en el Dicionario ModelState.

```
@Html.ValidationSummary(True)
```

Siendo el resultado final para la anterior etiqueta

```
<div class="validation-summary-errors">
    <ul>
        <li>Esta todo mal</li>
    </ul>
</div>
```

Suponiendo que en el controlador se han incluido errores a ModelState

```
ModelState.AddModelError("", "Esta todo mal")
ModelState.AddModelError("Nombre", "El Nombre no es correcto")
```

NOTE La propiedad excludePropertyErrors, indica que solo se representen los errores asociados al Modelo en general, y no a propiedades concretas, como en este caso Nombre

- Html.ValidationMessageFor(): Muestra los errores de validación asociados a un atributo del Modelo

```
@Html.ValidationMessageFor(Function(m) m.Nombre)
```

8.3. ModelState

El método de acción, es capaz de conocer el estado de la validación del Modelo a traves del campo ModelState de la clase Controller.

```
Function Create() As ActionResult
    If ModelState.IsValid Then

    End If
    Return View()
End Function
```

Si se desea incluir algún error de validación en el sistema, se puede añadir facilmente con

```
ModelState.AddModelError("", "The user name or password provided is
incorrect.")
```

8.4. IValidatableObject

Es una interface que se puede emplear para definir la validación de un tipo dentro de su definición.

Es una alternativa a la validación con anotaciones.

```
Public Class Libro
    Implements IValidatableObject
        Public Function Validate(validationContext As
ValidationContext) As IEnumerable(Of ValidationResult) Implements
IValidatableObject.Validate
            Throw New NotImplementedException()

                If Titulo <> Nothing And Titulo.Split(" ").Length > 2
Then
                    Return New ValidationResult("El nombre tiene muchas
palabras!", {"Nombre"})
                End If
        End Function
End Class
```

8.5. Validación en el lado del cliente

Es la validación de los datos antes de ser enviados, es la validación mas deseable desde el punto de vista del usuario de la aplicación, dado que ofrece una respuesta rapida, no llega a haber intercambio entre cliente y servidor y por tanto no hay tiempo de latencia.

Pero no puede ser la unica forma de validación, ya que el servidor no puede estar seguro de que lo que le llega ha sido validado antes por la capa del cliente, ya que puede haber sido interceptado o incluso en e mismo cliente ha podido ser puenteada la validación, por lo que deberian existir ambas.

En el cliente las validaciones se harán con javascript, en este caso volviendo a JQuery, se empleará la libreria jquery.validate, esta libreria ya esta incluida en el Layout, por lo que no hay que hacer nada al respecto, unicamente, asegurarse de que el **Web.config**, esta habilitada la caracteristica **ClientValidationEnabled**

```
<configuration>
    <appSettings>
        <add key="ClientValidationEnabled" value="true" />
    </appSettings>
</configuration>
```