



# Desarrollo de Servicio Web I

---



# ÍNDICE

Presentación

Red de contenidos

## **Unidad I: INTRODUCCION AL DESARROLLO WEB**

1.1 Tema 1 : Introducción a ASP.NET MVC	9
1.2 Tema 2 : Arquitectura de ASP.NET MVC	31

## **Unidad II: TRABAJANDO CON DATOS EN ASP.NET MVC**

2.1 Tema 3 : Interacción con el modelo de datos	63
2.2 Tema 4 : Manejo de Vistas	103
2.3 Tema 5 : Arquitectura “N” capas orientadas al Dominio	129

## **Unidad III: IMPLEMENTANDO UNA APLICACIÓN E-COMMERCE**

3.1 Tema 6 : Implementando una aplicación e-commerce	165
--	-----

## **Unidad IV: CONSUMO DE SERVICIOS**

4.1 Tema 7 : Implementacion y consumo de servicios WCF	189
4.2 Tema 8 : Implementacion consumo de servicios Web API	223

## **Unidad V: PATRONES DE DISEÑO CON ASP.NET MVC**

5.1 Tema 9 : Inversion of Control	243
-----------------------------------	-----

## APENDICE

A : Jquery y Ajax	262
-------------------	-----



## PRESENTACIÓN

Visual Studio 2015 y su plataforma .NET FrameWork 4.5.2 permite implementar desarrollos de software de manera rápida y robusta. ASP .NET, tanto en Web Form como en MVC, admiten crear aplicaciones en tiempo mínimo bajo una plataforma de librerías del .NET Framework. De esta manera, la aplicación puede desarrollarse para entornos web y, luego, tener la posibilidad de emplear cualquier dispositivo como cliente (Smartphone, Tablets, etc.) con muy poca modificación.

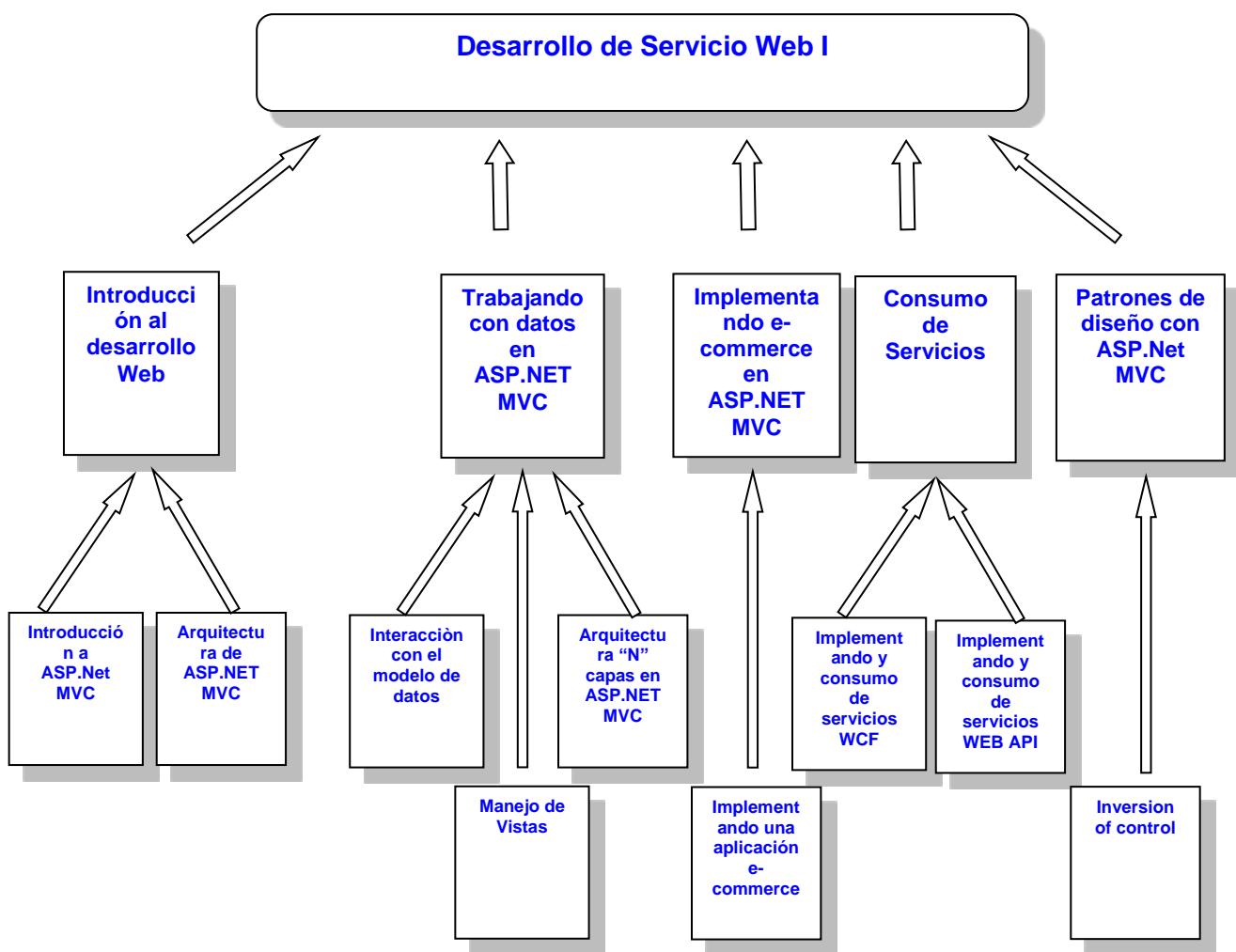
El curso de **Desarrollo de Servicio Web I** pertenece a la Escuela de Tecnología de Información y se dicta en las carreras de Tecnología de la institución. Este curso brinda un conjunto de herramientas de programación para trabajar en aplicaciones web, en función al diseño de páginas web y con un origen de datos que permita al alumno realizar, en forma eficiente, operaciones de consulta y actualización de datos bajo el entorno web.

El manual para este curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que desarrollamos durante semanas determinadas. En cada una de ellas, el alumno hallará los logros que se deberá alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos que debe desarrollar. Por último, encontrará las actividades y trabajos prácticos que deberá desarrollar en cada sesión, los que le permitirá reforzar lo aprendido en la clase.

El curso es eminentemente práctico: consiste en diseño y programación de aplicaciones web con base de datos utilizando ADO .NET Entity Framework. La primera parte de este manual enseña a familiarizarse con el entorno de desarrollo de una aplicación Web MVC: diseño del modelo, uso del Entity Framework Code First, generación de mantenimientos con el scaffolding, todo ello mediante ejemplos didácticos. Luego, se desarrollará el tema de las vistas y controladores, de manera más detallada, implementando una solución ecommerce. Finalmente se implementan mejoras a la solución dando una mayor interactividad del lado del cliente con Jquery, Ajax y adopción de patrones de software considerados Best Practices.

Este manual reviste importancia para una capacitación en conocimientos generales en el manejo de aplicaciones web, utilizando la tecnología de punta que se implementa actualmente en el mercado.

## RED DE CONTENIDOS





# INTRODUCCION AL DESARROLLO WEB

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla interfaces de usuario para una aplicación Web utilizando el patrón de diseño MVC.

## TEMARIO

### Tema 1: Introducción a ASP.NET MVC (1 horas)

1. Introducción a ASP.NET MVC
2. Patrón MVC: Modelo, Vista y Controlador

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando en patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



## 1. ASP.NET MVC

### 1.1 Introducción a ASP.NET MVC

ASP.NET MVC es una implementación reciente de la arquitectura Modelo-Vista-Controlador sobre la base ya existente del Framework ASP.NET otorgándonos de esta manera un sin fin de funciones que son parte del ecosistema del Framework .NET. Además que nos permite el uso de lenguajes de programación robustos como C#, Visual Basic .NET.

ASP.NET MVC nace como una opción para hacer frente al ya consagrado y alabado Ruby on Rails un framework que procura hacer uso de buenas prácticas de programación como la integración de Unit tests o la separación clara de ocupaciones, dándonos casi todos los beneficios otorgados por Ruby on Rails y sumando el gran y prolífico arsenal proporcionado por .NET.

Entre las características más destacables de ASP.NET MVC tenemos las siguientes:

- Uso del patrón Modelo-Vista-Controlador.
- Facilidad para el uso de Unit Tests.
- Uso correcto de estándares Web y REST.
- Sistema eficiente de routing de links.
- Control a fondo del HTML generado.
- Uso de las mejores partes de ASP.NET.
- Es Open Source.

La siguiente figura muestra los principales componentes de su arquitectura:

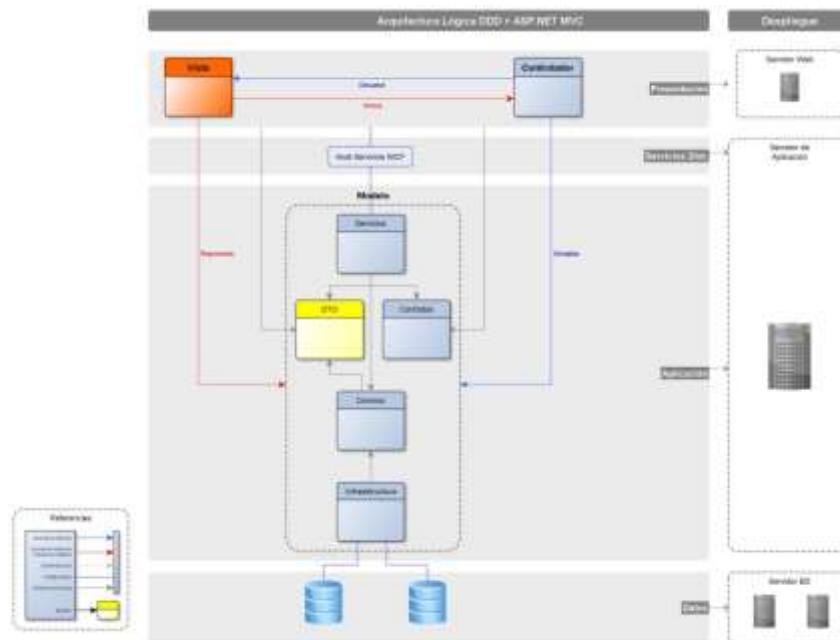


Figura: 1

Ref: <https://jestruch.wordpress.com/2012/02/21/arquitectura-ddd-domain-driven-design-asp-net-mvc/>

### ¿ASP.NET MVC es mejor que ASP.NET Web Form?

Esta pregunta se responde fácilmente, ASP.NET MVC lo deberíamos usar cuando tengamos que hacer un Software que sea de gran envergadura y en donde la mantenibilidad y escalabilidad sean factores primordiales, en contraste deberíamos de

usar ASP.NET web form cuando hagamos aplicaciones simples donde el factor primordial sea el tiempo.

El marco ASP.NET MVC ofrece las siguientes ventajas:

- Esto hace que sea más fácil de gestionar la complejidad de dividir una aplicación en el modelo, la vista y el controlador.
- No utiliza el estado de vista o formas basadas en servidor. Esto hace que el marco idóneo MVC para los desarrolladores que quieren un control total sobre el comportamiento de una aplicación.
- Utiliza un patrón Front Controller que procesa las solicitudes de aplicaciones web a través de un solo controlador. Esto le permite diseñar una aplicación que es compatible con una rica infraestructura de enrutamiento.
- Proporciona un mejor soporte para el desarrollo guiado por pruebas (TDD).
- Funciona bien para las aplicaciones web que son apoyados por grandes equipos de desarrolladores y diseñadores web que necesitan un alto grado de control sobre el comportamiento de la aplicación.

El marco de trabajo basado en formularios Web ofrece las siguientes ventajas:

- Es compatible con un modelo de eventos que conserva el estado a través de HTTP, lo que beneficia el desarrollo de aplicaciones Web de línea de negocio. La aplicación basada en formularios Web ofrece decenas de eventos que se admiten en cientos de controles de servidor.
- Utiliza un patrón Controlador Página que añade funcionalidad a las páginas individuales.
- Utiliza el estado de vista sobre las formas basadas en servidor, que puede hacer la gestión de la información de estado más fácil.
- Funciona bien para pequeños equipos de desarrolladores web y diseñadores que quieren aprovechar el gran número de componentes disponibles para el desarrollo rápido de aplicaciones.
- En general, es menos complejo para el desarrollo de aplicaciones, ya que los componentes (la clase de página, controles, etc.) son fuertemente integrado y por lo general requieren menos código que el modelo MVC.

## 1.2 Patrón MVC (Modelo Vista Controlador)

El Modelo-Vista-Controlador (MVC) es un patrón arquitectónico que separa una aplicación en tres componentes principales: el modelo, la vista y el controlador. El marco ASP.NET MVC proporciona una alternativa al modelo de formularios Web Forms ASP.NET para crear aplicaciones Web.

ASP.NET MVC es un marco de presentación de peso ligero, altamente comprobable de que (al igual que con las aplicaciones basadas en formularios web) se integra con las características ASP.NET existentes, como páginas maestras y autenticación basada en membresía. El framework MVC se define en la asamblea System.Web.Mvc.

Patrón de diseño MVC

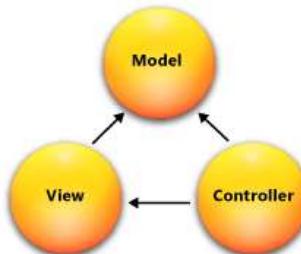


Figura: 2

Referencia: [https://msdn.microsoft.com/en-us/library/dd381412\(v=vs.108\).aspx](https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx)

Modelo

Contiene el núcleo de la funcionalidad (dominio) de la aplicación.

Encapsula el estado de la aplicación.

No sabe nada / independiente del Controlador y la Vista.

Vista

Es la presentación del Modelo.

Puede acceder al Modelo pero nunca cambiar su estado.

Puede ser notificada cuando hay un cambio de estado en el Modelo.

Controlador

Reacciona a la petición del Cliente, ejecutando la acción adecuada y creando el modelo pertinente

Es importante mencionar que el patrón MVC no es exclusivo para el diseño Web, en sus inicios fue muy utilizado para el desarrollo de interfaces gráficas de usuario (GUI), por otro lado tampoco es una implementación propietaria de alguna empresa tecnológica, sea Microsoft, Oracle o IBM.

MVC está implementando por muchas herramientas tales como:

- Ruby
- Java
- Perl
- PHP
- Python
- .NET

La siguiente figura muestra la idea grafica del patrón MVC para el entorno de la Web.

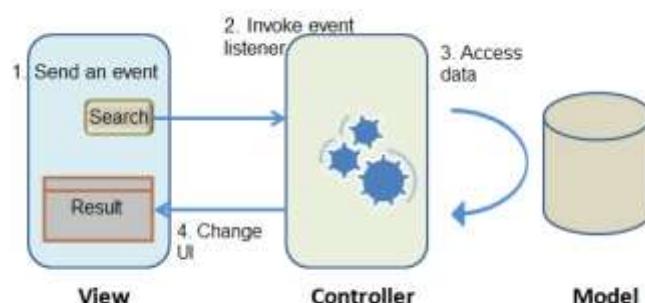


Figura: 3  
Referencia: <http://www.adictosaltrabajo.com/tutoriales/zk-mvc/>

### Características de ASP.NET MVC

A continuación mencionamos las siguientes características de este patrón:

- Soporte para la creación de aplicaciones para Facebook.
- Soporte para proveedores de autenticación a través del OAuth Providers.
- Plantillas por default renovadas, con un estilo mejorado.
- Mejoras en el soporte para el patrón Inversion Of Control e integración con Unity
- Mejoras en el ASP.NET Web Api, para dar soporte a las implementaciones basadas en RESTful
- Validaciones en lado del modelo
- Uso de controladores Asíncronos
- Soporte para el desarrollo de aplicaciones Web Móvil, totalmente compatible con los navegadores de los modernos SmartPhone (Windows Phone, Apple y Android), etc.

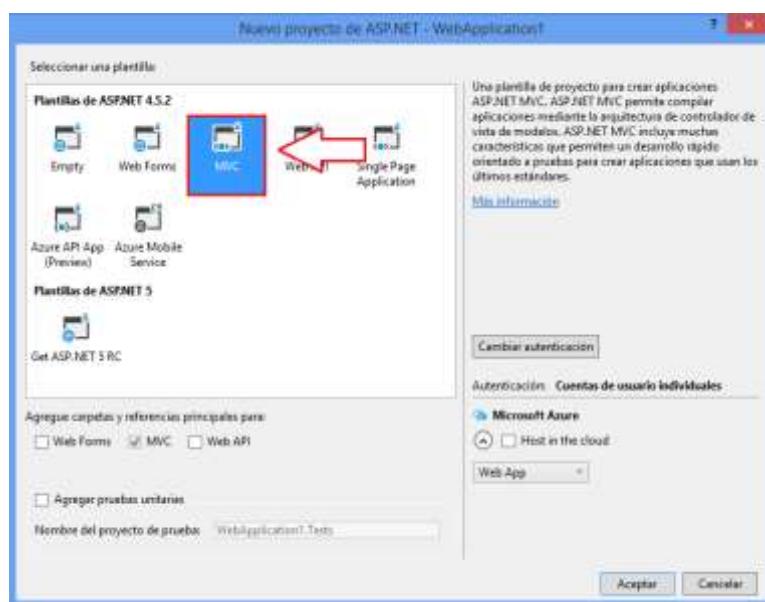
## Estructura de una aplicación ASP.NET MVC

Para crear una aplicación con ASP.NET MVC, abrimos Visual Studio y seleccionamos “Nuevo Proyecto”.

Selecciona la plantilla Aplicación web ASP.NET MVC, tal como se muestra. Asigne el nombre al proyecto



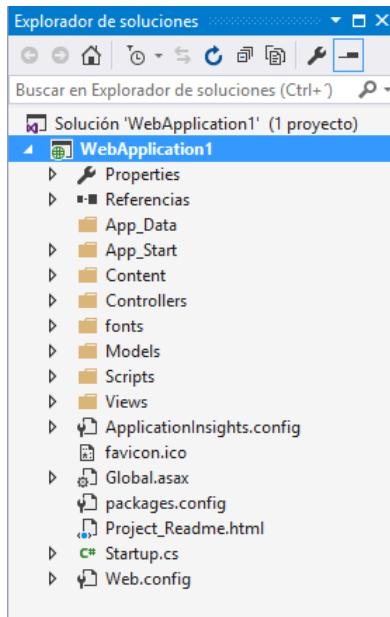
A continuación seleccionamos la plantilla del proyecto, el cual será de tipo MVC



Al seleccionar dicha plantilla, las carpetas y referencias principales serán de tipo MVC.  
Al terminar con el proceso hacer click en la opción ACEPTAR.

## Estructura de directorios y archivos.

Cuando creamos una nueva aplicación con ASP.NET MVC se crea por defecto una estructura de directorios, apropiada para la gran mayoría de las aplicaciones.

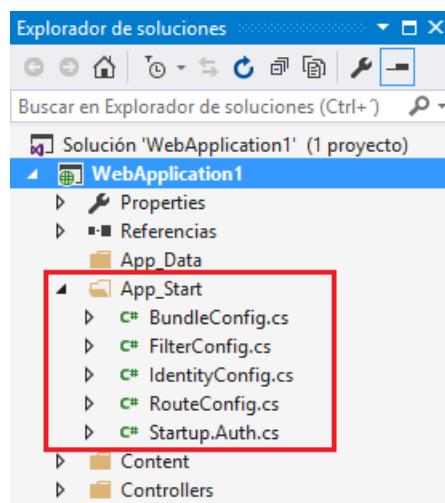


### Directorio App\_Data.

Este directorio está pensado para ubicar archivos de datos, normalmente bases de datos MSSQL. También es el lugar adecuado para archivos XML o cualquier otra fuente de datos. Inicialmente está vacío.

### Directorio App\_Start.

Este directorio, contiene los archivos de código que se ejecutan al inicializar la aplicación. Toda aplicación ASP.NET MVC es una instancia derivada de la clase System.Web.HttpApplication, definida en el archivo global.asax. Esta clase es la encargada de iniciar la aplicación, el directorio App\_Start está pensando para ubicar las clases de configuración para el inicio de la aplicación. La siguiente imagen muestra el contenido del directorio.

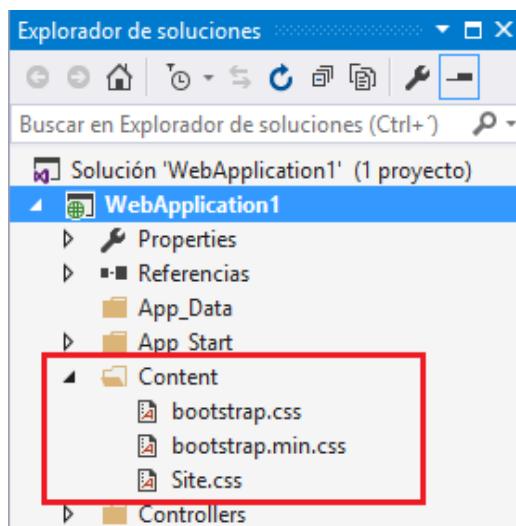


Por defecto contiene los siguientes archivos (clases):

- AuthConfig.cs
- BundleConfig.cs
- FilterConfig.cs
- RouteConfig.cs
- WebApiConfig.cs

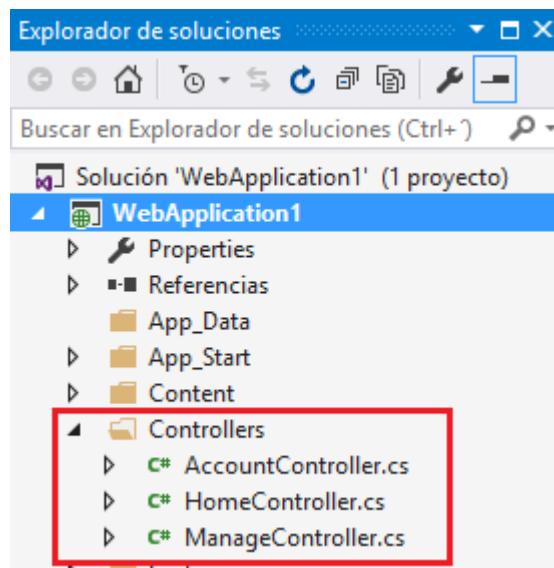
#### **Directorio Content.**

El directorio Content está pensado para el contenido estático de la aplicación, especialmente útil para archivos css e imágenes asociadas. ASP.NET MVC nos ofrece por defecto una organización en base a “temas”, que nos permite personalizar el aspecto visual de nuestra aplicación de forma fácil y rápida.



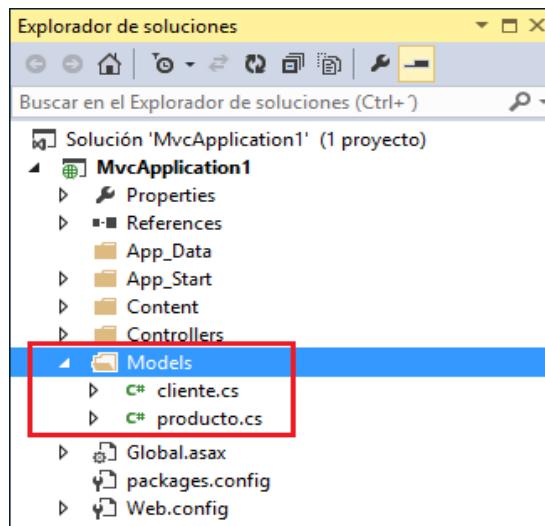
#### **Directorio Controllers.**

El directorio controllers es el lugar para los controladores. Los controladores son las clases encargadas de recibir y gestionar las peticiones http de la aplicación.



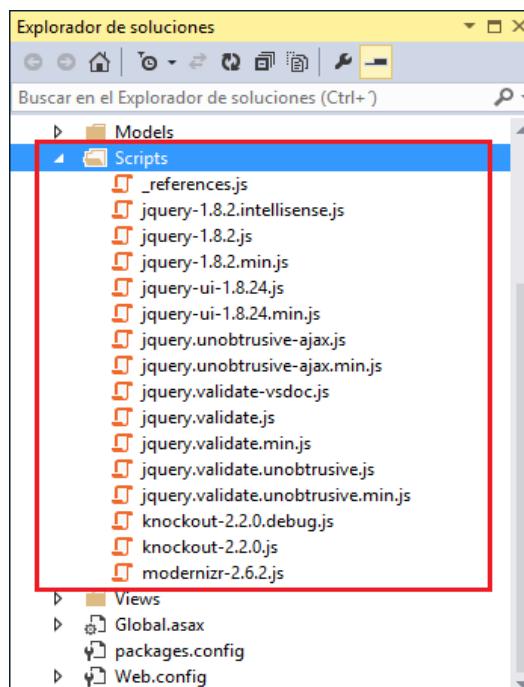
### Directorio Models

El directorio models es la ubicación que nos propone ASP.NET MVC para las clases que representan el modelo de la aplicación, los datos que gestiona nuestra aplicación.



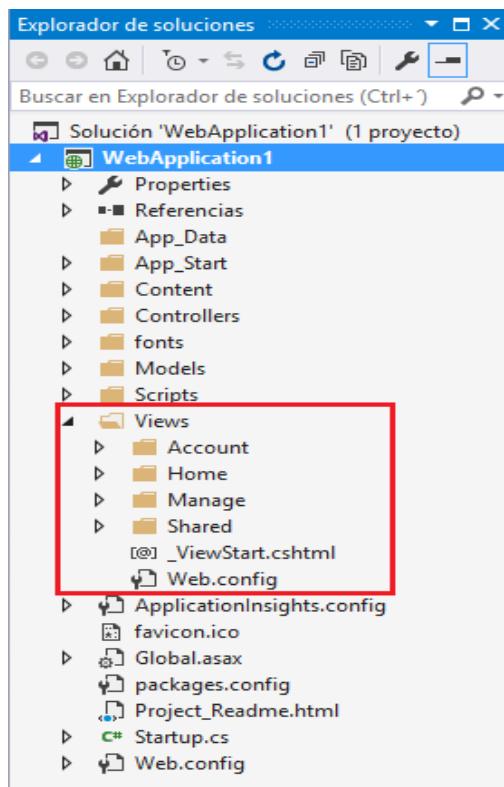
### Directorio Scripts

El directorio scripts está pensado para ubicar los archivos de javascript (\*.js). El código javascript es ejecutado en el contexto del navegador, es decir, en la parte cliente, y nos permite ejecutar acciones sin necesidad de enviar los datos al servidor.



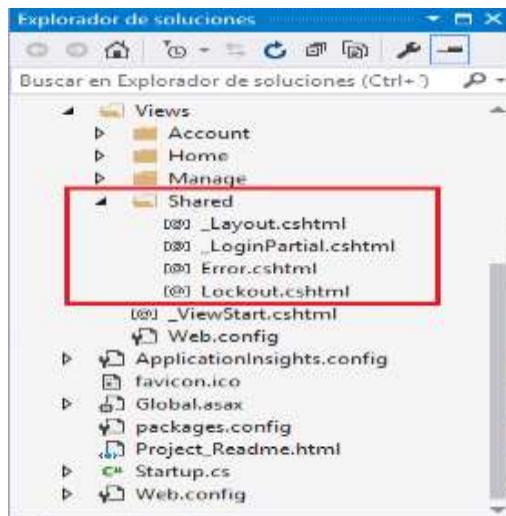
### Directorio Views

El directorio Views contiene los archivos de la vista. Los controladores devuelven vistas sobre las que imprimimos el modelo de nuestra aplicación. Estas vistas son interpretadas por el motor de renderización – Razor en nuestro caso.



### Directorio Shared

Contiene las vistas que van a ser reutilizadas en otras vistas, se incluye vistas parciales. Es muy importante respetar la ubicación de los archivos, ya que cuando desde una vista hagamos la llamada @Html.Partial("Error") para incluir la vista de la pantalla de error, el motor buscará en el directorio Shared para encontrar la vista Error.cshtml.



### Archivo \_ViewStart.cshtml

Este archivo establece el layout por defecto de las páginas. El contenido del archivo es sencillo y únicamente especifica el archivo de layout.

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

El layout es un archivo con extensión .cshtml que contiene la estructura general de documento, que es reutilizada en el resto de vistas. De este modo evitamos tener que reescribir el código en todas las vistas, reutilizando el código y permitiendo que este sea mucho más sencillo de mantener.

### Archivo \_Layout.cshtml

El archivo \_Layout.cshtml define la capa de la aplicación, que contiene la estructura general de documento, que es reutilizada en el resto de vistas.

El archivo \_Layout.cshtml se encuentra dentro del directorio Views/Shared. El contenido del archivo layout por defecto se muestra a continuación:

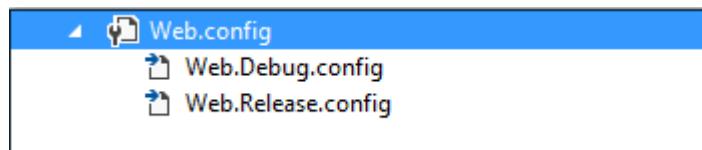
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
  </head>
  <body>
    @RenderBody()

    @Scripts.Render("~/bundles/jquery")
    @RenderSection("scripts", required: false)
  </body>
</html>
```

Fijémonos en la llamada que hace Razor al método @RenderBody(), es ahí donde se procesará la vista que estemos mostrando. Podemos tener múltiples archivos de layout dentro de nuestro proyecto.

### El archivo web.config

El archivo web.config es el archivo principal de configuración de ASP.NET. Se trata de un archivo XML donde se define la configuración de la aplicación. Veremos poco a poco el contenido de este fichero, aunque vamos a ver aquí algunas características generales que es necesario conocer.

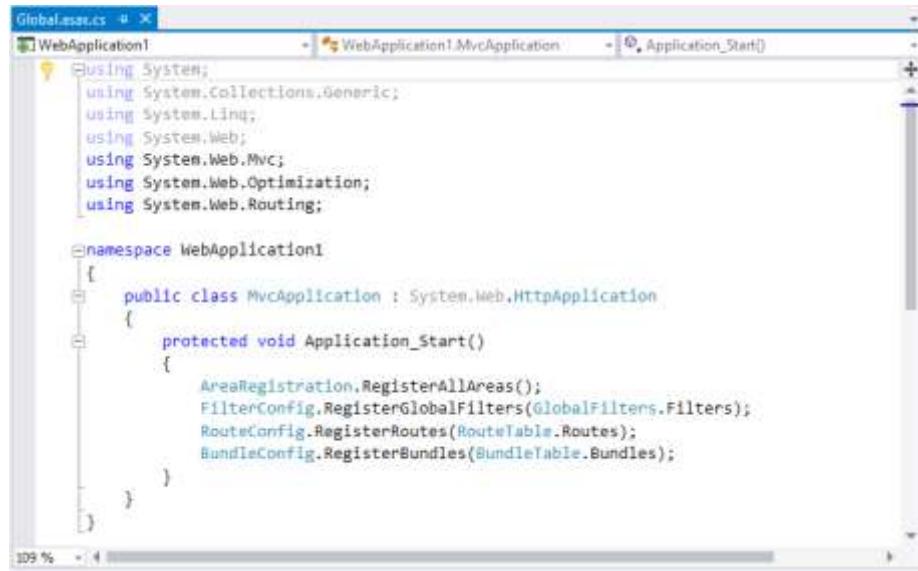


### El archivo global.asax

Toda aplicación ASP.NET MVC es una instancia de una clase derivada de System.Web.HttpApplication. Esta clase es el punto de entrada de nuestra aplicación, es el Main de la aplicación web.

Desde este archivo podemos manejar eventos a nivel de aplicación, sesión, cache, autenticacion, etc.

Este archivo varia mucho desde la versión anterior de ASP.NET MVC, aunque el funcionamiento es el mismo. En ASP.NET MVC se ha incluido el directorio App\_Start que nos permite organizar como se inicializa la aplicación.

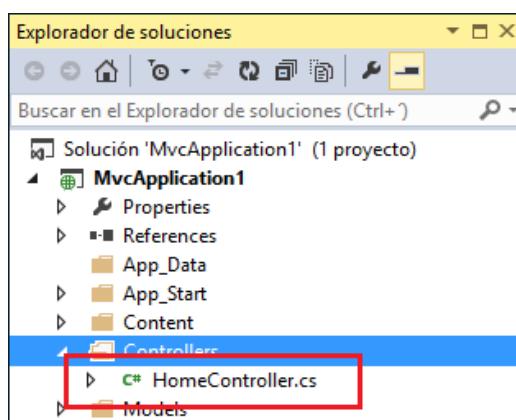


```
Global.asax.cs # X
WebApplication1 -> WebApplication1.MvcApplication -> Application_Start()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace WebApplication1
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

### Home Controller

La clase HomeController es el punto de entrada de la aplicación, la página por defecto. Cuando creamos un nuevo proyecto ASP.NET MVC se crea también un controlador HomeController situado directamente el folder Controllers.



## Laboratorio 1.1

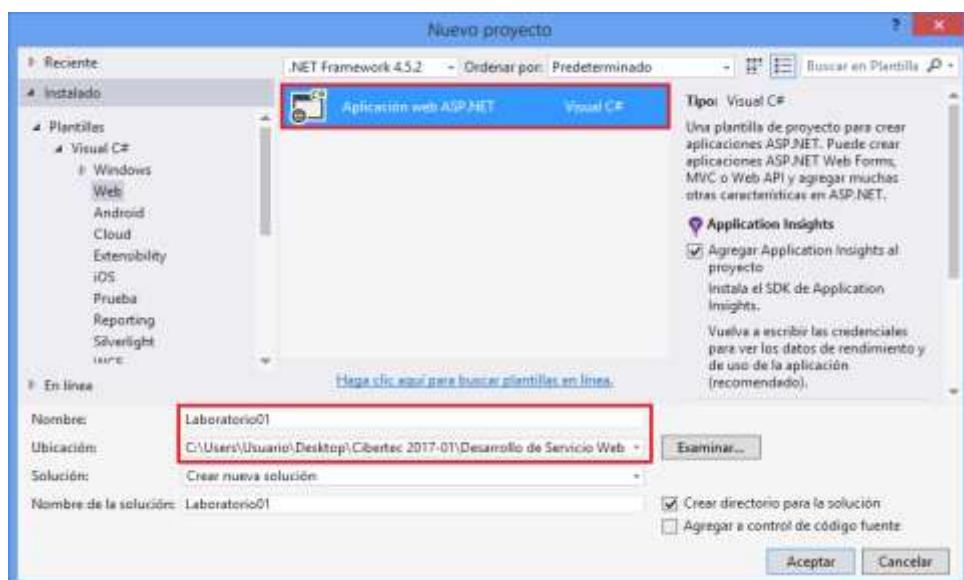
### Creando una aplicación ASP.NET MVC

Implemente un proyecto ASP.NET MVC aplicando el patrón arquitectónico MVC donde permita crear una página de inicio de un sitio web.

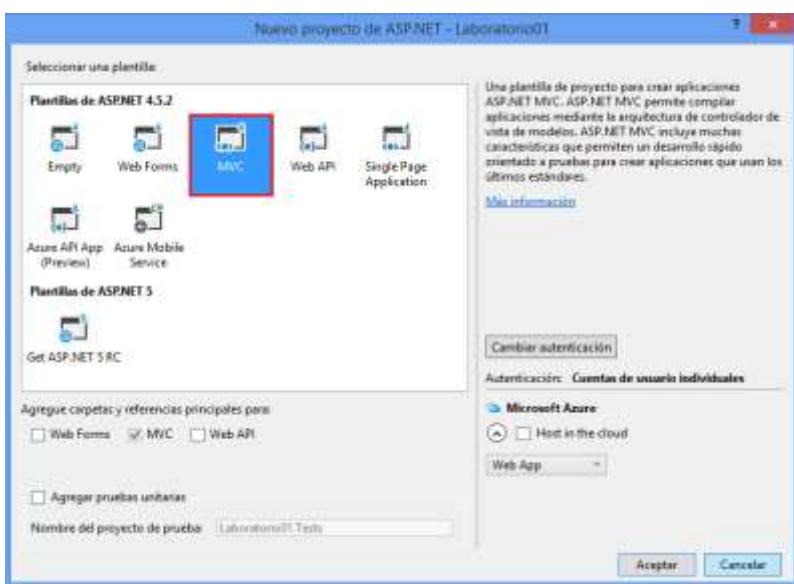
#### Creando el proyecto

Iniciamos Visual Studio 2015 y creamos un nuevo proyecto:

1. Seleccionar el proyecto Web Visual Studio 2015
2. Seleccionar el FrameWork: 4.5.2
3. Seleccionar la plantilla Aplicación web de ASP.NET
4. Asignar el nombre del proyecto
5. Presionar el botón ACEPTAR

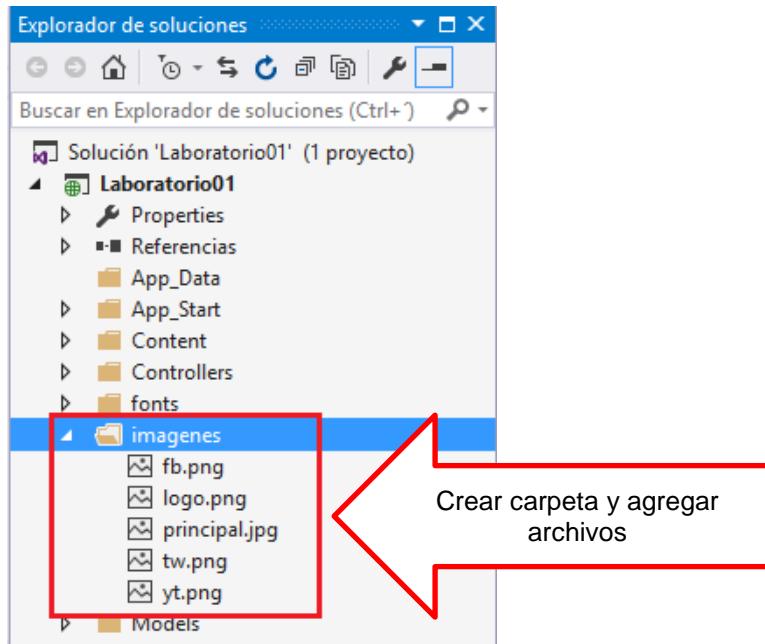


A continuación, seleccionar la plantilla del proyecto MVC. Presiona el botón ACEPTAR



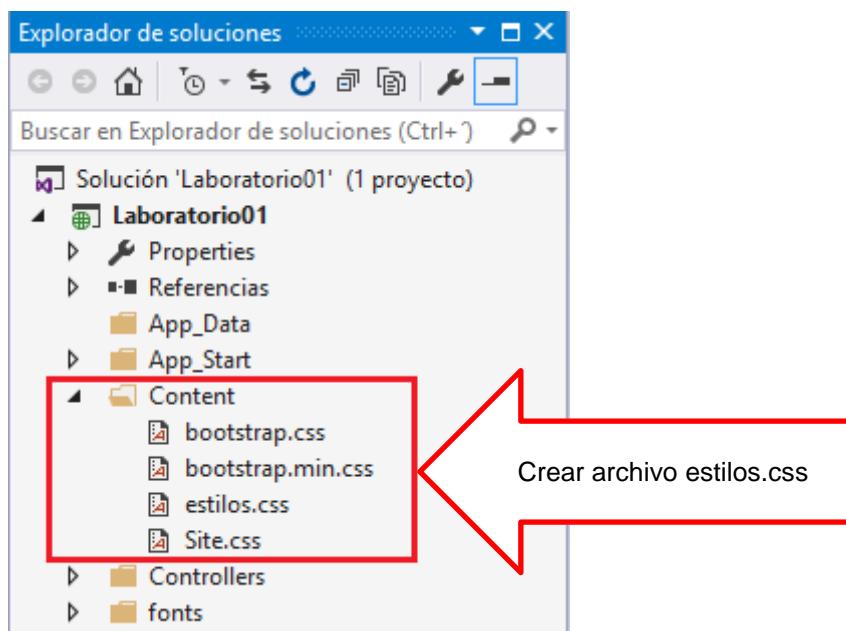
## Agregando la carpeta imágenes

En el explorador de soluciones, agregar una **carpeta Nueva**, llamada imágenes. En dicha carpeta agregar los archivos de imágenes: jpg.

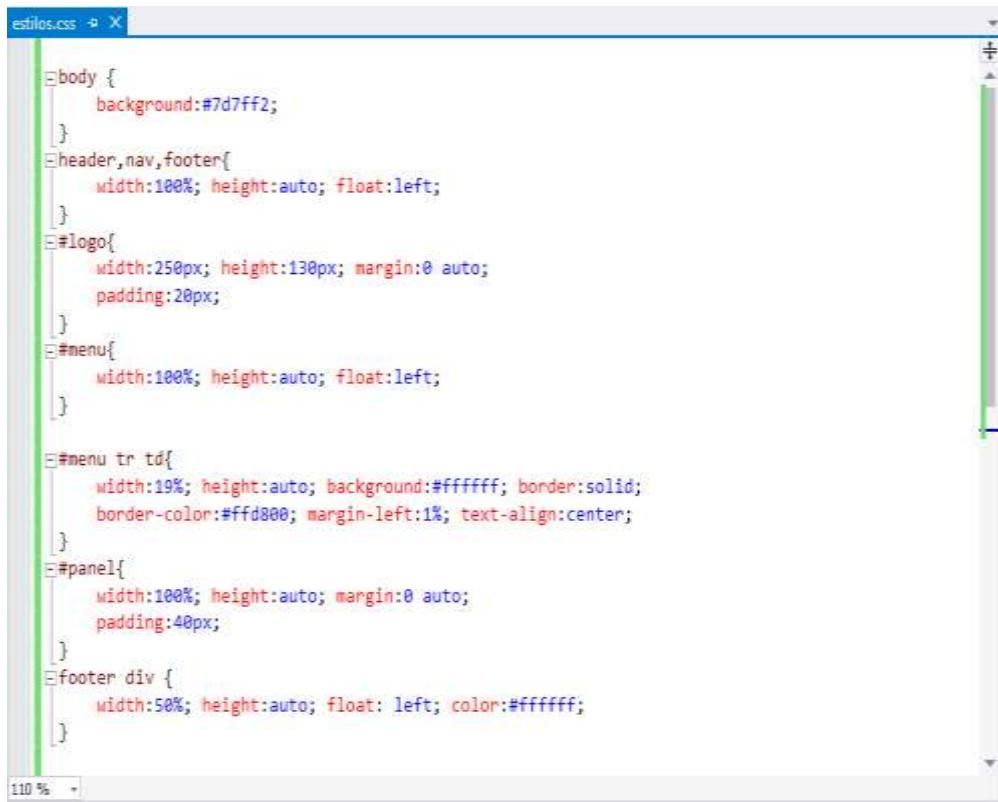


## Agregando Hojas de Estilo css

Para brindar un mejor estilo a la vista, agregamos, en la carpeta Content, una hoja de estilo llamada estilos.css, tal como se muestra la figura.



A continuación definimos estilos a las etiquetas que utilizará las páginas cshtml.



```

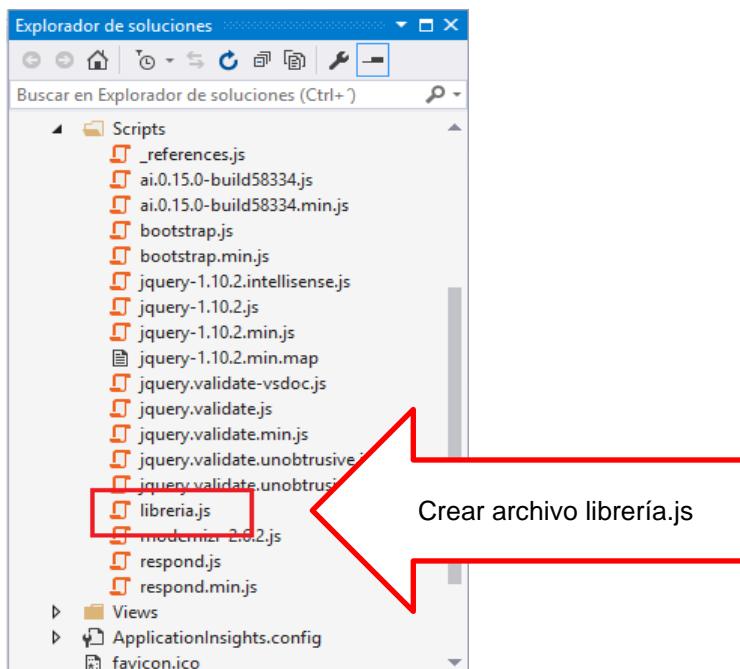
estilos.css X
body {
    background:#7d7ff2;
}
header,nav,footer{
    width:100%; height:auto; float:left;
}
#logo{
    width:250px; height:130px; margin:0 auto;
    padding:20px;
}
#menu{
    width:100%; height:auto; float:left;
}

#menu tr td{
    width:19%; height:auto; background:#fffff; border:solid;
    border-color:#ffd800; margin-left:1%; text-align:center;
}
#panel{
    width:100%; height:auto; margin:0 auto;
    padding:40px;
}
footer div {
    width:50%; height:auto; float: left; color:#fffff;
}

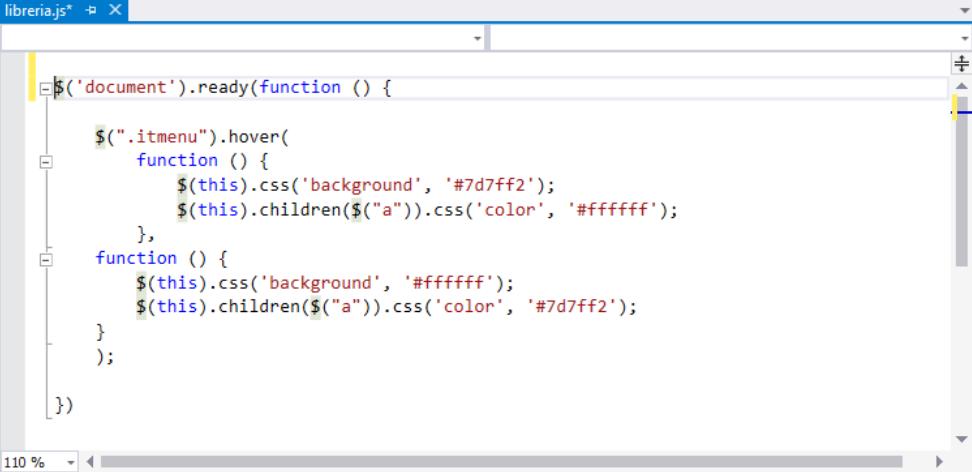
```

### Agregando archivo Script js

Para programar la página principal.cshtml, agregamos, en la carpeta Script, un archivo librería.js, tal como se muestra la figura



A continuación visualizamos el contenido del archivo js, donde programamos la clase .itmenu en el evento hover.



```

$(document).ready(function () {
    $(".itmenu").hover(
        function () {
            $(this).css('background', '#7d7ff2');
            $(this).children("a").css('color', '#ffffff');
        },
        function () {
            $(this).css('background', '#ffffff');
            $(this).children("a").css('color', '#7d7ff2');
        }
    );
})

```

### Trabajando con el \_Layout

Abrir la pagina \_Layout, para realizar el diseño de la página maestra.

Primero agregamos un link para enlazarnos al archivo estilos.css, y agregamos el link script para librería.js, tal como se muestra



```

!DOCTYPE html

    
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>@ViewBag.Title - Mi aplicación ASP.NET</title>
        @Styles.Render("~/Content/css")
        @Scripts.Render("~/bundles/modernizr")
        <link href("~/Content/estilos.css" rel="stylesheet" />
    
    <body>
        @RenderBody()
        @Scripts.Render("~/bundles/jquery")
        @Scripts.Render("~/bundles/bootstrap")
        <script src="~/Scripts/libreria.js"></script>
        @RenderSection("scripts", required: false)
    </body>
</html>

```

Agregar archivo estilos.css

Agregar el archivo libreria.js

En el body del \_Layout, diseña los bloques <header> y <nav> la cual se visualizará en todas las páginas que utilicen esta página maestra. Guardar los cambios efectuados en el archivo.

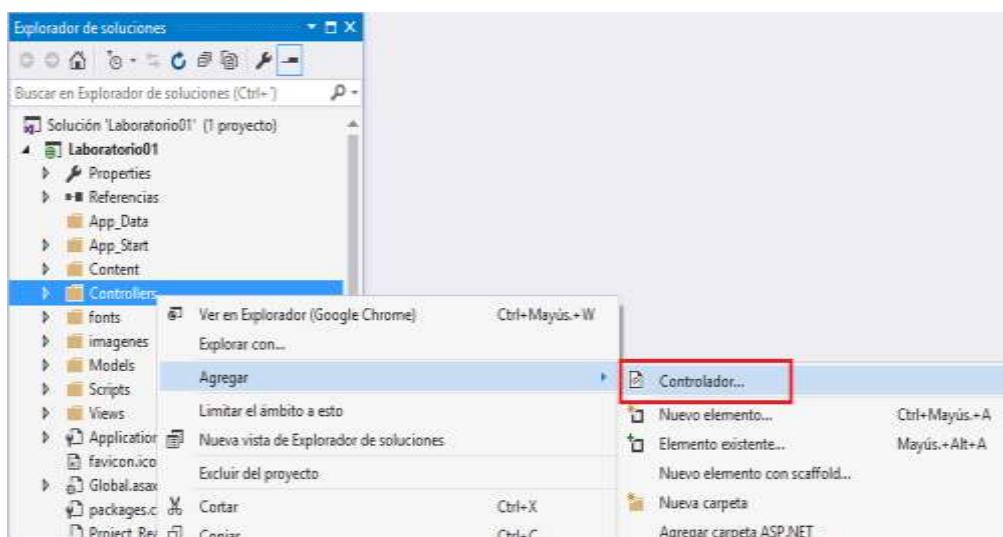
```

<_Layout.cshtml> X
    <body>
        <header>
            <center></center>
        </header>
        <nav>
            <table id="menu">
                <tr>
                    <td class="itmenu"><a href="">Inicio</a></td>
                    <td class="itmenu"><a href="">Productos</a></td>
                    <td class="itmenu"><a href="">Venta</a></td>
                    <td class="itmenu"><a href="">Promociones</a></td>
                    <td class="itmenu"><a href="">Contactenos</a></td>
                </tr>
            </table>
        </nav>
        <@RenderBody()>
        <footer>
            <div>
                Supermercados Castillejos<br />
                Calle 152, San Isidro<br />
                Teléfono: 123654<br />
                email: castillejos@hotmail.com<br />
                url: http://www.castillejos.com
            </div>
            <div>
                Pueden visitarnos en:<br />
                
                
                
            </div>
        </footer>
        <@Scripts.Render("~/bundles/jquery")>
        <@Scripts.Render("~/bundles/bootstrap")>
        <script src="~/Scripts/libreria.js"></script>
        <@RenderSection("scripts", required: false)>
    </body>

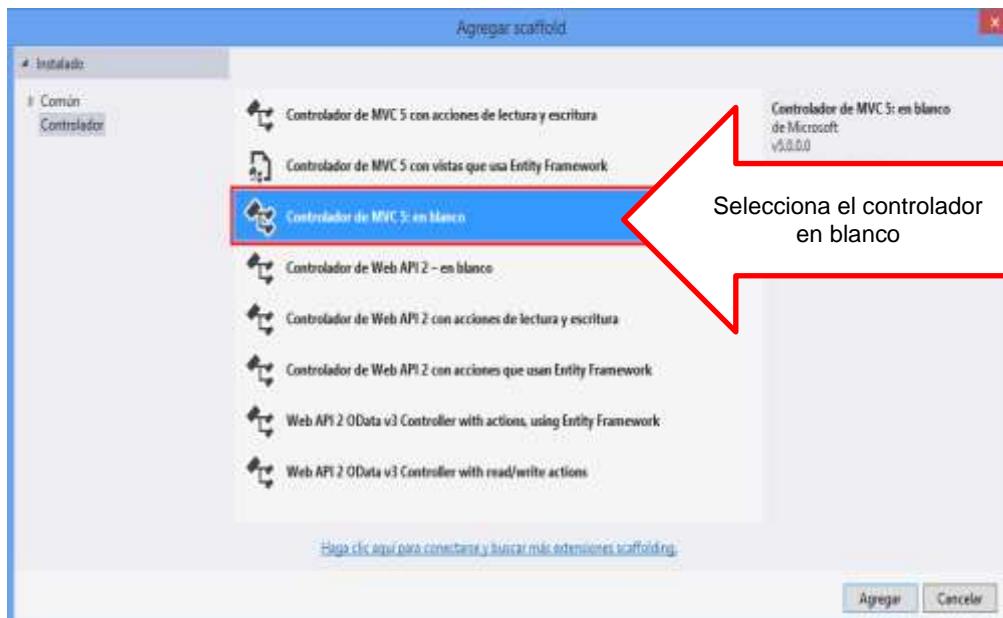
```

## Agregando un Controlador al Proyecto

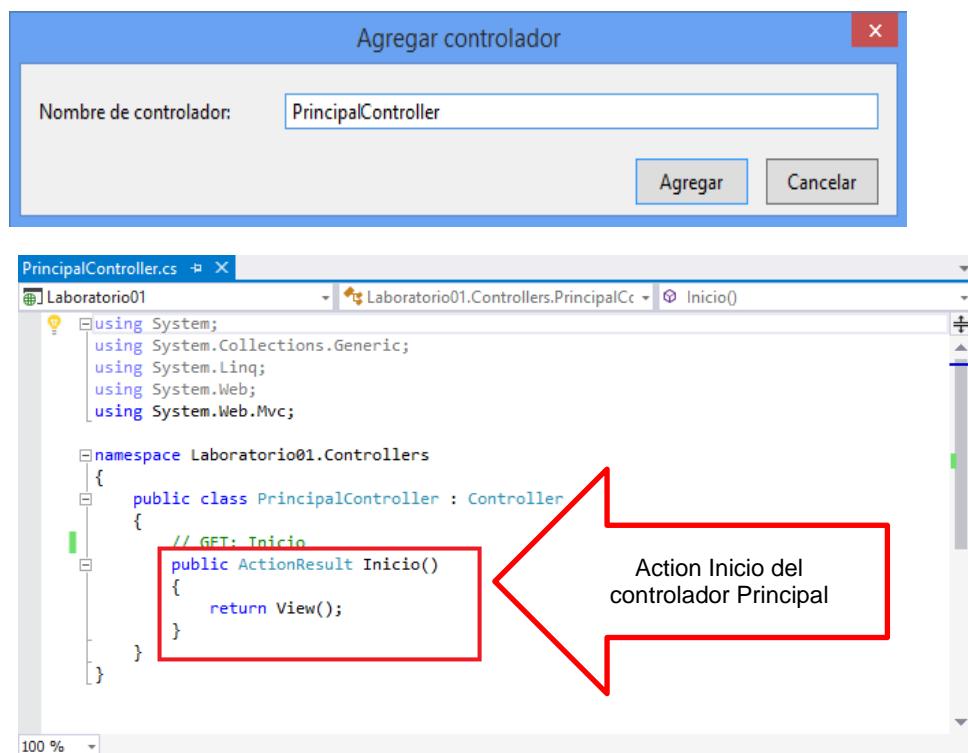
A continuación agregamos el controlador: En la carpeta Controllers, selecciona la opción **Agregar → Controlador...**, tal como se muestra en la figura.



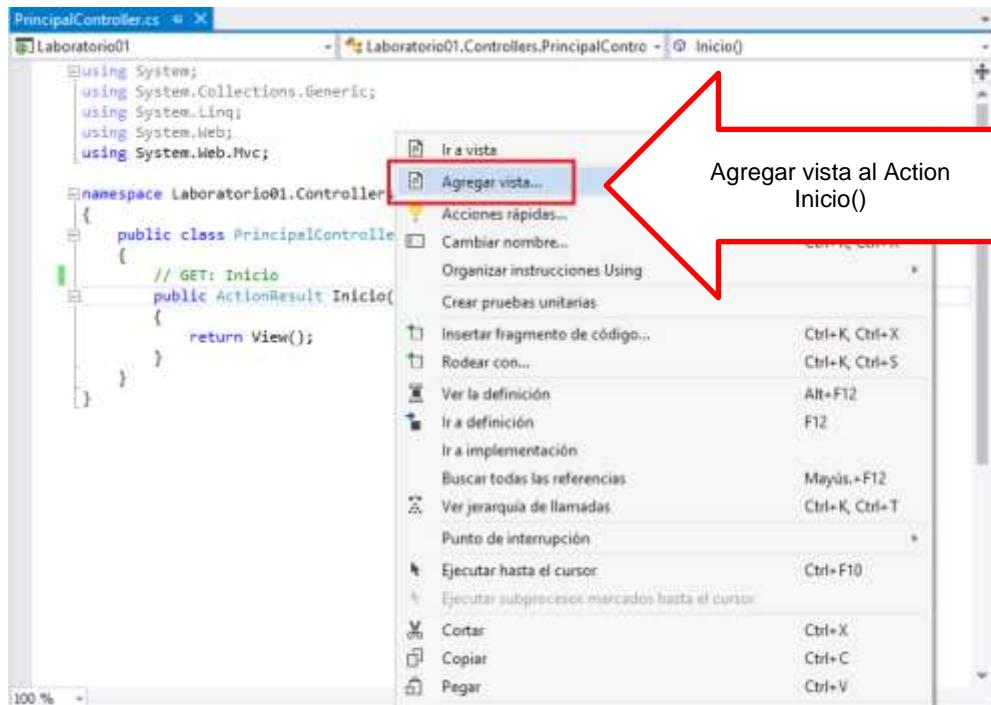
En la ventana Scaffold, selecciona el tipo de controlador. Para nuestra aplicación seleccionamos el controlador en blanco, tal como se muestra. A continuación presionar el botón Agregar



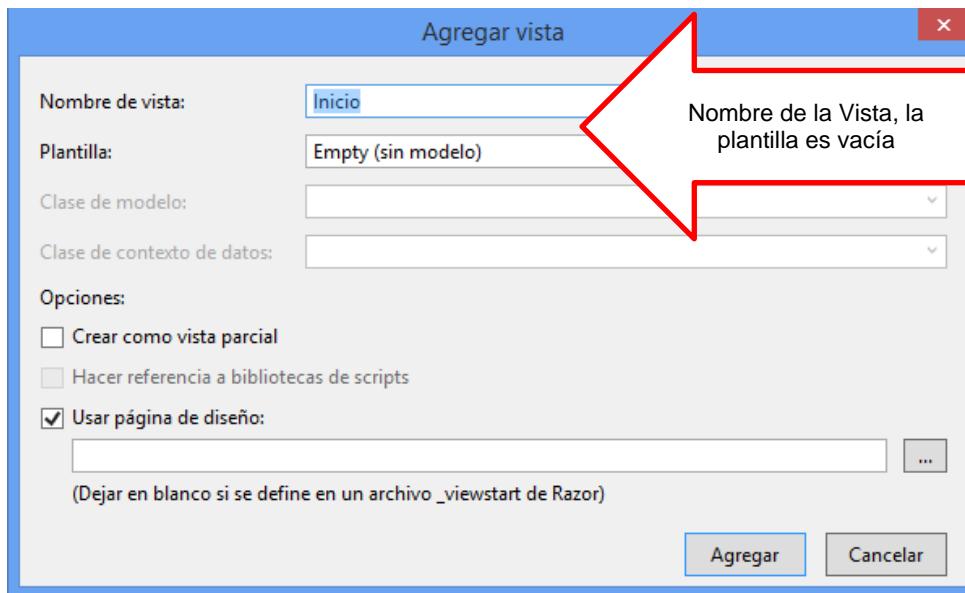
En la ventana Agregar controlador, asigne el nombre del controlador: PrincipalController, tal como se muestra en la figura.  
Presione el botón Agregar.



En el controlador, se muestra el ActionResult Inicio(). A continuación vamos a agregar una vista al Action.



En la ventana Agregar vista, se visualiza el nombre de la vista. No hacer cambios, por ahora. Presiona el botón AGREGAR



A continuación se muestra la Vista inicio.cshtml.

Realice los cambios a la vista, tal como se muestra en la figura



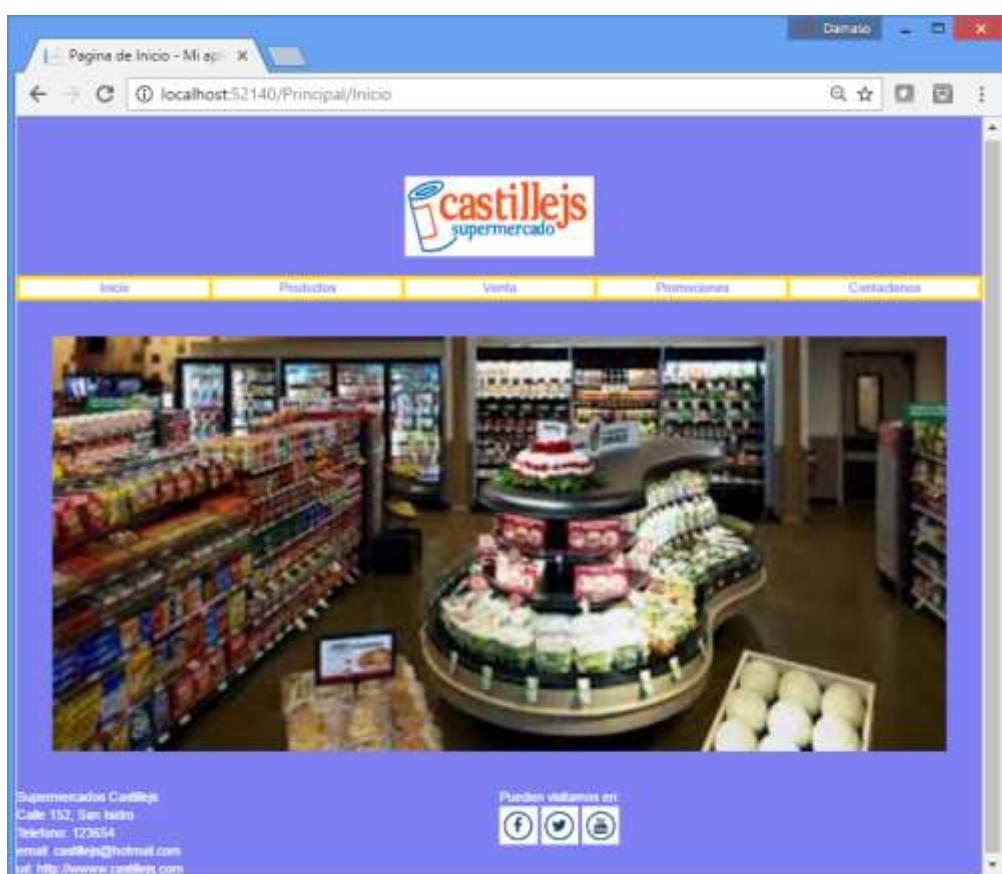
```
Inicio.cshtml + X PrincipalController.cs
<link href="~/Content/estilos.css" rel="stylesheet" />
@{
    ViewBag.Title = "Pagina de Inicio";
}


```

110 %

Agrega enlace a la hoja de estilo

Ejecute proyecto, presiona la tecla Ctrl + F5, donde se visualiza la Vista y su diseño a través de la pagina.



# Resumen

- └ ASP.NET MVC es una implementación reciente de la arquitectura Modelo-Vista-Controlador sobre la base ya existente del Framework ASP.NET otorgándonos de esta manera un sin fin de funciones que son parte del ecosistema del Framework .NET.
- └ Entre las características más destacables de ASP.NET MVC tenemos las siguientes:
  - Uso del patrón Modelo-Vista-Controlador.
  - Facilidad para el uso de Unit Tests.
  - Uso correcto de estándares Web y REST.
  - Sistema eficiente de routing de links.
  - Control a fondo del HTML generado.
  - Uso de las mejores partes de ASP.NET.
- └ El marco ASP.NET MVC ofrece las siguientes ventajas:
  - Es más fácil de gestionar una aplicación: modelo, la vista y el controlador.
  - No utiliza el estado de vista o formas basadas en servidor. Esto hace que el marco idóneo MVC para los desarrolladores que quieren un control total sobre el comportamiento de una aplicación.
  - Utiliza un patrón Front Controller que procesa las solicitudes de aplicaciones web a través de un solo controlador.
  - Proporciona un mejor soporte para el desarrollo guiado por pruebas (TDD).
  - Funciona bien para las aplicaciones web que son apoyados por grandes equipos de desarrolladores y diseñadores web que necesitan un alto grado de control sobre el comportamiento de la aplicación.
- └ El marco de trabajo basado en formularios Web ofrece las siguientes ventajas:
  - Es compatible con un modelo de eventos que conserva el estado a través de HTTP, lo que beneficia el desarrollo de aplicaciones Web de línea de negocio.
  - Utiliza un patrón Controlador que añade funcionalidad a las páginas individuales.
  - Utiliza el estado de vista sobre las formas basadas en servidor, que puede hacer la gestión de la información de estado más fácil.
  - Funciona bien para pequeños equipos de desarrolladores web y diseñadores que quieren aprovechar el gran número de componentes disponibles para el desarrollo rápido de aplicaciones.
- └ El Modelo-Vista-Controlador (MVC) es un patrón arquitectónico que separa una aplicación en tres componentes principales: el modelo, la vista y el controlador. El marco ASP.NET MVC proporciona una alternativa al modelo de formularios Web Forms ASP.NET para crear aplicaciones Web.
- └ Entre las características de este patrón:
  - Soporte para la creación de aplicaciones para Facebook.
  - Soporte para proveedores de autenticación a través del OAuth Providers.
  - Plantillas por default renovadas, con un estilo mejorado.
  - Mejoras en el soporte para el patrón Inversion Of Control e integración con Unity
  - Mejoras en el ASP.NET Web Api, para dar soporte a las implementaciones basadas en RESTful
  - Validaciones en lado del modelo
  - Uso de controladores Asíncronos
  - Soporte para el desarrollo de aplicaciones Web Móvil, totalmente compatible con los navegadores de los modernos SmartPhone (Windows Phone, Apple y Android), etc.
- └ Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
  - [https://msdn.microsoft.com/es-es/library/dd410120\(v=vs.100\).aspx](https://msdn.microsoft.com/es-es/library/dd410120(v=vs.100).aspx)
  - <http://es.slideshare.net/ogensollen/desarrollo-de-aplicaciones-web-con-aspnet-mvc5>
  - <http://nakedcode.net/?p=193>





# INTRODUCCION AL DESARROLLO WEB

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla interfaces de usuario para una aplicación Web utilizando el patrón de diseño MVC.

## TEMARIO

### Tema 2: Arquitectura de ASP.NET MVC (5 horas)

- 2.1 Plataforma ASP.NET MVC
- 2.2 Controladores
  - 2.2.1 URL de enrutamiento
  - 2.2.2 Acciones del controlador
  - 2.2.3 Implementando acciones (POST/GET)
- 2.3 Vistas
  - 2.3.1 Sintaxis Razor y Scaffolding
  - 2.3.2 ViewData y ViewBag
  - 2.3.3 Enviar datos de los controladores a las vistas y controladores
  - 2.3.4 Validaciones: ModelState, DataAnnotations

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando en patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



## 2. ARQUITECTURA DE ASP.NET MVC

El modelo–vista–controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento

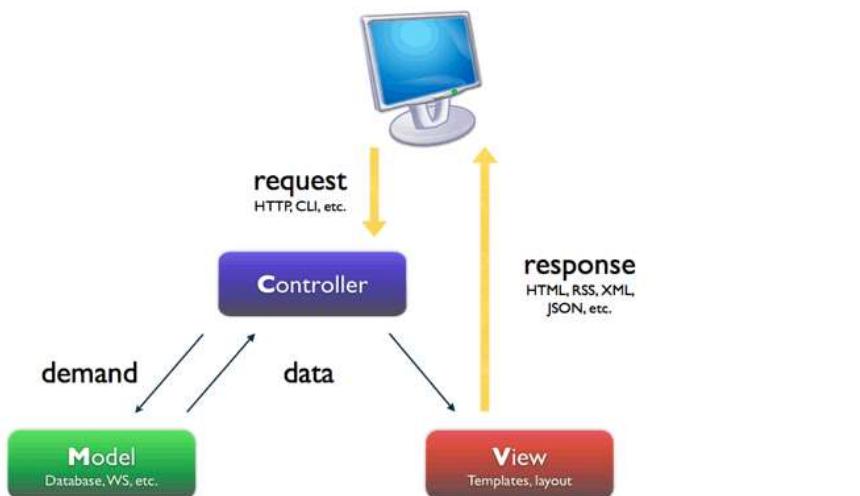


Figura: 1

Referencia: [http://librosweb.es/libro/jobeet\\_1\\_4/capitulo\\_4/la\\_arquitectura\\_mvc.html](http://librosweb.es/libro/jobeet_1_4/capitulo_4/la_arquitectura_mvc.html)

Los componentes del patrón MVC se podrían definir como sigue:

- El **Modelo**: Es la representación de la información con la cual el sistema opera y gestiona todos los accesos a la información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la **Vista** aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al **Modelo** a través del **Controlador**.
- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al **Modelo** cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su **Vista** asociada si se solicita un cambio en la forma en que se presenta el **Modelo** (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el **Controlador** hace de intermediario entre la **Vista** y el **Modelo**.
- La **Vista**: Presenta el **Modelo** (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho **Modelo** la información que debe representar como salida.

El diagrama del patrón MVC funciona de la siguiente manera:

- El navegador realiza una petición a una determinada URL.
- ASP.NET MVC recibe la petición y determinar el controlador que debe ejecutarse (veremos más adelante como se realiza este proceso).

- El controlador recibe la petición HTTP.
- Procesa los datos, y crea u obtiene el modelo.
- Retorna una vista, a la que normalmente le asigna el modelo (aunque no es necesario establecer un modelo).
- La vista que ha retorna el controlador, es interpretada por el motor de renderización de ASP.NET MVC «Razor», que procesa la vista para generar el documento HTML que será devuelto finalmente al navegador
- El navegador muestra la página.
- Nota: ASP.NET MVC dispone de varios motores de renderización – Razor, aspx, - , en este tutorial utilizaremos Razor.

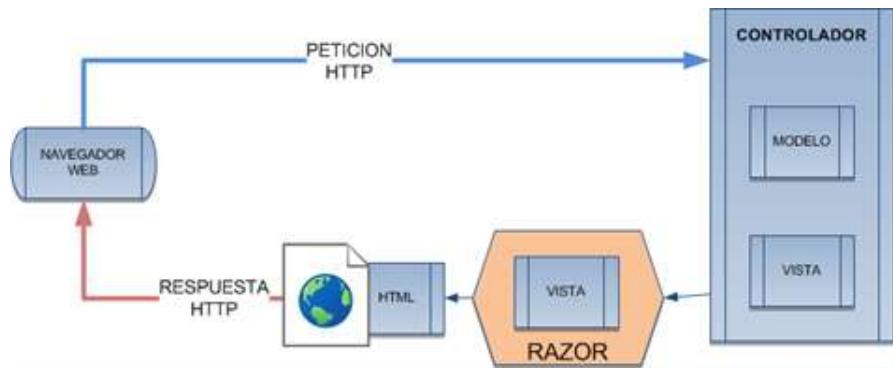


Figura: 2

Referencia: <http://www.devjoker.com/contenidos/articulos/525/Patron-MVC-Modelo-Vista-Controlador.aspx>

El marco de ASP.NET MVC ofrece las siguientes características:

- Separación de tareas: lógica de entrada, lógica de negocios e interfaz de usuario, facilidad para pruebas y desarrollo basado en pruebas. Todos los contratos principales del marco de MVC se basan en interfaz y se pueden probar mediante objetos simulados que imitan el comportamiento de objetos reales en la aplicación.
- Un marco extensible y conectable. Los componentes del marco de ASP.NET MVC están diseñados para que se puedan reemplazar o personalizar con facilidad. Puede conectar su propio motor de vista, directiva de enrutamiento de URL, serialización de parámetros de método y acción, y otros componentes. El marco de ASP.NET MVC también admite el uso de los modelos de contenedor Inyección de dependencia (DI) e Inversión de control (IOC). DI permite insertar objetos en una clase, en lugar de depender de que la clase cree el propio objeto. IOC especifica que si un objeto requiere otro objeto, el primer objeto debe obtener el segundo objeto de un origen externo como un archivo de configuración. Esto facilita las pruebas.
- Amplia compatibilidad para el enrutamiento de ASP.NET, un eficaz componente de asignación de direcciones URL que le permite compilar aplicaciones que tienen direcciones URL comprensibles y que admiten búsquedas. Las direcciones URL no tienen que incluir las extensiones de los nombres de archivo y están diseñadas para admitir patrones de nombres de direcciones URL que funcionan bien para la optimización del motor de búsqueda (SEO) y el direccionamiento de transferencia de estado representacional (REST, Representational State Transfer).
- Compatibilidad para usar el marcado en archivos de marcado de páginas de ASP.NET existentes (archivos .aspx), de controles de usuario (archivos .ascx) y de páginas maestras (archivos .master) como plantillas de vista. Puede usar las características de ASP.NET existentes con el marco de ASP.NET MVC, tales como páginas maestras anidadas, expresiones en línea (<%= %>), controles de servidor declarativos, plantillas, enlace de datos, localización, etc.
- Compatibilidad con las características de ASP.NET existentes. ASP.NET MVC le permite usar características, tales como la autenticación de formularios y la autenticación de Windows, la autorización para URL, la pertenencia y los roles, el

caching de resultados y datos, la administración de estados de sesión y perfil, el seguimiento de estado, el sistema de configuración y la arquitectura de proveedor.

## 2.1 Plataforma ASP.NET MVC

ASP.NET MVC es la plataforma de desarrollo web de Microsoft basada en el conocido patrón Modelo-Vista-Controlador. Está incluida en Visual Studio y aporta interesantes características a la colección de herramientas del programador Web.

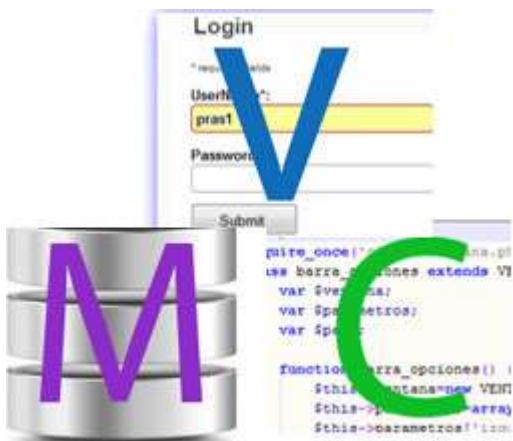


Figura: 3

Referencia: <http://codigobase.com/el-porque-del-mvc-modelo-vista-controlador>

Su arquitectura permite separar las responsabilidades de una aplicación Web en partes diferenciadas y ofrece diversos beneficios:

- Facilidad de mantenimiento
- Facilidad para realizar testeo unitario y desarrollo orientado a pruebas.
- URLs limpias, fáciles de recordar y adecuadas para buscadores.
- Control absoluto sobre el HTML resultante generado, con la posibilidad de crear webs "responsive" usando plantillas del framework Bootstrap de forma nativa.
- Potente integración con jQuery y otras bibliotecas JavaScript.
- Magnífico rendimiento y escalabilidad
- Gran extensibilidad y flexibilidad

### **Características de la plataforma ASP.NET MVC**

**ASP.NET Web API:** Es un nuevo framework el cual nos permite construir y consumir servicios HTTP (web API's) pudiendo alcanzar un amplio rango de clientes el cual incluye desde web browsers hasta dispositivos móviles. ASP.NET Web API es también una excelente plataforma para la construcción de servicios RESTful

**Mejora de los templates predeterminados de proyecto:** Los templates de proyecto de ASP.NET fueron mejorados para obtener sitios web con vistas mucho más modernas y proveer vistas con rendering adaptativo para dispositivos móviles. Los templates utilizan por defecto HTML5 y todas las características de los templates son instaladas utilizando paquetes NuGet de manera que se puede obtener las actualizaciones de los mismos de manera muy simple.

**Templates de proyectos móviles:** En el caso de que estés empezando un nuevo proyecto y quieras que el mismo corra exclusivamente en navegadores de dispositivos móviles y tablets, se puede utilizar el Mobile Application Project template, el cual está basado en jQuery Mobile, una librería open source para construir interfaces optimizadas para el uso táctil

Modos de visualización: el nuevo modo de visualización permite a MVC seleccionar el tipo de vista más conveniente dependiendo del navegador que se encuentre realizando el requerimiento. La disposición de las vistas y las vistas parciales pueden ser sobreescritas dependiendo de un tipo de navegador en particular.

Soporte para llamados asincrónicos basados en tareas: Podemos escribir métodos asincrónicos para cualquier controlador como si fueran métodos ordinarios, los cuales retornan un objeto tipo Task o Task<ActionResult>.

Unión y minimización: Nos permite construir aplicaciones web que carguen mucho más rápidamente y sean más reactivas para el usuario. Estas aplicaciones minimizan el número y tamaño de los requerimientos HTTP que nuestras páginas realizan para recuperar los recursos de JavaScript y CSS.

Mejoras para Razor: ASP.NET MVC 5 incluye la última view engine de Razor, la cual incluye mejor soporte para la resolución de referencias URL utilizando la sintaxis basada en tilde (~/), así como también provee soporte para atributos HTML condicionales.

## 2.2 Controladores

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

La clase **Controller** hereda de **ControllerBase** y es la implementación predeterminada de un controlador. Esta clase es responsable de las fases del procesamiento siguientes:

- Localizar el método de acción adecuado para llamar y validar que se le puede llamar.
- Obtener los valores para utilizar como argumentos del método de acción.
- Controlar todos los errores que se puedan producir durante la ejecución del método de acción.
- Proporcionar la clase WebFormViewEngine predeterminada para representar los tipos de página ASP.NET (vistas).

Todas las clases de controlador deben llevar el sufijo "**Controller**" en su nombre. En el ejemplo siguiente se muestra la clase de controlador de ejemplo, que se denomina HomeController. Esta clase de controlador contiene métodos de acción que representan las páginas de vista.

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.titulo = "Hola Mundo";
        return View();
    }

    public ActionResult Acerca()
    {
        ViewBag.nombre = "Cibertec";
        return RedirectToAction("Index");
    }
}
```

### 2.2.1 URL de Enrutamiento

Por defecto cuando creamos una aplicación ASP.NET MVC se define una tabla de enrutamiento que se encarga de decidir qué controlador gestiona cada petición Web basándose en la URL de dicha petición. Esta forma de enrutamiento presenta dos grandes ventajas con respecto a las aplicaciones tradicionales de ASP.NET:

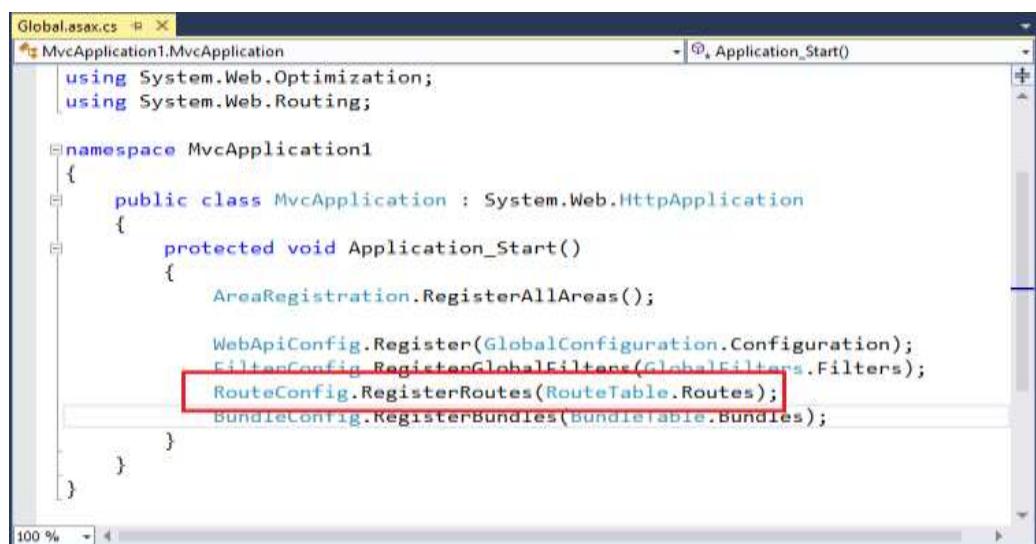
1. En cada petición URL no se asigna un archivo físico del disco como una página .aspx, sino que se asigna una acción de un controlador (más un parámetro), que nos mostrará una vista específica.
2. Las rutas son lógicas, es decir, siguen la estructura definida en la tabla de enrutamiento, lo que favorece y facilita la gestión de la navegación en nuestro sitio.

La tabla de enrutamiento que se genera por defecto al crear una aplicación ASP.NET MVC, la cual se encuentra en archivo **RouteConfig.cs** de la carpeta **App\_Start**.

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

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

Esta tabla de enrutamiento se implementa en el archivo global.asax, y esta definida de la siguiente manera



The screenshot shows the code editor for the Global.asax.cs file. The Application\_Start() method is highlighted. A red box surrounds the line of code: RouteConfig.RegisterRoutes(RouteTable.Routes);. The code is as follows:

```
Global.asax.cs p X
MvcApplication1.MvcApplication
using System.Web.Optimization;
using System.Web.Routing;

namespace MvcApplication1
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();

            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

## 2.2.2 Acciones del controlador

En aplicaciones ASP.NET MVC se organiza en torno a los controladores y métodos de acción. El controlador define los métodos de acción. Los controladores pueden incluir tantos métodos de acción como sea necesario.

Los métodos de acción tienen normalmente una asignación única con las interacciones del usuario. Son ejemplos de interacciones del usuario especificar una dirección URL en el explorador, hacer clic en un vínculo y enviar un formulario. Cada una de estas interacciones del usuario produce el envío de una solicitud al servidor. En cada caso, la dirección URL de la solicitud incluye información que el marco de MVC utiliza para invocar un método de acción.

Cuando un usuario introduce una dirección URL en el explorador, la aplicación MVC usa reglas de enrutamiento que están definidas en el archivo Global.asax para analizar la dirección URL y determinar la ruta de acceso del controlador. A continuación, el controlador determina el método de acción adecuado para administrar la solicitud. De forma predeterminada, la dirección URL de una solicitud se trata como una subrutina de acceso que incluye el nombre del controlador seguido por el nombre de la acción.

### Valores devueltos por el ActionResult

La mayoría de los métodos de acción devuelven una instancia de una clase que se deriva de ActionResult. La clase ActionResult es la base de todos los resultados de acciones. Sin embargo, hay tipos de resultados de acción diferentes, dependiendo de la tarea que el método de acción esté realizando. Por ejemplo, la acción más frecuente consiste en llamar al método **View**. El método **View** devuelve una instancia de la clase **ViewResult**, que se deriva de **ActionResult**. En la siguiente tabla se muestran los tipos de resultado de acción integrados y métodos auxiliares de acción que los devuelven.

Resultado de la acción	Método auxiliar	Descripción
ViewResult	View	Representa una vista como una página web
PartialViewResult	PartialView	Representa una vista parcial que define una sección de la vista que se puede representar dentro de otra vista
RedirectResult	Redirect	Redirecciona a otro método de acción utilizando dirección URL.
RedirectToRouteResult	RedirectToAction	Redirecciona a otro método de acción.
ContentResult	Content	Devuelve un tipo de contenido definido por el usuario
JsonResult	Json	Devuelve un objeto JSON serializado
JavaScriptResult	JavaScript	Devuelve un script que se puede ejecutar en el cliente
FileResult	File	Devuelve una salida binaria para escribir en la respuesta

### Parámetros de los métodos de acción

De forma predeterminada, los valores para los parámetros de los métodos de acción se recuperan de la colección de datos de la solicitud. La colección de datos incluye los pares nombre/valor para los datos del formulario, los valores de las cadenas de consulta y los valores de las cookies.

La clase de controlador localiza el método de acción y determina los valores de parámetro para el método de acción, basándose en la instancia RouteData y en los datos del formulario. Si no se puede analizar el valor del parámetro, y si el tipo del parámetro es un tipo de referencia o un tipo de valor que acepta valores NULL, se pasa null como el valor de parámetro. De lo contrario, se producirá una excepción.

En el ejemplo siguiente, el parámetro id se asigna a un valor de solicitud que también se denomina id. El método de acción no tiene que incluir el código para recibir un valor de parámetro de la solicitud y el valor de parámetro es por consiguiente más fácil de utilizar.

```
public ActionResult Consulta(int id)
{
    ViewData["id"] = id;
    return View();
}
```

El marco de MVC también admite argumentos opcionales para los métodos de acción. Los parámetros opcionales en el marco de MVC se administran utilizando argumentos de tipo que acepta valores NULL para los métodos de acción de controlador. Por ejemplo, si un método puede tomar una fecha como parte de la cadena de consulta, pero desea que el valor predeterminado sea la fecha de hoy si falta el parámetro de cadena de consulta.

```
public ActionResult Consulta(DateTime? fecha)
{
    fecha = DateTime.Today;
    ViewData["fecha"] = fecha;
    return View();
}
```

Si la solicitud no incluye un valor para este parámetro, el argumento es null y el controlador puede tomar las medidas que se requieran para administrar el parámetro ausente.

### 2.2.3 Implementando acciones (POST/GET)

El uso de POST equivale al uso de formularios HTML. La principal diferencia entre enviar datos via POST o via GET (es decir usando la URL, ya sea a través de querystring, o en valores de ruta) es que con POST los datos circulan en el cuerpo de la petición y no son visibles en la URL.



Cuando se envíe el formulario vía POST podemos obtener los datos y realizar operaciones. La realidad es que una acción puede estar implementada por un solo método por cada verbo HTTP, eso significa que para la misma acción (por lo tanto, la misma URL) puedo tener dos métodos en el controlador: uno que se invoque a través de GET y otro que se invoque a través de POST. Así pues podemos añadir el siguiente método a nuestro controlador.



```

index.cshtml  => HomeController.cs*  cliente.cs*
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
@using (Html.BeginForm("Registro", "Home", FormMethod.Post))
{
}

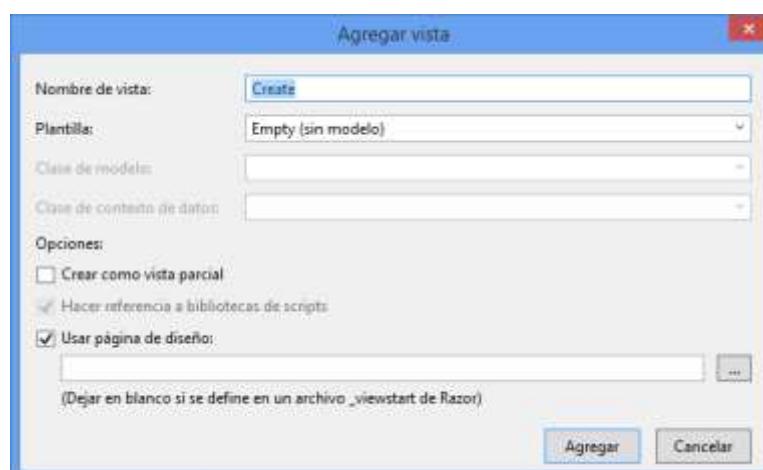
```

## 2.3 Vistas

En el modelo Model-View-Controller (MVC), las vistas están pensadas exclusivamente para encapsular la lógica de presentación. No deben contener lógica de aplicación ni código de recuperación de base de datos. El controlador debe administrar toda la lógica de aplicación. Una vista representa la interfaz de usuario adecuada usando los datos que recibe del controlador. Estos datos se pasan a una vista desde un método de acción de controlador usando el método View.

Las vistas en MVC ofrecen tres características adicionales de las cuales se puede especificar: Create a strongly-typed view, Create as a parcial view y Use a layout or master page

- **Creación de Vistas de Tipado Fuerte:** Esta casilla se selecciona cuando la vista va a estar relacionada a un Modelo y este objeto debe ser un parámetro de la acción en el controlador.
- **Creacion de Vistas Parciales:** Cuando es necesario reutilizar código HTML en diferentes partes del proyecto, se crea una vista de este tipo. Por ejemplo el menú debe estar presente en gran parte de la aplicación, esta vista sería parcial y solo se crearía una sola vez. Para crear una vista parcial se debe nombrar de la siguiente forma: \_NombreVistaParcial, la nombre se le debe anteponer el símbolo “\_”. Ejemplo \_LoginPartial.cshtml ubicado en la carpeta /Views/Shared.
- **Usar como Plantilla o Pagina Maestra:** Es una vista genérica de toda la aplicación, es la que contendrá el llamado a los archivos JS y CSS. Las vistas de este tipo deben cumplir con la misma regla para llamar el archivo, al nombre se le debe anteponer el símbolo “\_”. El llamado dinámico de las vistas por el Controlador se realiza por medio de la función RenderBody().



### 2.3.1 Sintaxis Razor y Scaffolding

**Scaffolding** implica la creación de plantillas a través de los elementos del proyecto a través de un método automatizado.

Los scaffolds generan páginas que se pueden usar y por las que se puede navegar, es decir, implica la construcción de páginas CRUD. Los resultados que se aplica es ofrecer una funcionalidad limitada.

La técnica scaffolding es un proceso de un solo sentido. No es posible volver a aplicar la técnica scaffolding en los controladores y las vistas para reflejar un modelo sin sobrescribir los cambios. Por lo tanto, se debe evaluar los módulos que se han personalizado para saber a qué modelos se les puede volver a aplicar la técnica scaffolding y a cuáles no.

Cuando tiene la clase del modelo listo, Scaffolding y la Vista permite realizar CRUD (Create, Read, Update, Delete) operaciones. Todo lo que necesitas hacer es seleccionar la plantilla scaffold y el modelo de datos para generar los métodos de acción que se implementarán en el controlador.



**Razor** es una sintaxis basada en C# que permite usarse como motor de programación en las vistas o plantillas de nuestros controladores. No es el único motor para trabajar con ASP.NET MVC. Entre los motores disponibles destaco los más conocidos: Spark, NHaml, Brail, StringTemplate o NVelocity, algunos de ellos son conversiones de otros lenguajes de programación. En Razor el símbolo de la arroba (@) marca el inicio de código de servidor.

El uso de la @ funciona de dos maneras básicas:

- @expresión: Renderiza la expresión en el navegador. Así @item.Nombre muestra el valor de ítem.Nombre.
- @{ código }: Permite ejecutar un código que no genera salida HTML.

### 2.3.2 ViewData y ViewBag

El **ViewData** es un objeto del tipo diccionario (clave – valor) que no requiere instanciarse. En este modelo se pueden pasar datos desde el controlador a la vista a través de una clase diccionario “ViewDataDictionary”

```
public ActionResult Index()
{
    ViewData["id"] = valor;
    return View();
}
```

El **ViewBag** es muy parecido a el **ViewData**, es un objeto tipo clave – valor pero se le asigna de manera diferente. En este modelo no requiere ser casteados.

```
public ActionResult Index()
{
    ViewBag.id= valor;
    return View();
}
```

El **ViewModel** permite para pasar información de una acción de un controlador a una vista. Esta propiedad Model no funciona como las anteriores, sino que lo que se pasa es un objeto, que se manda de la acción hacia la vista.

Cuando usamos Model para acceder a los datos, en lugar de **ViewData** o **ViewBag**, definimos en la primera línea, @model. Esa línea le indica al framework de que tipo es el objeto que la vista recibe del controlador (es decir, de que tipo es la propiedad Model).

A diferencia de **ViewData** y **ViewBag** que existen tanto en el controlador como en la vista, el controlador no tiene una propiedad Model. En su lugar se usa una sobrecarga del método View() y se manda el objeto como parámetro

### 2.3.3 Enviar datos de los controladores a las vistas y controladores

Para pasar datos a la vista, se utiliza la propiedad **ViewData** y **ViewBag**.

Un método de acción puede almacenar los datos en el objeto **TempDataDictionary** del controlador antes de llamar al método **RedirectToAction** del controlador para invocar la acción siguiente. La propiedad **TempData** almacena el estado de la sesión. Los métodos de acción que se llaman después de que se haya establecido el valor de **TempDataDictionary** pueden obtener los valores de objeto y, a continuación, procesarlos o mostrarlos. El valor de **TempData** se conserva hasta que se lee o hasta que se agota el tiempo de espera de la sesión.

**Session** es la forma de persistir datos mientras la sesión actual esté activa. Esto nos permite acceder a datos desde múltiples controladores, acciones y vistas. El acceso y a una variable de **Session** es igual que a una variable de TempData:

Existen 3 modos de sesión en ASP.NET:

- **InProc**: Se guarda en el propio servidor web y no se comparte entre otros servidores de una misma web farm.
- **StateServer**: Se guarda en un servidor único y se comparte entre otros servidores de una misma web farm, pero si se cae el servidor de sesión, se cae toda la aplicación.
- **SQLServer**: Se guarda en la base de datos, es menos performante y difícil de escalar.

### 2.3.4 Validaciones: ModelState, DataAnnotations

El espacio de nombres **System.ComponentModel.DataAnnotations**, nos proporciona una serie de clases, atributos y métodos para realizar validación dentro de nuestros programas en .NET.

Cuando se utiliza el Modelo de Datos Anotaciones Binder, utiliza atributos validador para realizar la validación. El namespace **System.ComponentModel.DataAnnotations** incluye los siguientes atributos de validación:

- Rango - Permite validar si el valor de una propiedad se sitúa entre un rango específico de valores.
- RegularExpression - Permite validar si el valor de una propiedad coincide con un patrón de expresión regular especificada.
- Requerido - Le permite marcar una propiedad según sea necesario.
- StringLength - Le permite especificar una longitud máxima para una propiedad de cadena.
- Validación - La clase base para todos los atributos de validación.

#### Tipos de atributos

**ErrorMessage**, es una propiedad de esta clase que heredaran todos nuestros atributos de validación y que es importante señalar ya que en ella podremos customizar el mensaje que arrojará la validación cuando esta propiedad no pase la misma.

```
[Required(ErrorMessage = "Ingrese el codigo del cliente")]
public string idcliente { get; set; }
```

Esta propiedad tiene un '**FormatString**' implícito por el cual mediante la notación estándar de '**FormatString**' podemos referirnos primero al nombre de la propiedad y después a los respectivos parámetros.

#### MaxLengthAttribute y MinLengthAttribute

Estos atributos fueron añadidos para la versión de Entity Framework 4.1. Especifican el tamaño máximo y mínimo de elementos de una propiedad de tipo array.

```
[MaxLength(50, ErrorMessage = "Longitud Maxima 50 caracteres")]
public string direccion { get; set; }
```

Es importante señalar que **MaxLengthAttribute** y **MinLengthAttribute** son utilizados por EF para la validación en el lado del servidor y estos se diferencian del **StringLengthAttribute**, que utilizan un atributo para cada validación, y no solo sirven para validar tamaños de string, sino que también validan tamaños de arrays.

#### RegularExpressionAttribute

Especifica una restricción mediante una expresión regular

```
[RegularExpression("9{5-}", ErrorMessage = "Minimo 5 digitos")]
public string telefono { get; set; }
```

#### RequiredAttribute

Especifica que el campo es un valor obligatorio y no puede contener un valor null o string.

```
[Required(ErrorMessage = "Ingrese el codigo del cliente")]
public string idcliente { get; set; }
```

**RangeAttribute**

Especifica restricciones de rango de valores para un tipo específico de datos

```
[Required(ErrorMessage="Ingrese la edad")]
[Range(18,70,ErrorMessage="18 a 70 años")]
public int edad { get; set; }
```

**DataTypeAttribute**

Especifica el tipo de dato del atributo.

```
[DataType(DataType.PhoneNumber)]
[RegularExpression("[0-9]{7,19}", ErrorMessage = "Minimo 7 maximo 20 digitos")]
public string telefono { get; set; }

[DataType(DataType.EmailAddress)]
[Required(ErrorMessage = "Ingrese el email")]
public string email { get; set; }
```

## Laboratorio 2.1

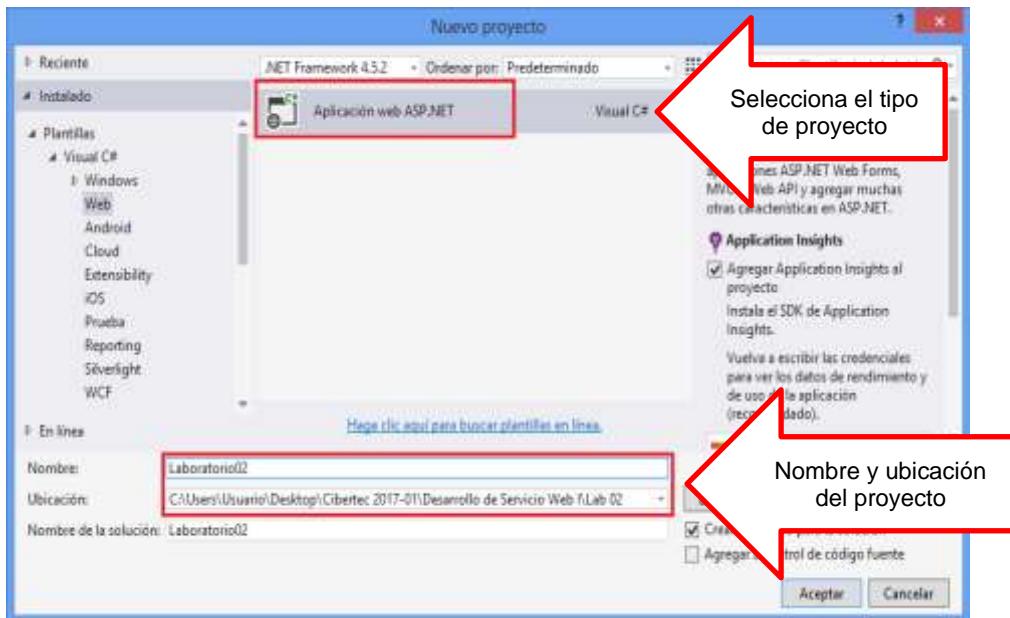
### Creando una aplicación ASP.NET MVC

Implemente un proyecto ASP.NET MVC donde permita listar y registrar los productos desde la web. Valide los datos del producto utilizando anotaciones.

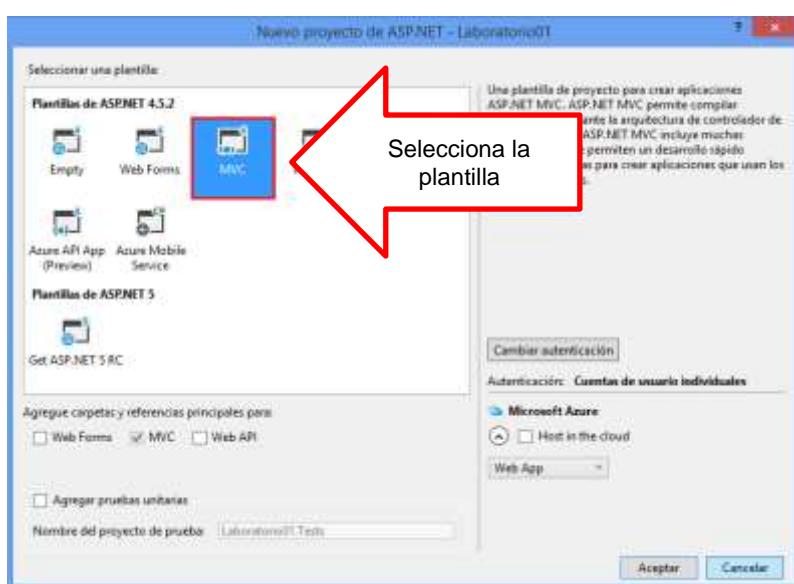
#### Creando el proyecto

Iniciamos Visual Studio 2015 y creamos un nuevo proyecto:

1. Seleccionar el proyecto Web.
2. Seleccionar el FrameWork: 4.5.2
3. Seleccionar la plantilla Aplicación web de ASP.NET
4. Asignar el nombre del proyecto
5. Presionar el botón ACEPTAR

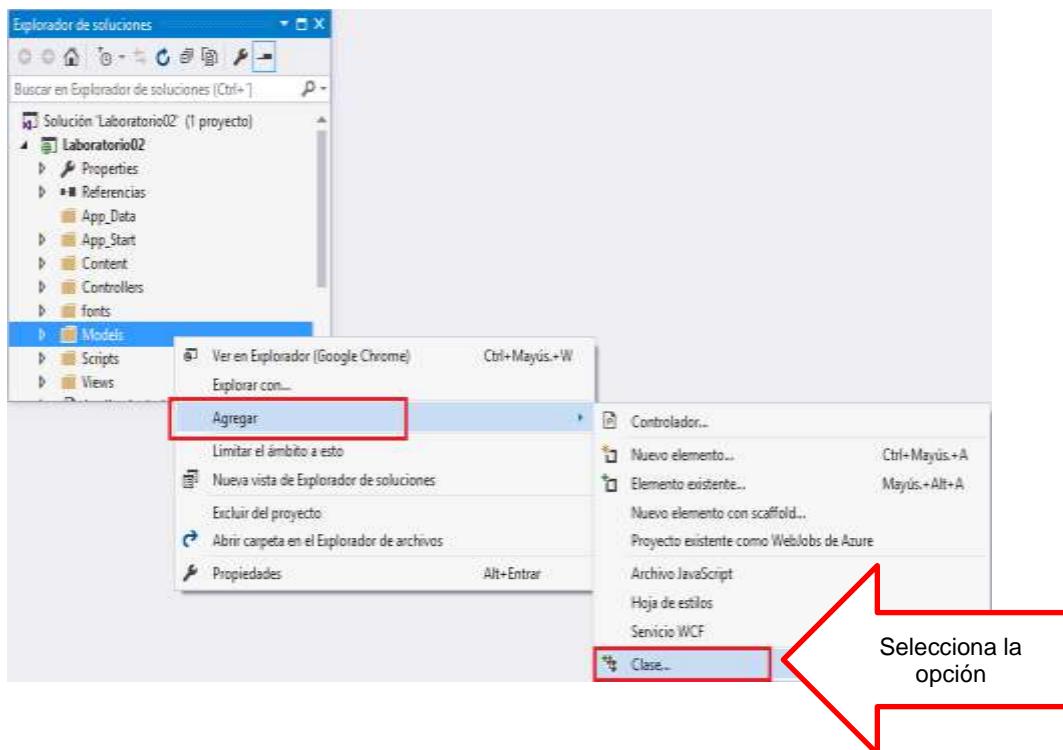


A continuación, seleccionar la plantilla del proyecto MVC. Presiona el botón ACEPTAR

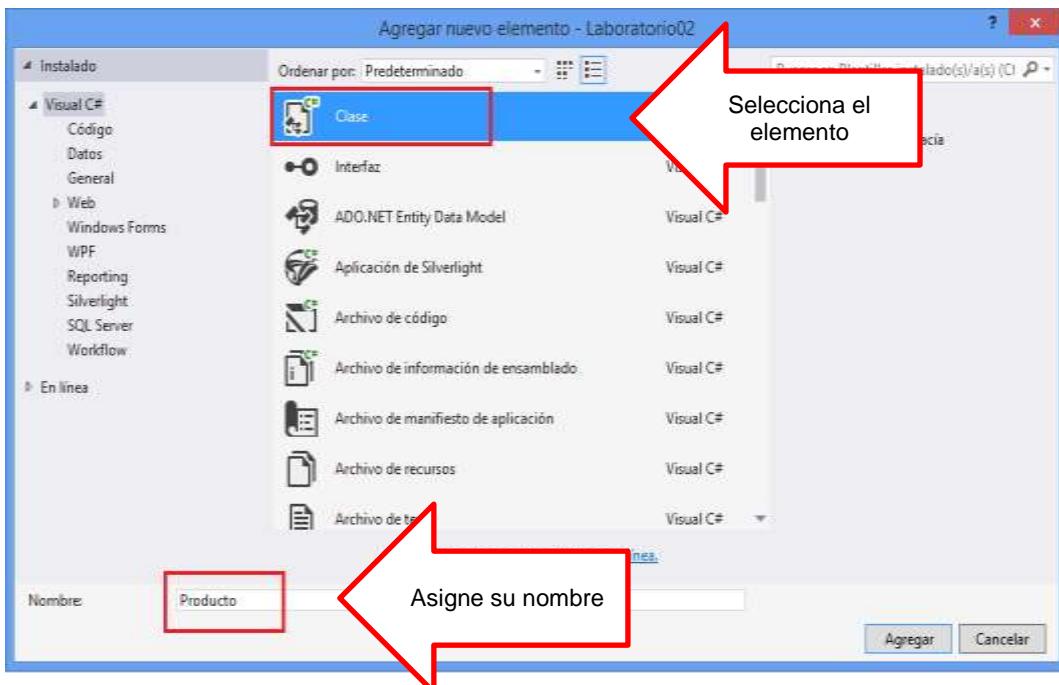


## Trabajando con el Modelo

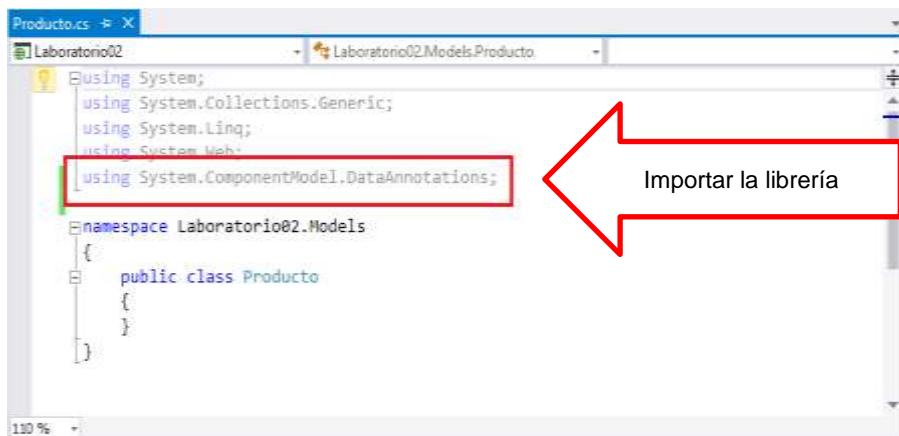
Primero, creamos una clase en la carpeta Models: Agregar una clase llamada Producto, tal como se muestra



En la ventana **Agregar nuevo elemento**, selecciona el elemento **Clase** y asigne el nombre: Producto, presiona el botón Agregar



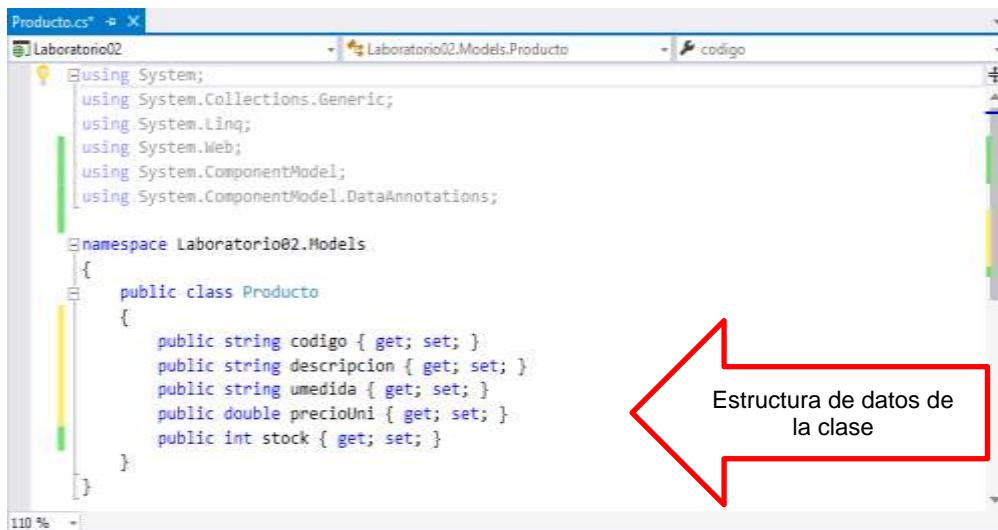
En la clase importar la librería de Anotaciones y Validaciones



```
Producto.cs  * X
Laboratorio02  Laboratorio02.Models.Producto
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace Laboratorio02.Models
{
    public class Producto
    {
    }
}
```

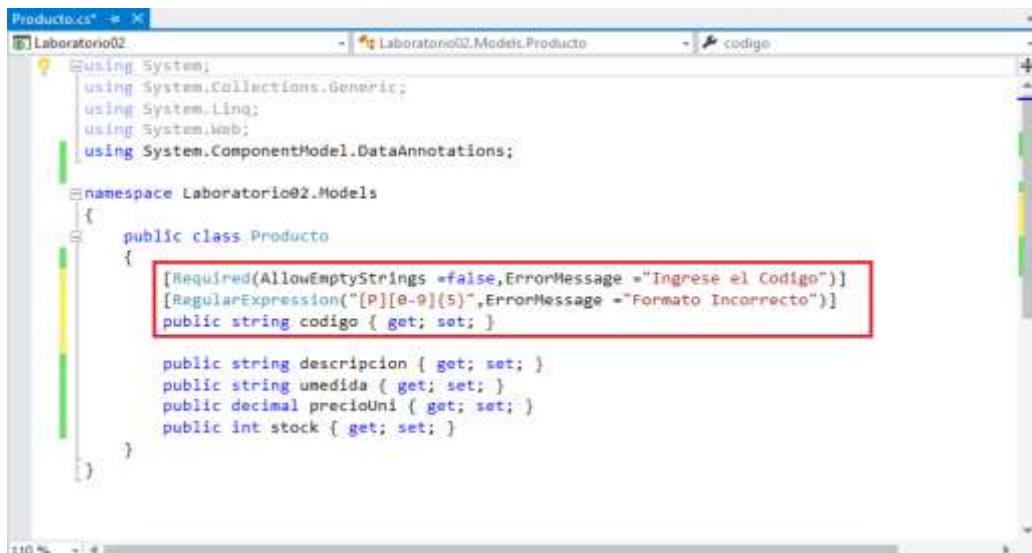
En la clase Producto, primero, defina la estructura de datos de la clase, tal como se muestra.



```
Producto.cs  * X
Laboratorio02  Laboratorio02.Models.Producto  código
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace Laboratorio02.Models
{
    public class Producto
    {
        public string codigo { get; set; }
        public string descripcion { get; set; }
        public string unmedida { get; set; }
        public double precioUni { get; set; }
        public int stock { get; set; }
    }
}
```

A continuación validamos el campo código: no debe estar vacío y su formato es P99999, tal como se muestra

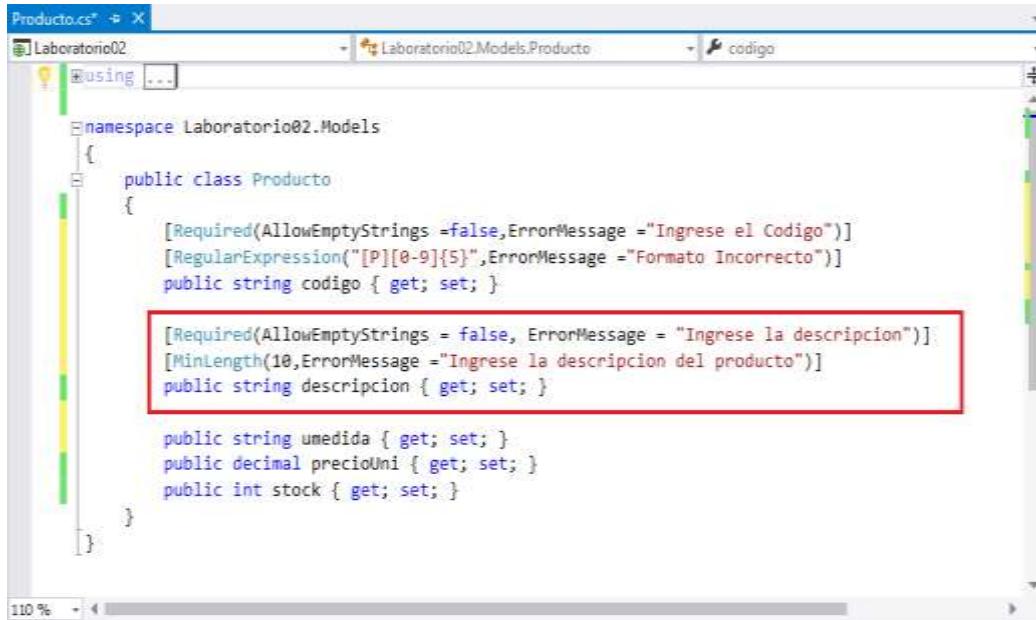


```
Producto.cs  * X
Laboratorio02  Laboratorio02.Models.Producto  código
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace Laboratorio02.Models
{
    public class Producto
    {
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Código")]
        [RegularExpression("[P][0-9]{5}", ErrorMessage = "Formato Incorrecto")]
        public string codigo { get; set; }

        public string descripcion { get; set; }
        public string unmedida { get; set; }
        public decimal precioUni { get; set; }
        public int stock { get; set; }
    }
}
```

Luego validamos el campo descripción: no debe estar vacío y la longitud mínima de caracteres es 10, tal como se muestra



```

using System;
using System.ComponentModel.DataAnnotations;

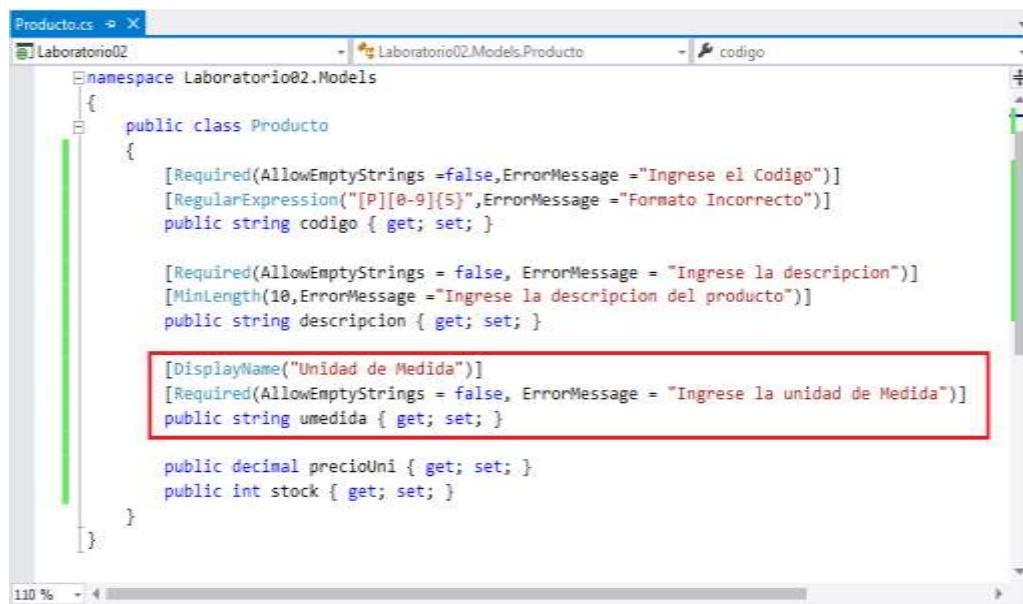
namespace Laboratorio02.Models
{
    public class Producto
    {
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Código")]
        [RegularExpression("[P][0-9]{5}", ErrorMessage = "Formato Incorrecto")]
        public string codigo { get; set; }

        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la descripción")]
        [MinLength(10, ErrorMessage = "Ingrese la descripción del producto")]
        public string descripcion { get; set; }

        public string umedida { get; set; }
        public decimal precioUni { get; set; }
        public int stock { get; set; }
    }
}

```

Continuamos con la validación del campo umedida, el cual será obligatorio, y asignamos el nombre a visualizar (DisplayName), importar la librería System.ComponentModel



```

using System;
using System.ComponentModel.DataAnnotations;

namespace Laboratorio02.Models
{
    public class Producto
    {
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Código")]
        [RegularExpression("[P][0-9]{5}", ErrorMessage = "Formato Incorrecto")]
        public string codigo { get; set; }

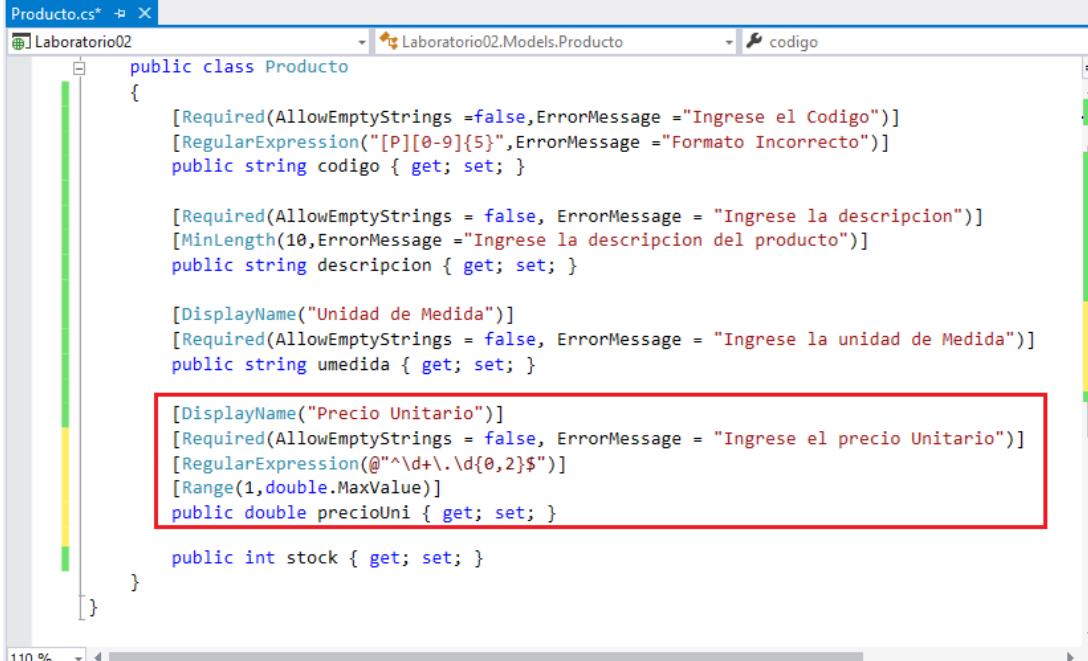
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la descripción")]
        [MinLength(10, ErrorMessage = "Ingrese la descripción del producto")]
        public string descripcion { get; set; }

        [DisplayName("Unidad de Medida")]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la unidad de medida")]
        public string umedida { get; set; }

        public decimal precioUni { get; set; }
        public int stock { get; set; }
    }
}

```

Luego validamos el campo preUni: asignar un Nombre para visualizarlo, indicar que el campo obligatorio y solo debe contener dos decimales (RegularExpression) y su rango de valores se encuentra entre 1 al valor máximo del tipo de dato.



```

public class Producto
{
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Código")]
    [RegularExpression("[P][0-9]{5}", ErrorMessage = "Formato Incorrecto")]
    public string codigo { get; set; }

    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la descripción")]
    [MinLength(10,ErrorMessage = "Ingrese la descripción del producto")]
    public string descripcion { get; set; }

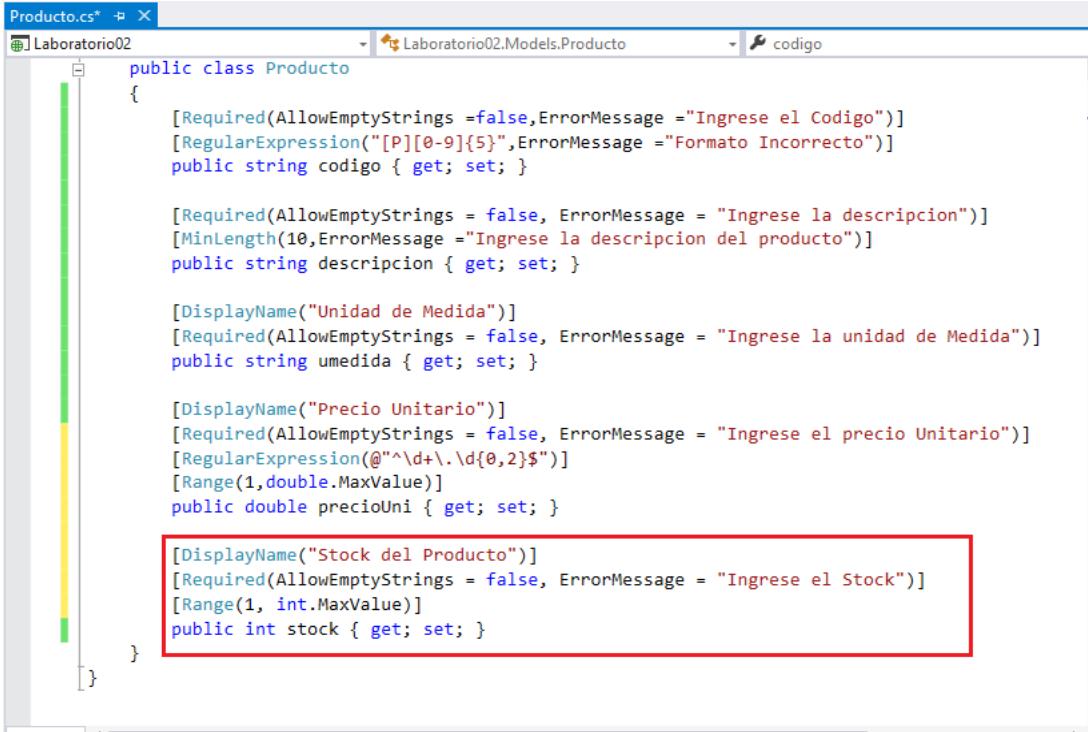
    [DisplayName("Unidad de Medida")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la unidad de medida")]
    public string umedida { get; set; }

    [DisplayName("Precio Unitario")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el precio Unitario")]
    [RegularExpression(@"^\d+\.\d{0,2}$")]
    [Range(1,double.MaxValue)]
    public double precioUni { get; set; }

    public int stock { get; set; }
}

```

Y por ultimo validamos el campo stock: asignamos un nombre, campo obligatorio y cuyo rango de valores es 1 hasta el valor máximo del tipo de dato



```

public class Producto
{
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Código")]
    [RegularExpression("[P][0-9]{5}", ErrorMessage = "Formato Incorrecto")]
    public string codigo { get; set; }

    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la descripción")]
    [MinLength(10,ErrorMessage = "Ingrese la descripción del producto")]
    public string descripcion { get; set; }

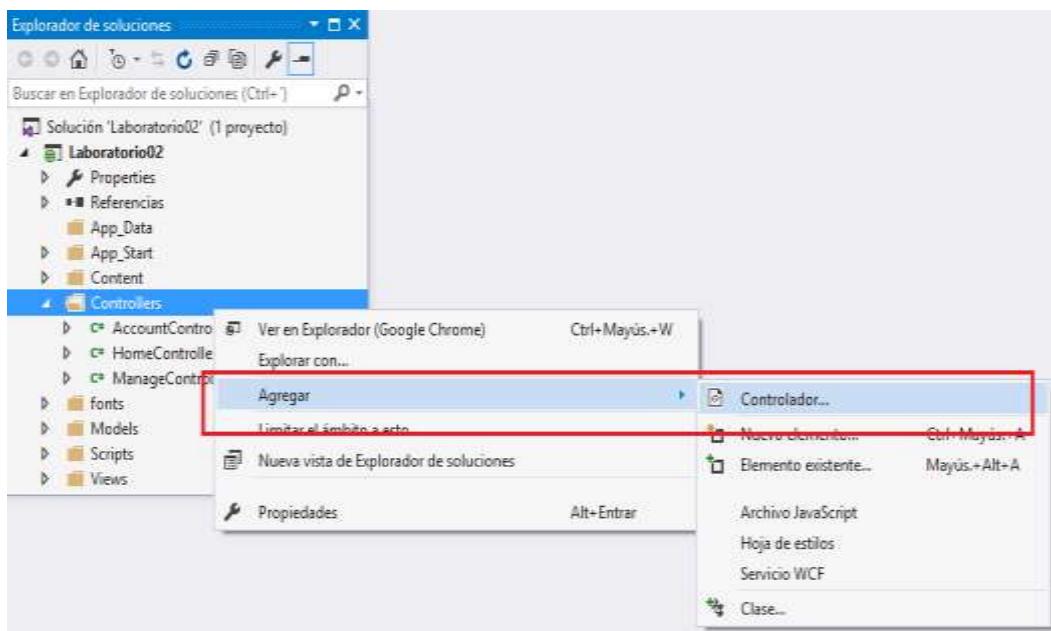
    [DisplayName("Unidad de Medida")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la unidad de medida")]
    public string umedida { get; set; }

    [DisplayName("Precio Unitario")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el precio Unitario")]
    [RegularExpression(@"^\d+\.\d{0,2}$")]
    [Range(1,double.MaxValue)]
    public double precioUni { get; set; }

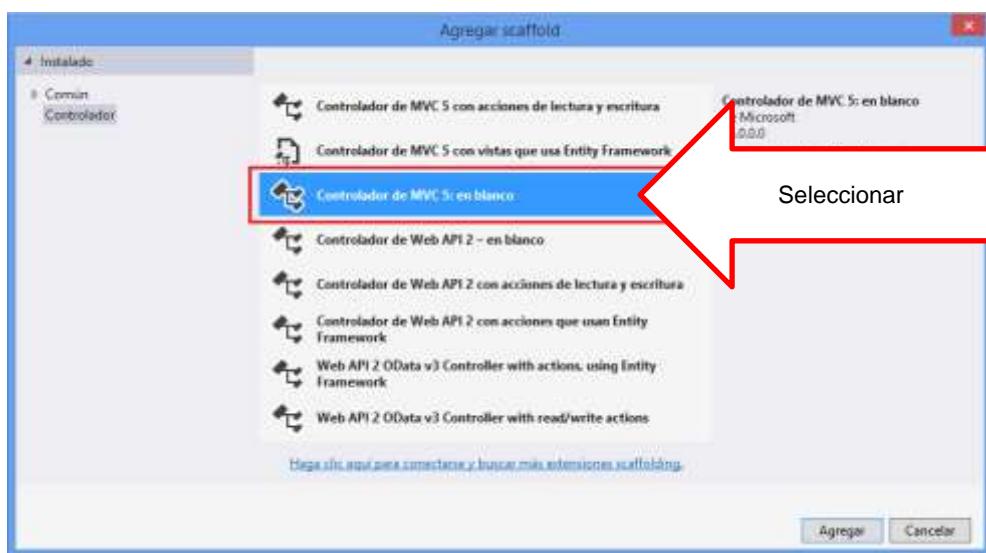
    [DisplayName("Stock del Producto")]
    [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Stock")]
    [Range(1, int.MaxValue)]
    public int stock { get; set; }
}

```

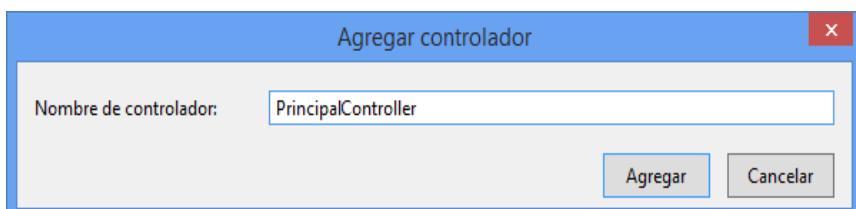
A continuación agregar en la carpeta Controller un controlador, tal como se muestra



En la ventana Scaffold, selecciona el controlador MVC 5 en blanco, tal como se muestra



A continuación ingrese el nombre del Controlador, presiona al botón Agregar



En la ventana de código PrincipalController:

- Importar la carpeta Models, la cual almacena la clase Producto
- Definir una lista de Producto, a nivel controlador

```

PrincipalController.cs  X
Laboratorio02          Laboratorio02.Controllers.PrincipalController  Inventario
  Using System;
  Using System.Collections.Generic;
  Using System.Linq;
  Using System.Web;
  Using System.Web.Mvc;
  Using Laboratorio02.Models;

  Namespace Laboratorio02.Controllers
  {
    Public class PrincipalController : Controller
    {
      // definir la lista de Producto
      List<Producto> Inventario = new List<Producto>();

      Public ActionResult Index()
      {
        Return View();
      }
    }
  }

```

En el ActionResult Index(), enviar a la vista la lista de Producto, llamado Inventario. La sintaxis es:

```
static List<Producto> Inventario = new List<Producto>();
```

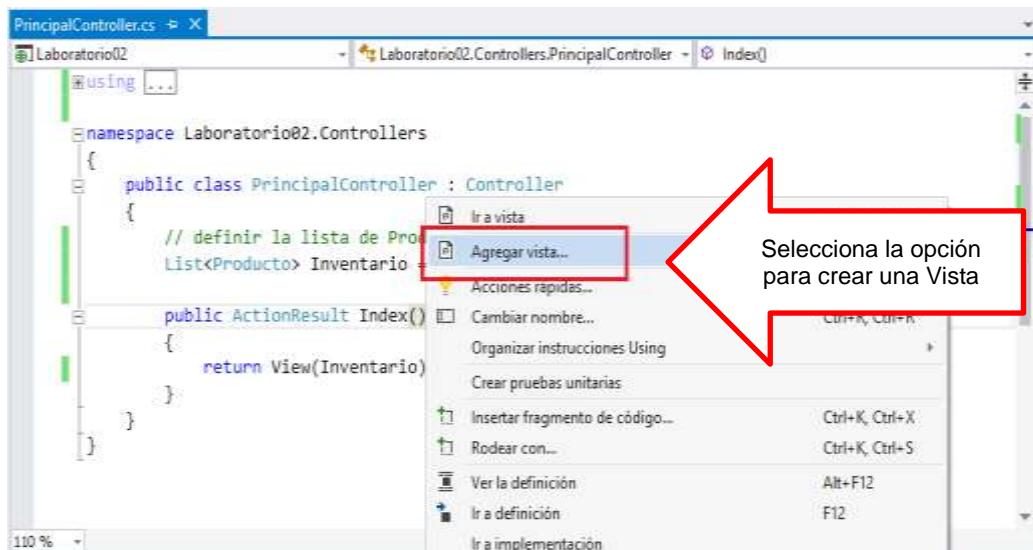
```

PrincipalController.cs  X
Laboratorio02          Laboratorio02.Controllers.PrincipalController  Inventario
  Using ...
  Namespace Laboratorio02.Controllers
  {
    Public class PrincipalController : Controller
    {
      // definir la lista de Producto
      List<Producto> Inventario = new List<Producto>();

      Public ActionResult Index()
      {
        Return View(Inventario);
      }
    }
  }

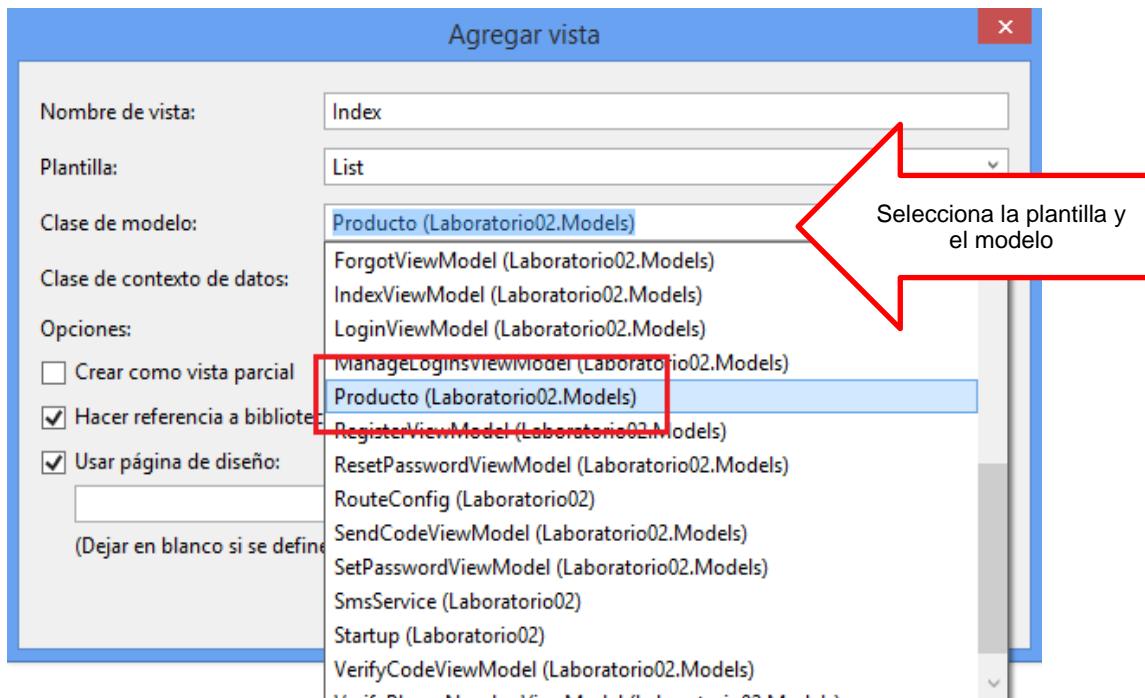
```

A continuación agregar una Vista a la acción Index: hacer click derecho a la acción y selecciona Agregar Vista, tal como se muestra



En la ventana Agregar Vista, aparece el nombre de la Vista: Index. No cambiar el nombre de la vista.

- Selecciona la plantilla: List
- Selecciona la clase de modelo: Producto, el cual listaremos los productos almacenados en la colección.
- Presionar el botón Agregar



Generada la Vista, se visualiza tal como se muestra.

```

Index.cshtml -> PrincipalController.cs
@model IEnumerable<Laboratorio02.Models.Producto>
@if (ViewBag.Title != null)
{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

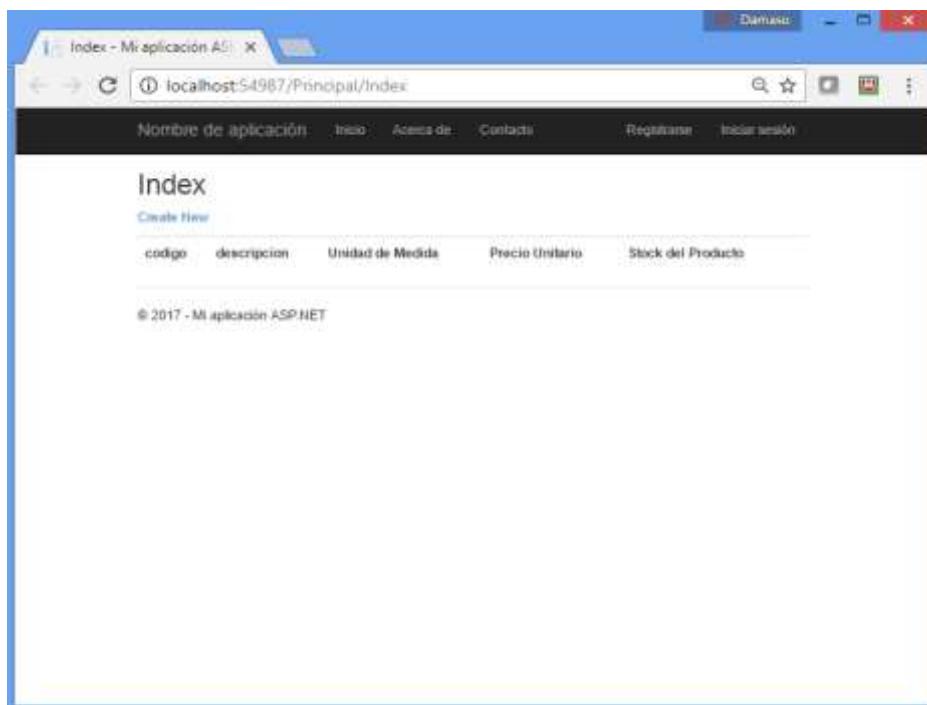

<p>@Html.ActionLink("Create New", "Create")</p>



| @Html.DisplayNameFor(model => model.codigo)                                                                                                                                                                                         | @Html.DisplayNameFor(model => model.descripcion)                                                                                                                                                                   | @Html.DisplayNameFor(model => model.umedida) | @Html.DisplayNameFor(model => model.precioUnitario) | @Html.DisplayNameFor(model => model.stock) |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|-----------------------------------------------------|--------------------------------------------|--|
| @Html.DisplayFor(modelItem => item.codigo) @Html.DisplayFor(modelItem => item.descripcion) @Html.DisplayFor(modelItem => item.umedida) @Html.DisplayFor(modelItem => item.precioUnitario) @Html.DisplayFor(modelItem => item.stock) | @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ }) |                                              |                                                     |                                            |  |


```

Ejecute la Vista Ctrl + F5, visualizando la ventana Index.



En el controlador Principal, defina la acción Create, el cual envía la estructura de la clase Producto, y recibe los datos para ser validados y agregados a la colección.

```

PrincipalController.cs  X
Labatorio02          Laboratorio02.Controllers.PrincipalController  Inventario
namespace Laboratorio02.Controllers
{
    public class PrincipalController : Controller
    {
        // definir la lista de Producto
        List<Producto> Inventario = new List<Producto>();

        public ActionResult Index()
        {
            return View(new Producto());
        }

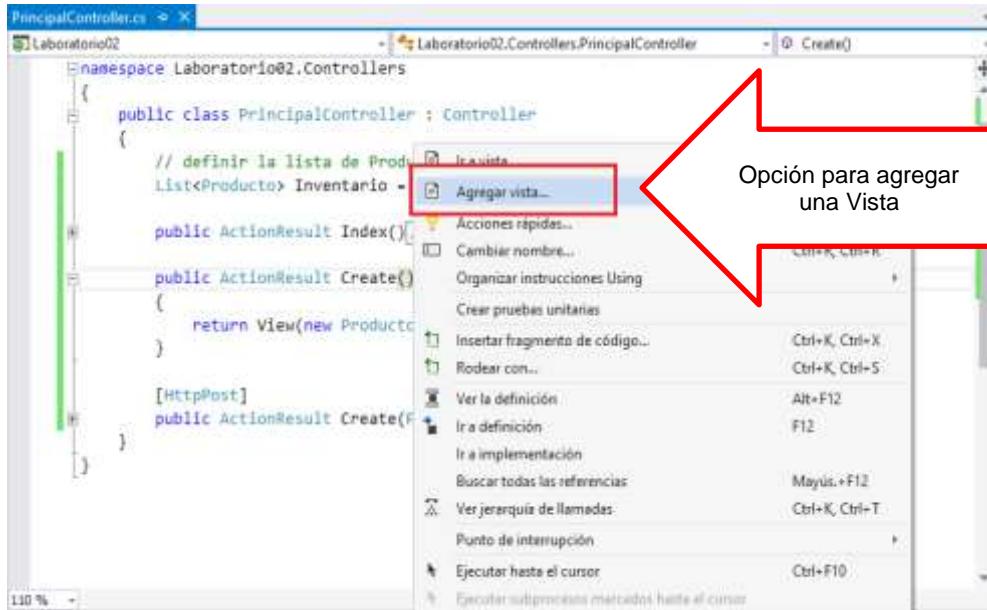
        [HttpPost]
        public ActionResult Create(Producto reg)
        {
            if (!ModelState.IsValid)
            {
                return View(reg);
            }
            //si esta validado los datos ingresados
            Inventario.Add(reg);
            return RedirectToAction("Index");
        }
    }
}

```

Action que envía un nuevo Producto

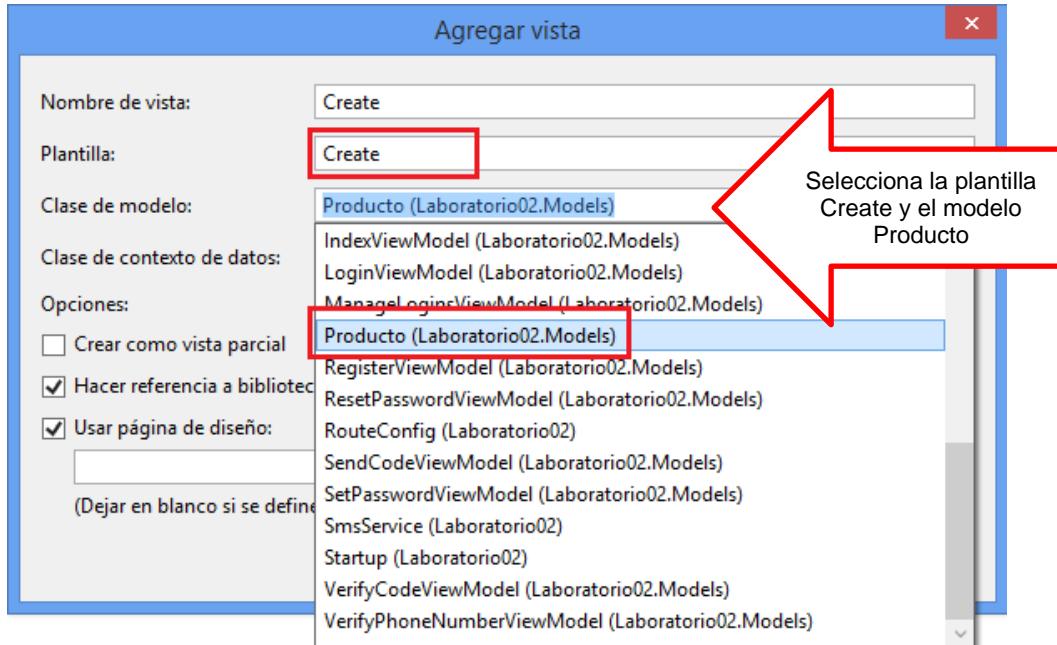
Action que recibe el registro y lo agrega a la colección

Definida la acción Create, agregar una vista: hacer click derecho a la acción y seleccionar la opción Agregar Vista, tal como se muestra



En la ventana Agregar Vista, aparece el nombre de la Vista: Create. No cambiar el nombre de la vista.

- Selecciona la plantilla: Create
- Selecciona la clase de modelo: Producto, el cual ingresamos los datos de un producto a la página.
- Presiona el botón Agregar



Página Create creada por la plantilla Create.

```

Create.cshtml > X PrincipalController.cs
@model Laboratorio02.Models.Producto

{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Producto</h4>
        <br />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.codigo, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.codigo, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.codigo, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.descripcion, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.TextAreaFor(model => model.descripcion, new { @class = "form-control" })
                @Html.ValidationMessageFor(model => model.descripcion, "", new { @class = "text-danger" })
            </div>
        </div>
    </div>
}

```

Ejecuta el proyecto, ingresar a la vista Create, los datos de los productos. Sus campos se encuentran validados desde la definición de la clase, tal como se muestra.

Create - Mi aplicación A

localhost:54987/Principal/Create

Nombre de aplicación Inicio Acerca de Registrarse Iniciar sesión

Create

Producto

codigo  
Ingrese el Código

descripcion  
Ingrese la descripción

Unidad de Medida  
Ingrese la unidad de medida

Precio Unitario  
0  
El campo Precio Unitario debe coincidir con la expresión regular ^d+\\.d{0,2}\$.

Stock del Producto  
0  
El campo Stock del Producto debe estar entre 1 y 2147483647.

Create

Back to List

## Laboratorio 2.2

### Creando una aplicación ASP.NET MVC - RAZOR

Del proyecto ASP.NET MVC anterior, el cual permite listar y registrar los productos desde la web, implemente los mismos procesos utilizando Lenguaje Razor.

#### Trabajando con el Controlador

Desde el mismo proyecto, creamos el controlador Principal.

En esta página hacemos una referencia a la carpeta Models, y definimos la colección de tipo List llamada Inventario.

```

PrincipalController.cs
-----
@using System;
@using System.Collections.Generic;
@using System.Linq;
@using System.Web;
@using System.Web.Mvc;
@using Laboratorio02.Models;

namespace Laboratorio02.Controllers
{
    public class PrincipalController : Controller
    {
        // definir la lista de Producto
        static List<Producto> Inventario = new List<Producto>();
    }
}

```

En el controlador, defina la vista Listado(), el cual envía a la Vista la lista de la colección Inventario (Inventario.ToList()), a través del ViewData.

```

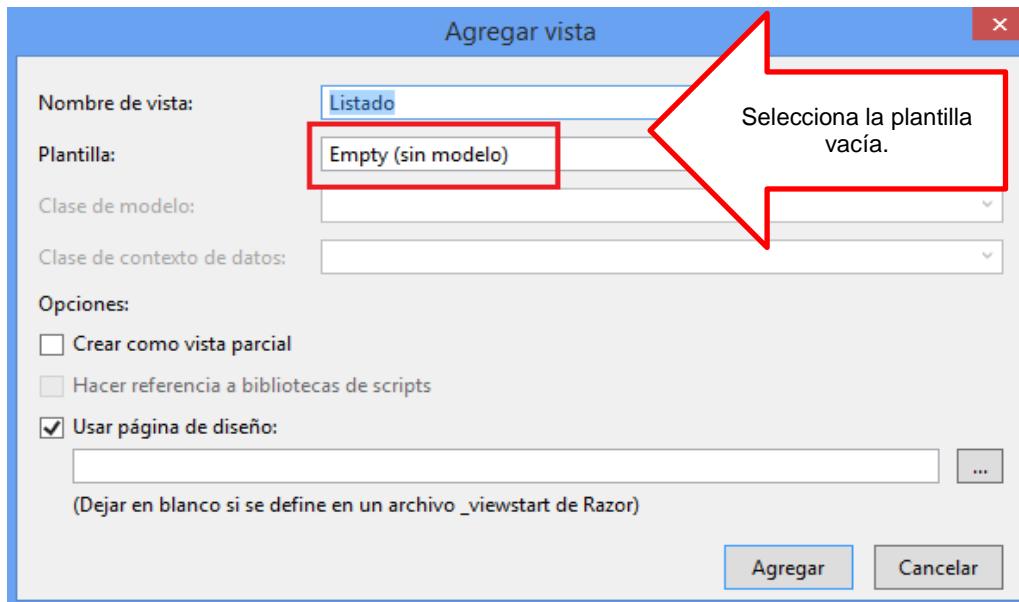
PrincipalController.cs
-----
@using System;
@using System.Collections.Generic;
@using System.Linq;
@using System.Web;
@using System.Web.Mvc;
@using Laboratorio02.Models;

namespace Laboratorio02.Controllers
{
    public class PrincipalController : Controller
    {
        // definir la lista de Producto
        static List<Producto> Inventario = new List<Producto>();

        public ActionResult Listado()
        {
            ViewData["lista"] = Inventario.ToList();
            return View();
        }
    }
}

```

A continuación agregamos la Vista al Action Listado. Observe que la plantilla debe ser vacía (Empty). Presiona el botón Agregar



En la vista, primero importamos la carpeta Models, porque utilizamos la clase Producto en el proceso del Listado.

A continuación declaramos variables en el proceso: productos el cual recibe la lista almacenada en el ViewBag; una variable st (sub total) y una variable c de tipo enter

```

Listado.cshtml* ✘ X PrincipalController.cs
@using Laboratorio02.Models
@{
    ViewBag.Title = "Listado";
    var productos = (List<Producto>)ViewData["lista"];
    double st = 0;
    int c = 0;
}

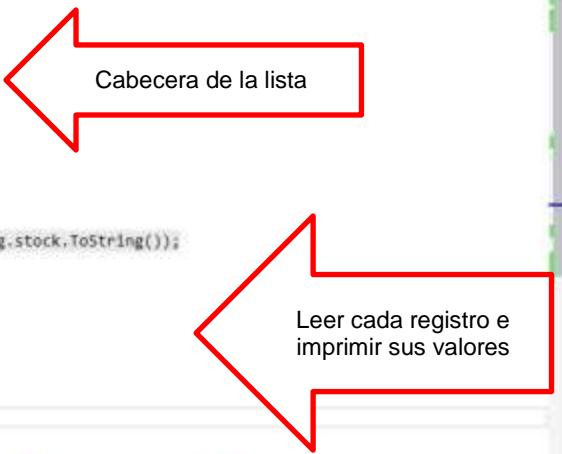


## Listado de Productos


@Html.ActionLink("Nuevo Registro", "Nuevo")

```

A continuación implementamos el proceso del listado a través de una etiqueta `<table>`, tal como se muestra. Para leer cada registro utilizamos el comando foreach para visualizar cada registro.



```

Listado.cshtml  *  PrincipalController.cs

<h2>Listado de Productos</h2>
@Html.ActionLink("Nuevo Registro", "Nuevo")

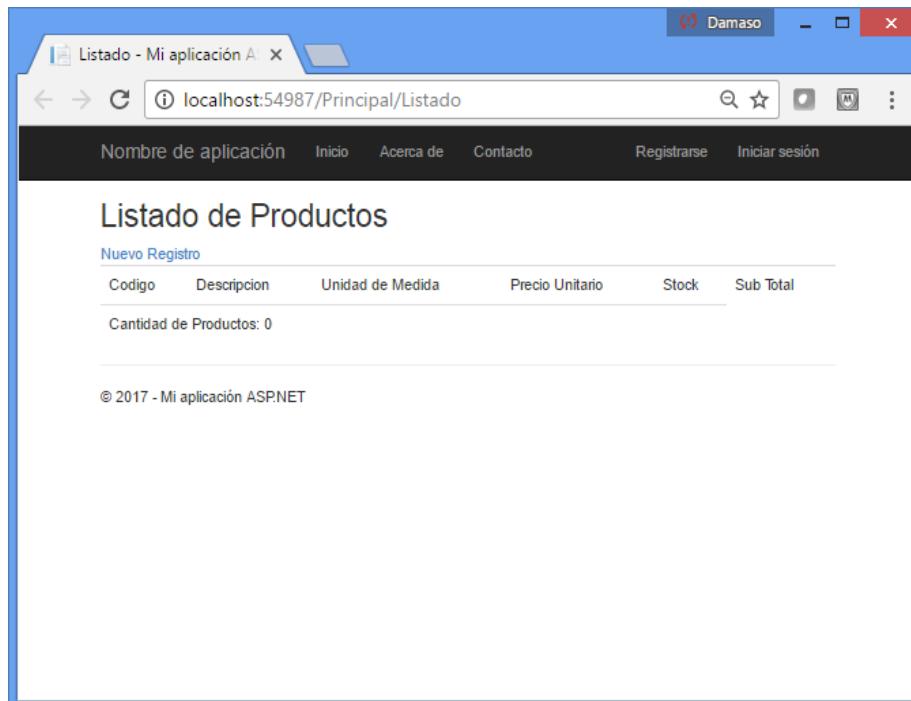


|        |             |                  |                 |       |           |
|--------|-------------|------------------|-----------------|-------|-----------|
| Código | Descripción | Unidad de Medida | Precio Unitario | Stock | Sub Total |
|--------|-------------|------------------|-----------------|-------|-----------|


@foreach(var reg in productos)
{
    st = reg.precioUni * double.Parse(reg.stock.ToString());
    <tr>
        <td>@reg.codigo</td>
        <td>@reg.descripcion</td>
        <td>@reg.unidad</td>
        <td>@reg.precioUni</td>
        <td>@reg.stock</td>
        <td>@st</td>
    </tr>
}
<tr>
    <td colspan="5">Cantidad de Productos: @productos.Count()</td>
</tr>

```

Guarde el proyecto y ejecuta la pagina Ctrl + F5, donde se visualiza la pagina en blanco



A continuación defina el ActionResult Nuevo, para enviar un Registro y recibir el registro para almacenar en la colección, tal como se muestra

```

PrincipalController.cs
namespace Laboratorio02.Controllers
{
    public class PrincipalController : Controller
    {
        // definir la lista de Producto
        static List<Producto> Inventario = new List<Producto>();

        public ActionResult Listado()
        {
            return View(new Producto());
        }

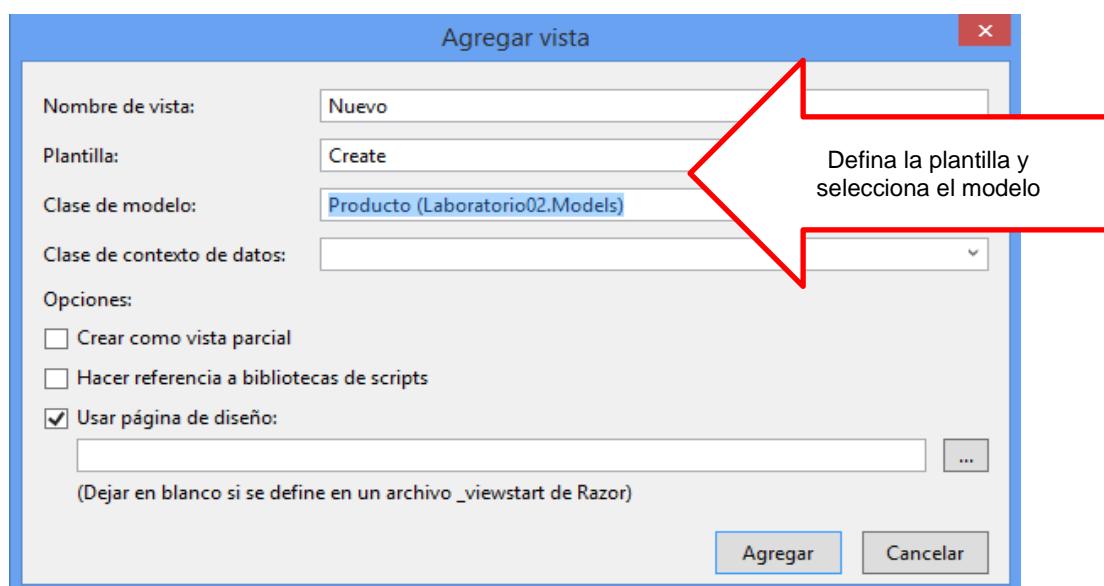
        [HttpPost]
        public ActionResult Nuevo(Producto reg)
        {
            if (!ModelState.IsValid)
            {
                return View(reg);
            }
            Inventario.Add(reg);
            return RedirectToAction("Listado");
        }
    }
}

```

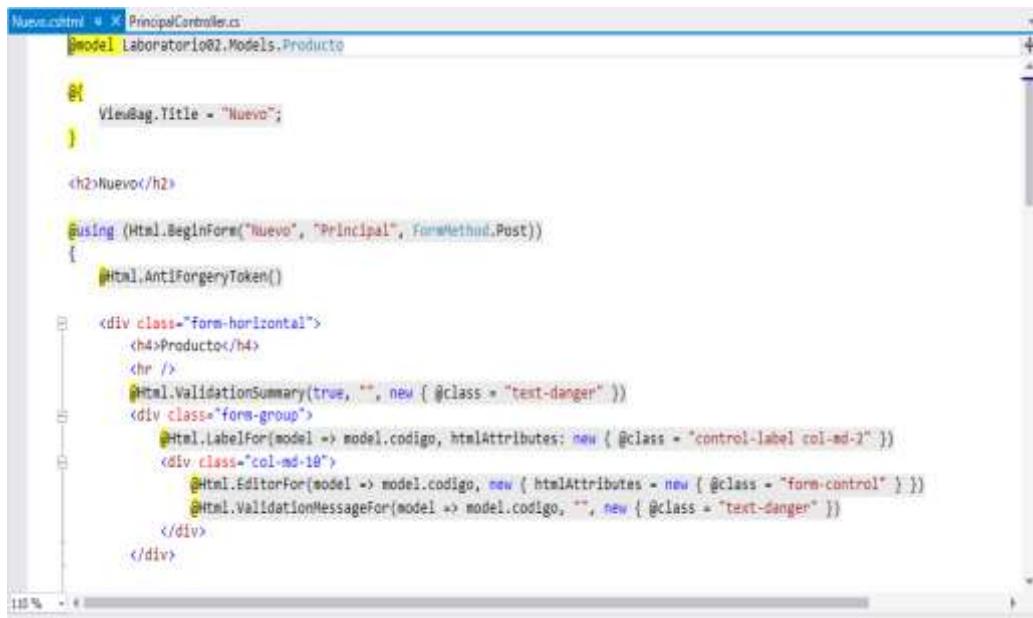
Action que envía un nuevo Producto

Action que recibe la instancia de Producto y lo almacena en la colección

A continuación defina su vista: su plantilla es Create y el modelo es Producto



Vista de la página Nuevo.cshtml, tal como se muestra.



```
Nuevo.cshtml  X PrincipalController.cs
@model Laboratorio02.Models.Producto

@{
    ViewBag.Title = "Nuevo";
}



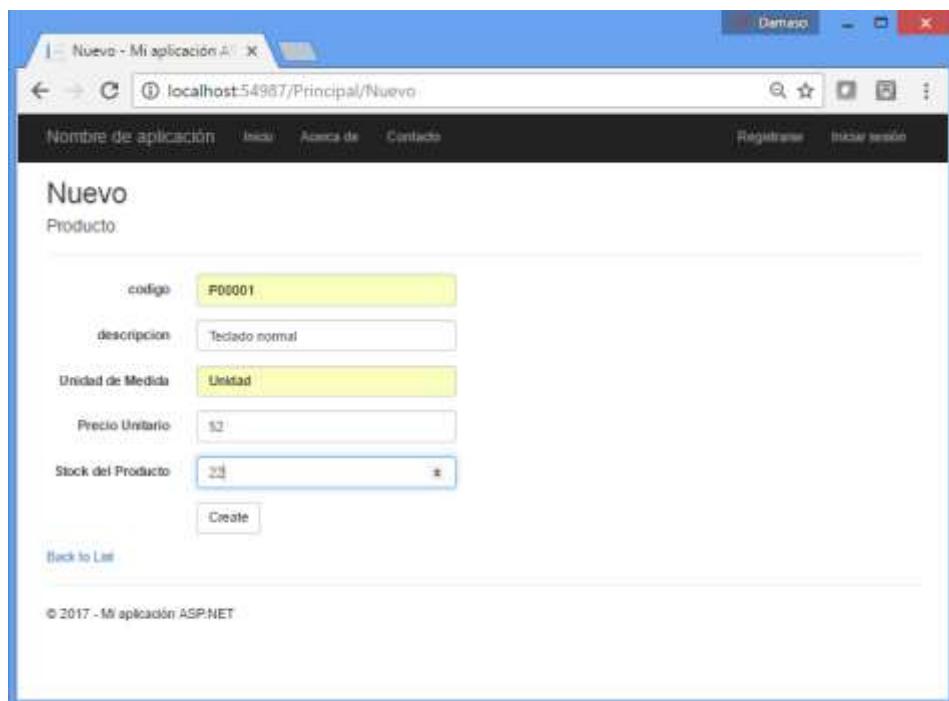
## Nuevo



@using (Html.BeginForm("Nuevo", "Principal", FormMethod.Post))
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Producto</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.codigo, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.codigo, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.codigo, "", new { @class = "text-danger" })
            </div>
        </div>
    </div>
}
```

Guarde la Vista, para ejecutar presiona Ctrl + F5, ingrese los datos, al presionar el botón Create nos direccionamos a la pagina Listado, visualizando el registro agregado



# Resumen

- El modelo–vista–controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.
- Entre las características más destacables de ASP.NET MVC tenemos las siguientes:
  - El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).
  - El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos).
  - La Vista: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.
- ASP.NET MVC es la plataforma de desarrollo web de Microsoft basada en el conocido patrón Modelo-Vista-Controlador. Está incluida en Visual Studio y aporta interesantes características a la colección de herramientas del programador Web.
- Cuando creamos una aplicación ASP.NET MVC se define una tabla de enrutamiento que se encarga de decidir que controlador gestiona cada petición Web basándose en la URL de dicha petición.
- En el modelo Model-View-Controller (MVC), las vistas están pensadas exclusivamente para encapsular la lógica de presentación. No deben contener lógica de aplicación ni código de recuperación de base de datos. El controlador debe administrar toda la lógica de aplicación. Una vista representa la interfaz de usuario adecuada usando los datos que recibe del controlador.
- Scaffolding implica la creación de plantillas a través de los elementos del proyecto a través de un método automatizado.
- Razor es una sintaxis basada en C# que permite usarse como motor de programación en las vistas de nuestros controladores. No es el único motor para trabajar con ASP.NET MVC.
- El espacio de nombres System.ComponentModel.DataAnnotations, nos proporciona una serie de clases, atributos y métodos para validar dentro de nuestros programas en .NET.
- El marco de ASP.NET MVC asigna direcciones URL a las clases a las que se hace referencia como **controladores**. Los controladores procesan solicitudes entrantes, controlan los datos proporcionados por el usuario y las interacciones y ejecutan la lógica de la aplicación adecuada.
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
  - <http://www.vitaminasdev.com/Recursos/8/asp-net-mvc-vs-asp-net-webforms>
  - [http://librosweb.es/libro/jobeet\\_1\\_4/capitulo\\_4/la\\_arquitectura\\_mvc.html](http://librosweb.es/libro/jobeet_1_4/capitulo_4/la_arquitectura_mvc.html)
  - <http://www.devjoker.com/contenidos/articulos/525/Patron-MVC-Modelo-Vista-Controlador.aspx>
  - [https://msdn.microsoft.com/es-es/library/dd381412\(v=vs.108\).aspx](https://msdn.microsoft.com/es-es/library/dd381412(v=vs.108).aspx)
  - <https://amatiasbaldi.wordpress.com/2012/05/16/microsoft-mvc4/>
  - <http://www.forosdelweb.com/f179/jquery-c-mvc-crear-objeto-json-enviarlo-action-del-controlador-1082637/>
  - <http://www.mug-it.org.ar/343016-Comunicando-cliente-y-servidor-con-jQuery-en-ASPnet-MVC3.note.aspx>
  - <https://danielggarcia.wordpress.com/2013/11/12/el-controlador-en-asp-net-mvc-4-i-enrutado/>
  - <http://www.desarrolloweb.com/articulos/pasar-datos-controladores-vistas-dotnet.html>

UNIDAD DE  
APRENDIZAJE  
**2**

# TRABAJANDO CON DATOS EN ASP.NET MVC

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla aplicaciones con acceso a datos teniendo implementando procesos de consulta y actualización de datos

## TEMARIO

### Tema 3: Interacción con el modelo de datos (9 horas)

- 3.1 La clase DBContext
- 3.2 ADO.NET Entity FrameWork
- 3.3 Expresiones Lambda con Entity FrameWork
- 3.4 Ejecutando consultas, uso de parámetros
- 3.5 Realizando mantenimiento del modelo Entity Framework

## ACTIVIDADES PROPUESTAS

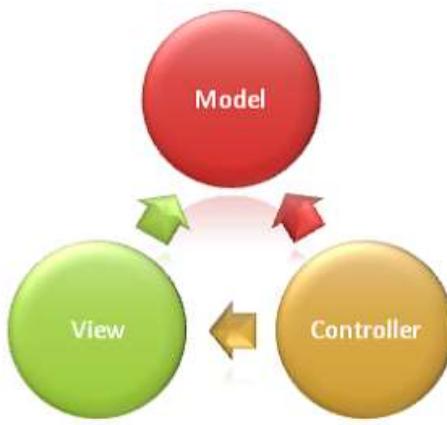
- Los alumnos implementan un proyecto web utilizando en patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



### 3. INTERACCION CON EL MODELO DE DATOS

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software el cual separa la capa de datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

El modelo MVC está compuesto de tres componentes que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.



#### El modelo de datos (Data Model).

Los objetos en el modelo de datos representan clases que interactúan con una base de datos. Normalmente, se puede pensar en el modelo de datos como el conjunto de las clases creadas por herramientas como Entity Framework (EF). Estas clases pueden comenzar ya sea en forma de tablas existentes en la base de datos que hay que leer para generar clases (enfoque database-first) o empezar como clases que se utilizarán para generar tablas de la base de datos (enfoque code-first). Además, puede crear manualmente estas clases utilizando ADO.NET para implementar la base de datos de interacción particular que necesita su aplicación.

#### 3.1 La clase DbContext

El Entity Framework le permite consultar, insertar, actualizar y eliminar datos, utilizando Common Language Runtime (CLR) objetos (conocido como entidades). El Entity Framework mapas de las entidades y las relaciones que se definen en el modelo a una base de datos. El Entity Framework proporciona facilidades para hacer lo siguiente:

- a. materializarse datos devueltos desde la base de datos como objetos de entidad;
- b. seguimiento de los cambios que se hicieron a los objetos; manejar la concurrencia;
- c. propagar los cambios de objeto de nuevo a la base de datos; y enlazar objetos con los controles.

La clase principal que es responsable de la interacción con datos como objetos es **System.Data.Entity.DbContext** (a menudo referido como contexto). La clase de contexto gestiona la entidad objetos en tiempo de ejecución, que incluye llenar objetos con datos de una base de datos, el control de cambios, y la persistencia de los datos a la base de datos.

Una instancia de DbContext representa una combinación de los modelos de unidad de trabajo y repositorio, de modo que pueda emplearse para consultar una base de datos y agrupar los cambios que, seguidamente, se volverán a escribir en el almacenamiento como una unidad. DbContext es conceptualmente similar a ObjectContext.

```

namespace MVC_Negocios2014
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class Negocios2014Entities : DbContext
    {
        public Negocios2014Entities()
            : base("name=Negocios2014Entities")
        {

        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public DbSet<tb_clientes> tb_clientes { get; set; }
        public DbSet<tb_paises> tb_paises { get; set; }
        public DbSet<tb_pedidoscabe> tb_pedidoscabe { get; set; }
        public DbSet<tb_pedidosdeta> tb_pedidosdeta { get; set; }
    }
}

```

DbContext se usa generalmente con un tipo derivado que contiene DbSet < TEntity > propiedades de las entidades fundamentales del modelo. Estos conjuntos se inicializan automáticamente cuando se crea la instancia de la clase derivada.

#### Definición de la clase derivada DbContext

El método recomendado para trabajar con el contexto es definir una clase que deriva de DbContext y expone DbSet propiedades que representan las colecciones de las entidades especificadas en el contexto. Si está trabajando con el Diseñador del Entity Framework, el contexto se generará automáticamente. Si está trabajando con el Código Primero, normalmente escribe el contexto mismo.



DbContext es responsable de las siguientes actividades:

1. EntitySet: DbContext contiene un conjunto de entidades (DbSet < TEntity >) las cuales están referenciadas a un origen de datos.
2. Consulta: DbContext convierte LINQ-to-Entities (LINQ a Entidades) ejecuta las consulta SQL y enviarlo a la base de datos.
3. Control de seguimiento: Se realiza un seguimiento de los cambios que se produjeron en las entidades después de que ha sido la consulta de la base de datos.

4. La persistencia de datos: Se realiza también la inserción, actualización y supresión de la base de datos, en base a lo que dice la entidad.
5. El almacenamiento en caché: DbContext hace caché de primer nivel por defecto. Almacena las entidades que han sido recuperados durante el tiempo de vida de una clase de contexto.
6. Administrar relaciones: DbContext también gestiona la relación utilizando CSDL, MSL y SSDL en DB-First o el enfoque o el uso de la API de fluidez (es una forma avanzada de especificar la funcionalidad de la configuración de DataAnnotations) en Code-First aproximación Model-First.
7. Objeto Materialización: DbContext convierte los datos, los cuales están almacenadas en un origen de datos, en tablas de objetos de entidad.

Para definir una clase DbContext, primero importar la librería System.Data.Entity A la clase Negocios2014DB extienda a tipo DbContext. A continuación defina un DbSet llamado tb\_materiales de tipo <tb\_material> tal como se muestra. Defina el método OnModelCreating para no pluralizar los nombres de las clases y sus tablas.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
namespace Mvc_Mantenimiento.Models
{
    public class Negocios2014DB : DbContext
    {
        public DbSet<tb_material> tb_materiales { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }
}

```

Por defecto, el contexto gestiona las conexiones a la base de datos. El contexto se abre y cierra las conexiones según sea necesario. Por ejemplo, el contexto se abre una conexión para ejecutar una consulta y, a continuación, cierra la conexión cuando se hayan procesado todos los conjuntos de resultados.

Hay casos en los que desea tener más control sobre cuando la conexión se abre y se cierra. Por ejemplo, cuando se trabaja con SQL Server, apertura y cierre de la misma conexión es caro. Puede administrar este proceso manualmente mediante la propiedad de conexión.

### 3.2 ADO.NET Entity Framework

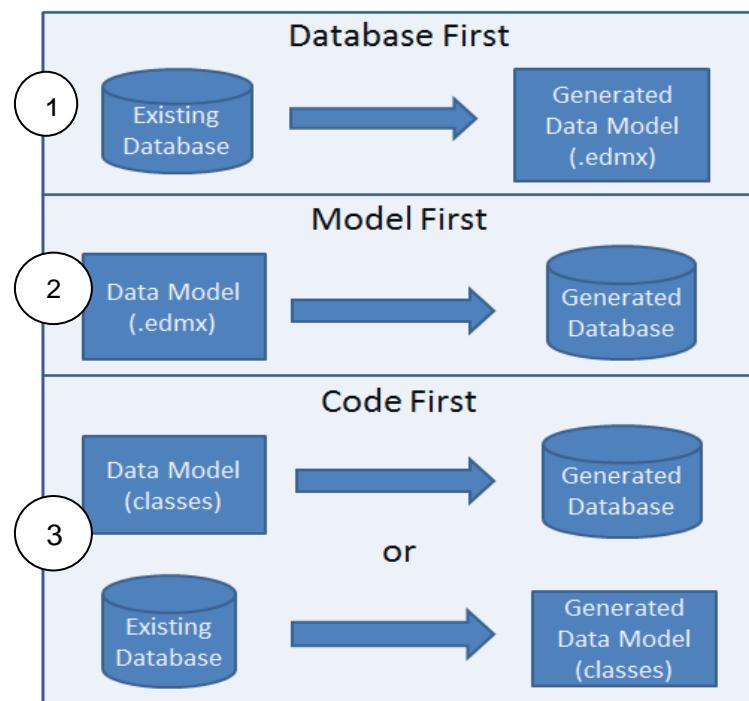
Entity Framework es un conjunto de herramientas incluidas en ADO.NET que dan soporte para el Desarrollo de Aplicaciones orientadas a datos. Arquitectos y desarrolladores de aplicaciones orientadas a datos se debaten con la necesidad de realizar dos diferentes objetivos. Por un lado deben modelar las entidades, sus relaciones y los problemas de lógica de negocio que deben implementar, y por otro lado deben trabajar con los motores de base de datos para almacenar y recuperar la información. La data puede usar múltiples orígenes de datos y cada uno de ellos puede trabajar con sus propios protocolos de comunicación.

Entity Framework permite crear aplicaciones de acceso a datos programando con un modelo de aplicaciones conceptuales en lugar de programar directamente con un esquema de almacenamiento relacional. El objetivo es reducir la cantidad de código y el

mantenimiento necesarios para las aplicaciones orientadas a datos. Las aplicaciones de Entity Framework ofrecen las siguientes ventajas:

- Las aplicaciones pueden funcionar en términos de un modelo conceptual más centrado en la aplicación, que incluye tipos con herencia, miembros complejos y relaciones.
- Las aplicaciones están libres de dependencias de codificación rígida de un motor de datos o de un esquema de almacenamiento.
- Las asignaciones entre el modelo conceptual y el esquema específico de almacenamiento pueden cambiar sin tener que cambiar el código de la aplicación.
- Los desarrolladores pueden trabajar con un modelo de objeto de aplicación coherente que se puede asignar a diversos esquemas de almacenamiento, posiblemente implementados en sistemas de administración de base de datos diferentes.
- Se pueden asignar varios modelos conceptuales a un único esquema de almacenamiento.
- La compatibilidad con Language Integrated Query (LINQ) proporciona validación de la sintaxis en el momento de la compilación para consultas en un modelo conceptual.

La siguiente figura muestra las opciones disponibles en Entity Framework para poder trabajar:



- I. Database First: en este modo de trabajo se puede generar el modelo de datos a partir de una base datos existente
- II. Model First: en este modo de trabajo se puede generar la base de datos a partir del modelo visual generado en Visual Studio (modelo .edmx)
- III. Code First: en este modo de trabajo se puede generar la base de datos a partir de clases básicas conocidas como POCO (Plain Old CLR Object), que son objetos que ignoran la persistencia y viceversa, se puede a partir de la base de datos generar las clases POCO.

### 3.3 Expresiones Lambda con Entity FrameWork

Una expresión es un fragmento de código que se puede evaluar como un valor, objeto, método o espacio de nombres único. Las expresiones pueden contener un valor literal, una llamada a un método, un operador y sus operandos, o un nombre simple. Los nombres simples pueden ser el nombre de una variable, el miembro de un tipo, el parámetro de un método, un espacio de nombres o un tipo.

Las expresiones pueden utilizar operadores que a su vez utilizan otras expresiones como parámetros, o llamadas a métodos cuyos parámetros son a su vez otras llamadas a métodos. Por consiguiente, las expresiones pueden ser de muy simples a muy complejas.

En las consultas de LINQ to Entities, las expresiones pueden contener cualquier elemento permitido por los tipos dentro del espacio de nombres **System.Linq.Expressions**, incluyéndose las expresiones lambda. Las expresiones que se pueden utilizar en las consultas de LINQ to Entities son un supraconjunto de las expresiones que se pueden utilizar para consultar Entity Framework. Las expresiones que forman parte de consultas en Entity Framework están limitadas por las operaciones admitidas por **ObjectQuery<T>** y el origen de datos subyacente.

Las **expresiones lambda** son funciones anónimas que devuelven un valor. Se pueden utilizar en cualquier lugar en el que se pueda utilizar un delegado. Son de gran utilidad para escribir métodos breves que se pueden pasar como parámetro.

Las expresiones lambda se declaran en el momento en el que se van a usar como parámetro para una función.

#### **Arboles de Expresión**

Las expresiones lambda vistas en el punto anterior se utilizan exclusivamente como funciones. Hay otra manera de utilizarlas que resulta de gran utilidad y es ampliamente usada en el ORM de la casa, Entity Framework.

En las expresiones lambda vistas hasta ahora, el compilador genera una función ejecutable que se permite invocar desde la parte del código en que necesitemos dicha función de una forma directa. Cuando construimos un árbol de expresión no se genera una función ejecutable, sino una estructura de datos en forma de árbol donde cada uno de los nodos es una expresión, almacenando las operaciones definidas, por ejemplo  $x == y$ .

Este almacén tiene gran utilidad, ya que nos permite enviar el árbol entre capas, recorrerlo, analizarlo, serializarlo y llegado el caso compilarlo y ejecutarlo. La potencia de este tipo se aprecia perfectamente, y como comentaba anteriormente, en Entity Framework, ya que se pueden definir consultas de recuperación de datos en una capa superior y evitar su compilación y ejecución hasta que llegue a la capa de acceso a datos.

Las expresiones lambda son utilizadas ampliamente en Entity Framework. Cuando realizamos una consulta del tipo

```
var consulta = post.Where(p => p.User == "admin");
```

La cláusula **Where** tiene como parámetro una expresión lambda. La operación descrita para recuperar los posts del usuario admin se almacenará como una estructura de datos que podemos pasar entre capas como parámetro de métodos. Esta expresión la analizará y ejecutará el proveedor de datos que se está usando y la convertirá en código SQL para realizar la consulta contra la base de datos.

A continuación se muestra una pequeña lista de operadores de consulta estándar que se definen en **System.Linq.Queryable**

Operador	Descripción	Ejemplo
SingleOrDefault	Devuelve solo un elemento que satisface la condición o un valor por default, este método emitirá una exception si existe más de un element que satisface la condición	<pre>NegociosDataContext MyDB=new NegociosDataContext(); var item = MyDB.listaCart.SingleOrDefault(c =&gt; c.Identificador == ShoppingCartId &amp;&amp; c.AlbumId == album.AlbumId);</pre>
Single	Devuelve solo un elemento que satisface la condición , este método emitirá una exception si existe más de un element que satisface la condición o si ningún elemento satisface la condición	<pre>NegociosDataContext MyDB = new NegociosDataContext(); var item = MyDB.listaAlbum.Single(a =&gt; a.AlbumId == id);</pre>
Skip	Omite un numero específico de elementos y devuelve los elementos restantes	<pre>int[] grados = { 59, 82, 70, 56, 92, 98, 85 };  IQueryable&lt;int&gt; bajoGrades = grados.AsQueryable().OrderByDescending(g =&gt; g).Skip(3);</pre>
Take	Devuelve un específico número de elementos contiguos desde un punto de inicio, en una colección o secuencia, es análogo al operador TOP de SQL	<pre>int[] grados = { 59, 82, 70, 56, 92, 98, 85 };  IQueryable&lt;int&gt; topTresGrades = grades.AsQueryable().OrderByDescending(grade =&gt; grade).Take(3);</pre>
Any	Devuelve un valor bool indicando si algún elemento de una determinada colección, satisface la condición	<pre>Pet[] pets = { new Pet { Name="Barley", Age=8, Vacuna=true },   new Pet { Name="Boots", Age=4, Vacuna=false },   new Pet { Name="Whiskers", Age=1, Vacuna=false } };  bool unvaccinated = pets.AsQueryable().Any(p =&gt; p.Age &gt; 1 &amp;&amp; p.Vaccinated == false);</pre>

### 3.4 Ejecutando consultas, uso de parámetros

La mayoría de las consultas en la documentación introductoria query language Integrated query (LINQ) se escriben utilizando la sintaxis declarativa de la consulta LINQ. Sin embargo, la sintaxis de la consulta se debe traducir en llamadas a métodos para Common Language Runtime (CLR) .NET cuando se compila el código. Estas llamadas a métodos invocan los operadores de consulta estándar, que tienen nombres como **Where**, **Select**, **GroupBy**, **Join**, **Max**, y **Average**. Puede llamarlas directamente con sintaxis de método en lugar de sintaxis de consulta.

La sintaxis de consulta y la sintaxis de método son semánticamente idénticas, pero muchas personas encuentran la sintaxis de consulta más sencilla y más fácil de leer. Algunas consultas deben expresarse como llamadas a método. Por ejemplo, debe utilizar una llamada al método para expresar una consulta que recupera el número de elementos que coinciden con una condición especificada. También debe utilizar una llamada al método para una consulta que recupera el elemento que tiene el valor máximo de una secuencia de origen.

En la documentación de referencia de los operadores de consulta estándar en el espacio de nombres **System.Linq** generalmente se utiliza la sintaxis de método. Por consiguiente, aunque esté empezando a escribir consultas LINQ, le resultará útil estar familiarizado con el uso de la sintaxis de método en consultas y en expresiones de consulta.

### 3.5 Realizando mantenimiento del modelo Entity Framework

Los objetos de un contexto del objeto son instancias de tipos de entidad que representan los datos en el origen de datos. Puede modificar, crear y eliminar objetos de un contexto y Entity Framework realizará el seguimiento de los cambios efectuados en estos objetos. Cuando se llama al método **SaveChanges**, Entity Framework genera y ejecuta comandos que llevan a cabo instrucciones equivalentes de inserción, actualización o eliminación con el origen de datos.

Entity Framework permite asignar las operaciones de inserción, actualización y eliminación de un tipo de entidad a procedimientos almacenados. Si está pensando en asignar procedimientos almacenados a sus entidades, es recomendable asignar las tres operaciones. Si, por ejemplo, asigna un tipo de entidad a procedimientos almacenados de inserción y actualización, pero no la asigna al procedimiento almacenado de eliminación y, a continuación, intenta eliminar un objeto de ese tipo, la operación de eliminación producirá un error en tiempo de ejecución con una excepción **UpdateException**.

#### Crear y agregar objetos

Si desea insertar datos en el origen de datos, debe crear una instancia de un tipo de entidad y agregar el objeto a un contexto del objeto. Para poder guardar un objeto nuevo en el origen de datos, primero debe establecer todas las propiedades que no admitan los valores null. Cuando trabaje con clases generadas por Entity Framework, considere el uso del método estático **CreateNombreObjeto** del tipo de entidad para crear una nueva instancia de un tipo de entidad.

Las herramientas de Entity Data Model incluyen este método en cada clase cuando generan los tipos de entidad. Este método de creación se usa para crear una instancia de un objeto y establecer las propiedades de la clase que no pueden ser null. El método incluye un parámetro para cada propiedad que tenga aplicado el atributo **Nullable="false"** en el archivo de CSDL.

#### Eliminar objetos

Al llamar al método **DeleteObject** sobre **ObjectSet**, o al método **DeleteObject** sobre **ObjectContext**, se marca el objeto especificado para su eliminación. La fila no se elimina del origen de datos hasta que se llama a **SaveChanges**. El comportamiento de eliminar objetos difiere en Entity Framework dependiendo del tipo de relación al que pertenece el objeto.

En una relación de identificación, donde una clave principal de la entidad principal forma parte de la clave principal de la entidad dependiente, al eliminar un objeto se pueden eliminar también los objetos relacionados. Los objetos dependientes no pueden existir sin una relación definida con el objeto primario. Al eliminar el objeto primario, también se eliminan todos los objetos secundarios. Esto es igual que habilitar el atributo **<OnDelete Action="Cascade" />** en la asociación para la relación.

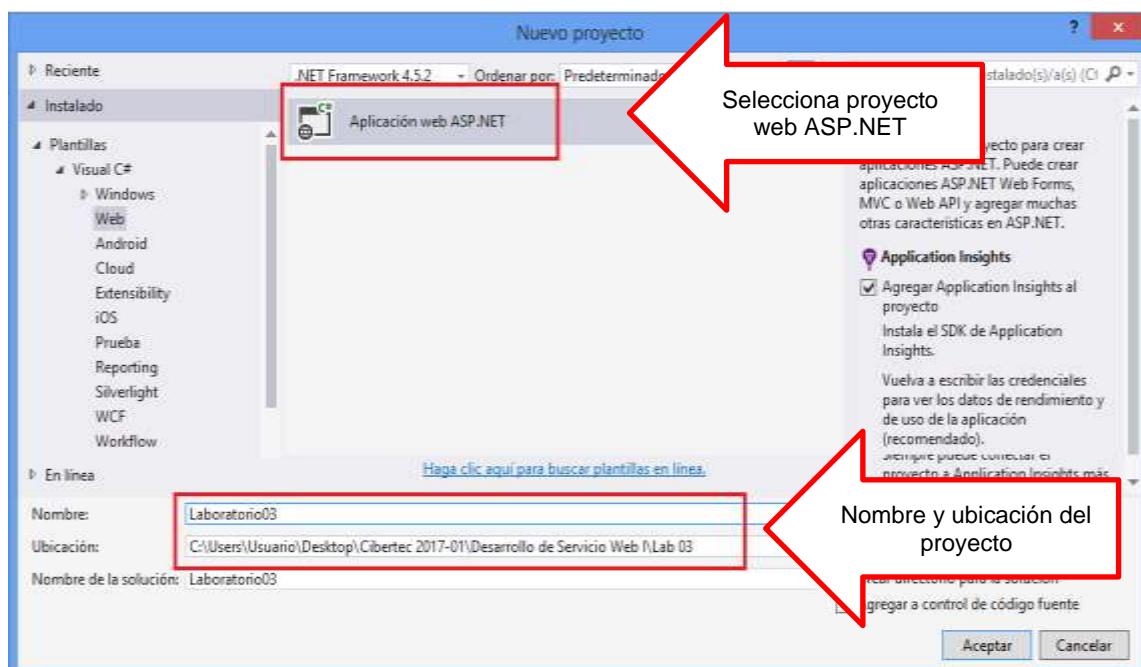
## Laboratorio 3.1

### Aplicación ASP.NET MVC con acceso a datos

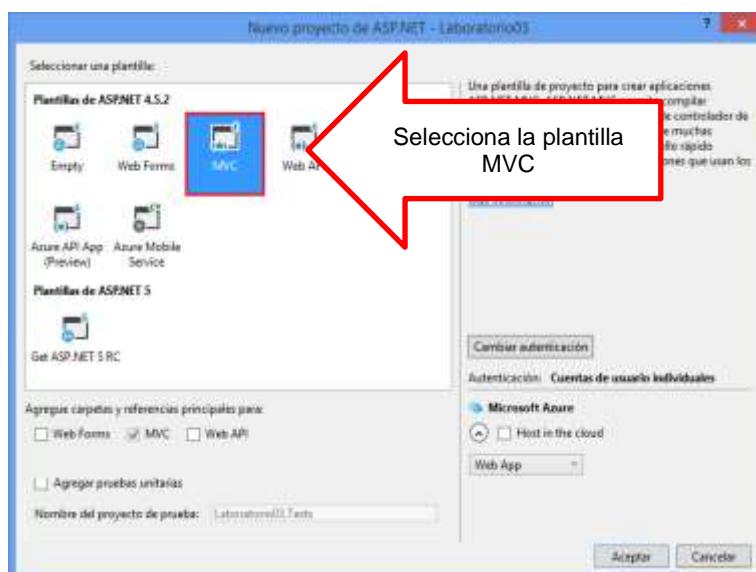
Implemente un proyecto ASP.NET MVC donde permita listar los registros de la tabla tb\_clientes, los cuales se encuentra almacenado en la base de datos negocios2014. Utilice el modelo de Contexto

#### SOLUCION

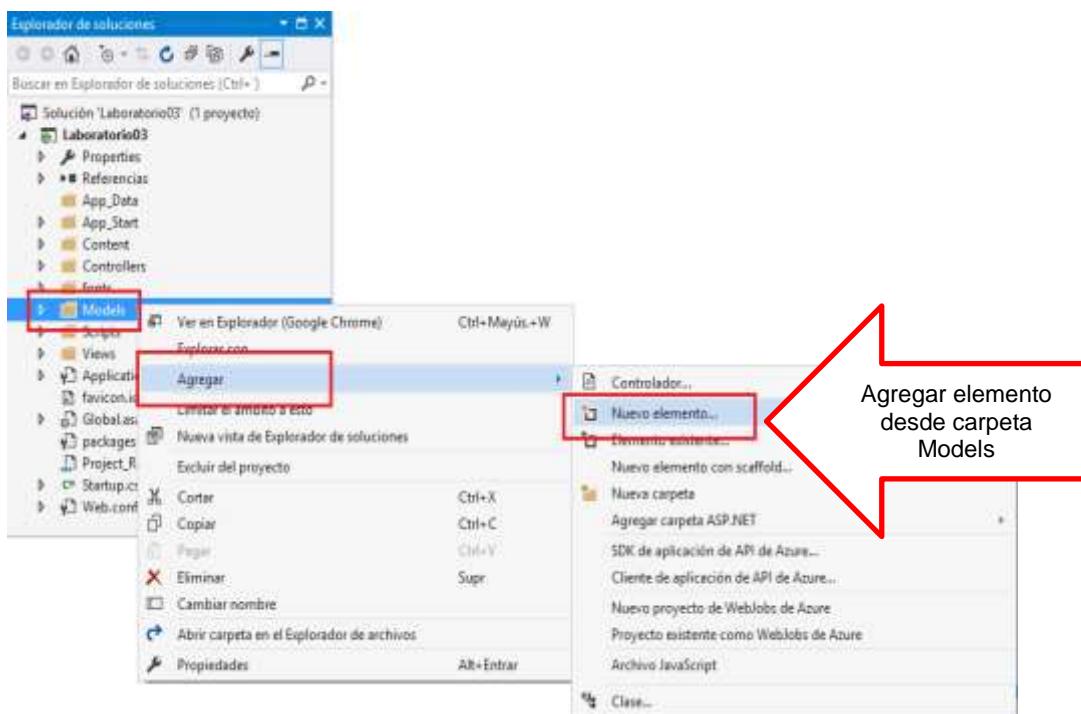
Crear un nuevo proyecto tipo Aplicación web ASP.NET, tal como se muestra.



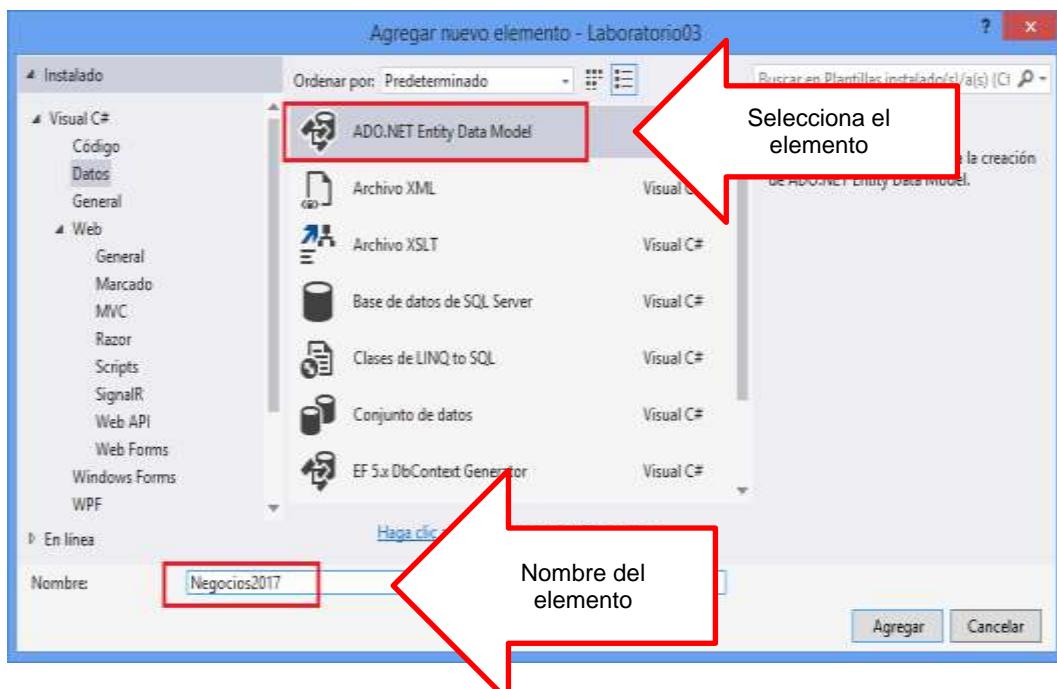
Selecciona la plantilla del proyecto web: MVC, tal como se muestra. Para continuar presionar el botón AGREGAR



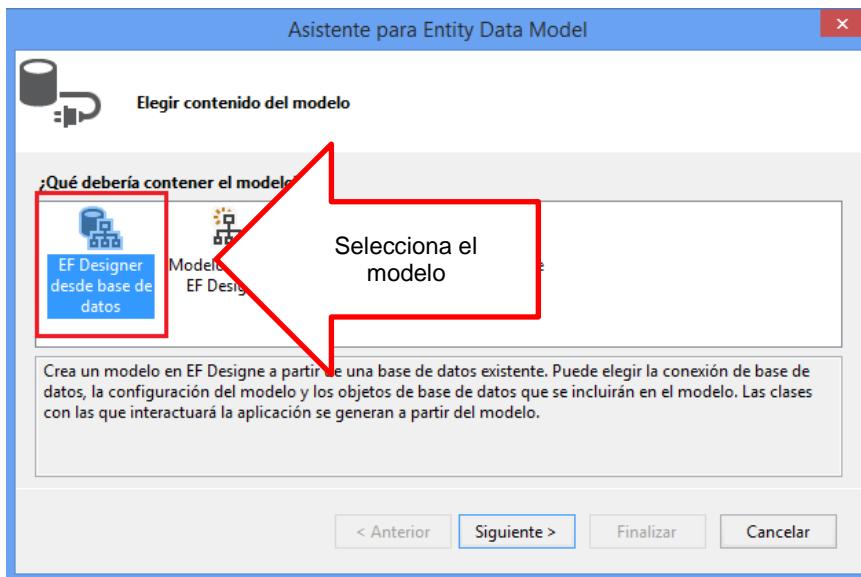
A continuación agregamos el modelo ADO Entity Model. Desde la carpeta **MODELS**, selecciona la opción **AGREGAR → Nuevo elemento**, tal como se muestra.



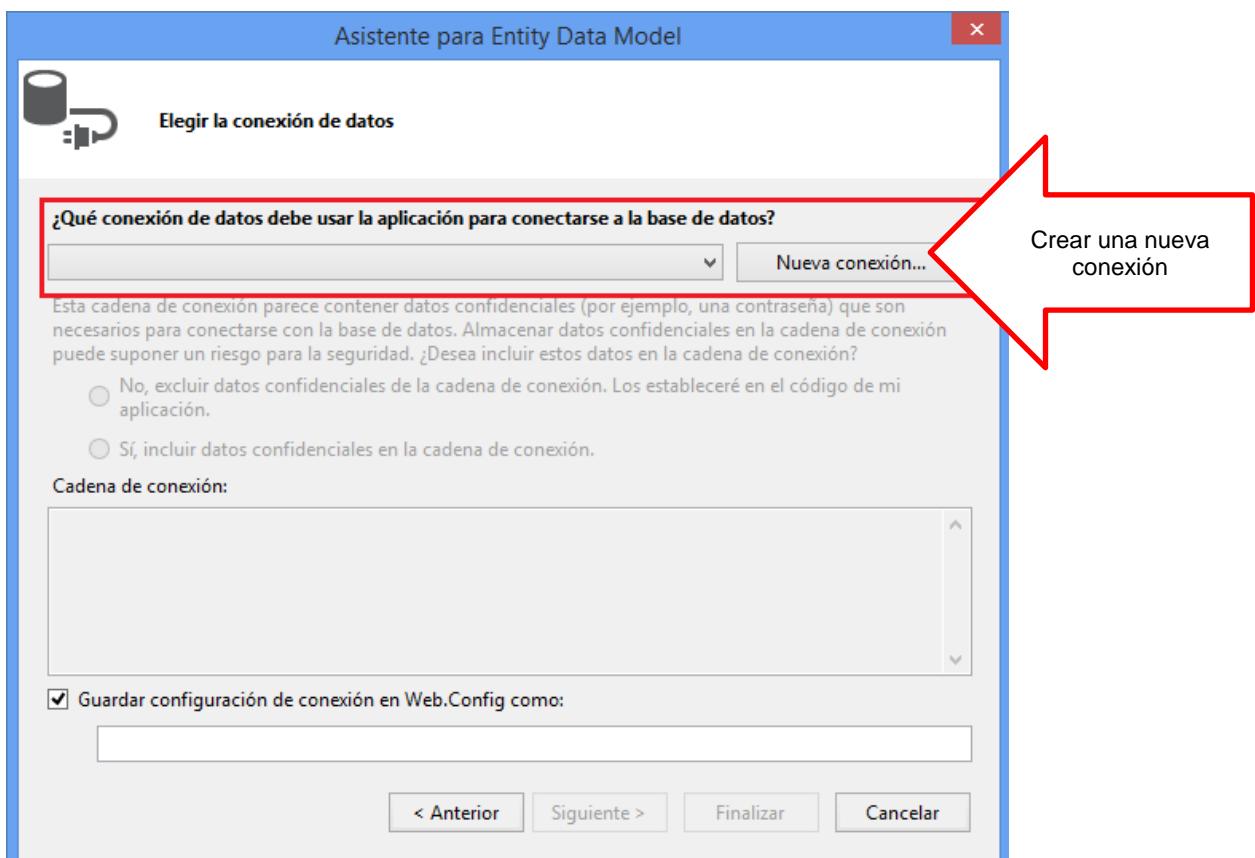
Selecciona, desde la opción Datos, el elemento ADO.NET Entity Data Model, tal como se muestra. Asigne el nombre al elemento: Negocios2017. Presiona el botón AGREGAR



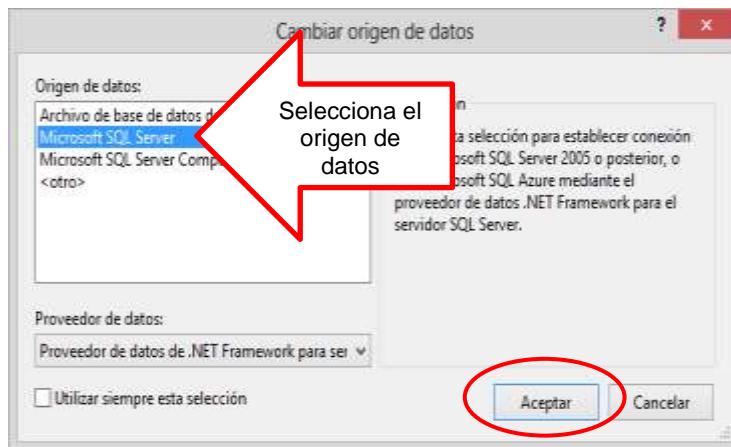
1. Elija el contenido del modelo, para ello selecciona la opción **GENERAR DESDE LA BASE DE DATOS**. Presione el botón Siguiente.



2. Elija la conexión, para crear una nueva conexión, presiona el botón **Nueva conexión**



3. Selecciona el origen de datos: **Microsoft SQL Server**, presiona el botón ACEPTAR, tal como se muestra.

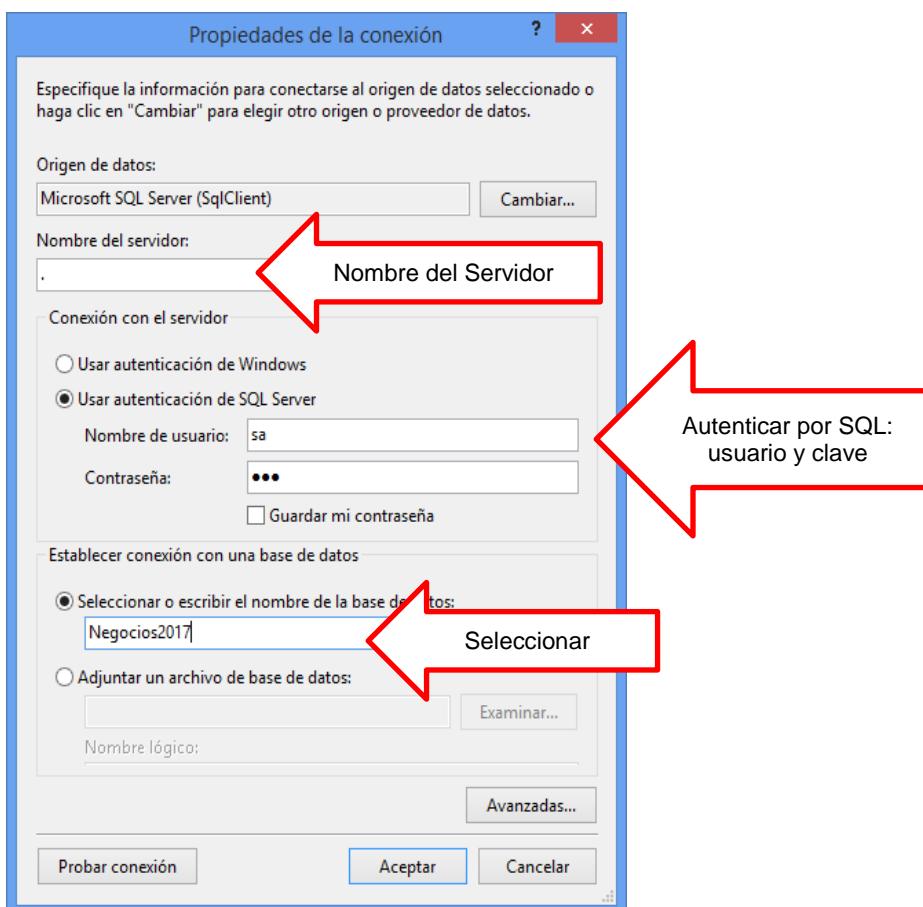


4. Defina las propiedades de la conexión:

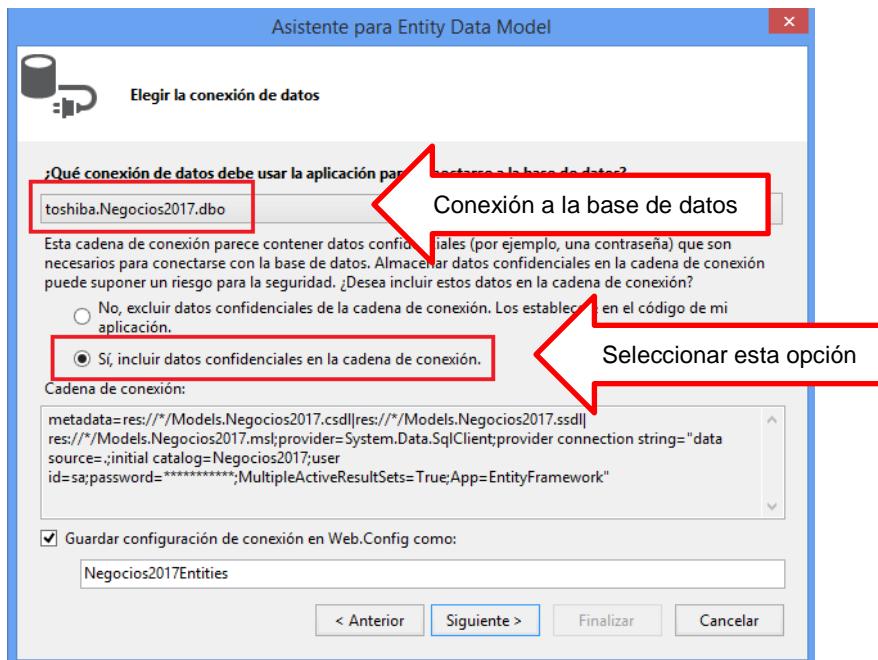
Nombre del servidor,

Autenticación, si es por SQL Server, ingrese el usuario y la contraseña.

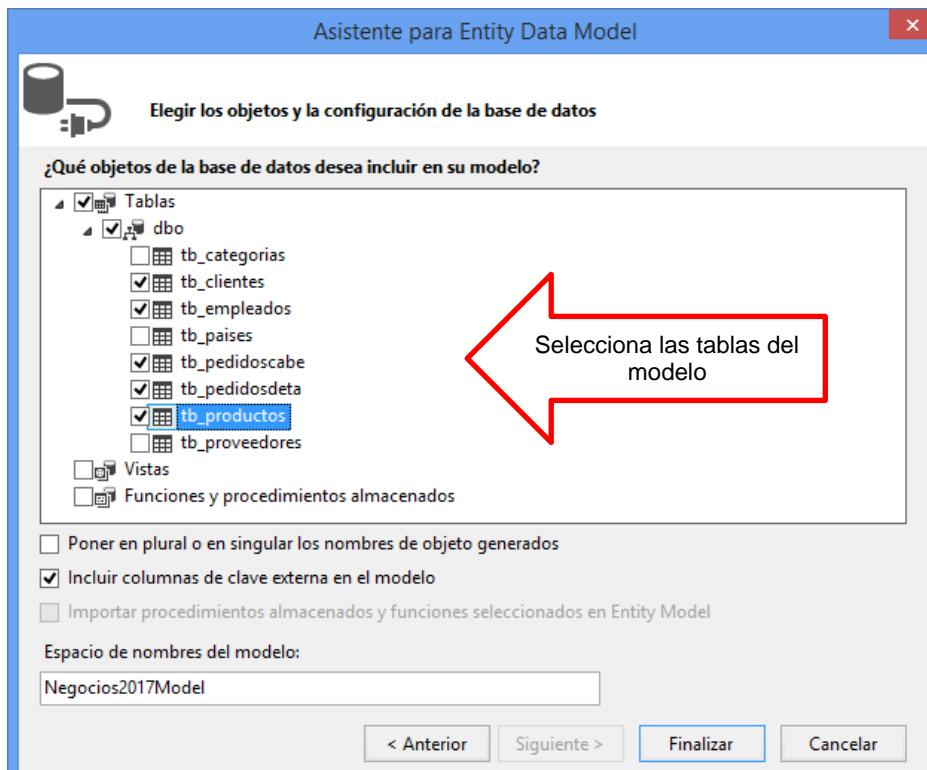
Selecciona la base de datos a trabajar



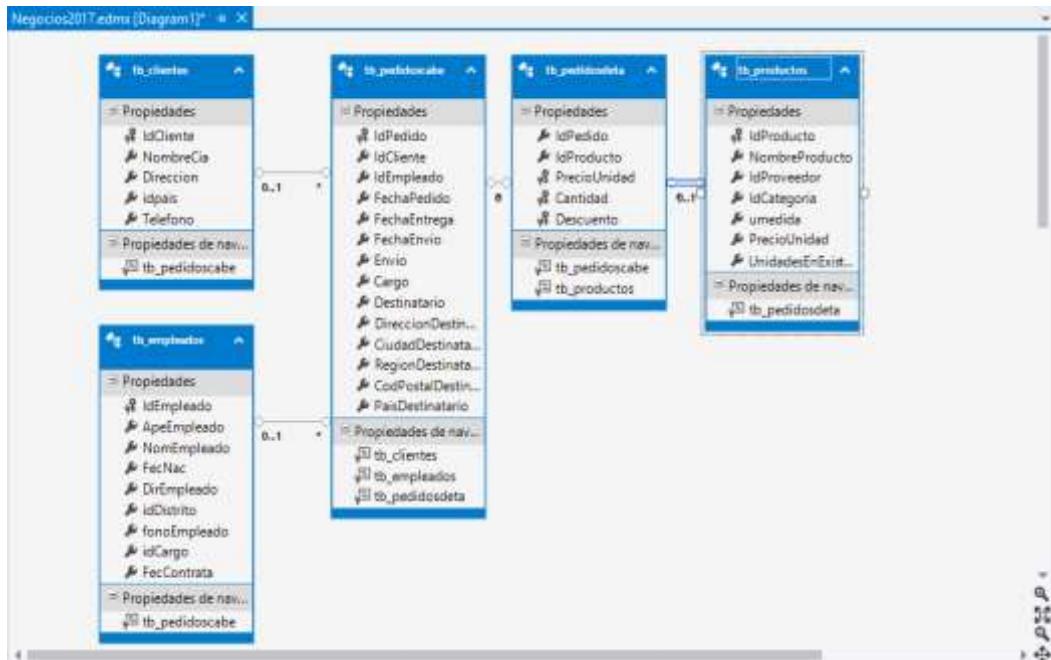
5. Habiendo definido la conexión, ésta se visualiza nueva en la ventana Elegir la conexión de datos, tal como se muestra. A continuación presiona el botón Siguiente.



6. Selecciona las tablas que incluirás en el modelo, tal como se muestra. Al terminar presiona el botón FINALIZAR.



Al finalizar se genera el Diagrama del modelo, tal como se muestra. COMPILE LA SOLUCION (presiona la combinación Ctrl + May + B), a continuación



El DbContext del modelo (Negocios2017.Context.cs), expone cada uno de los DbSet de las tablas que conforman el modelo, tal como se muestra

```

namespace Laboratorio03.Models
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

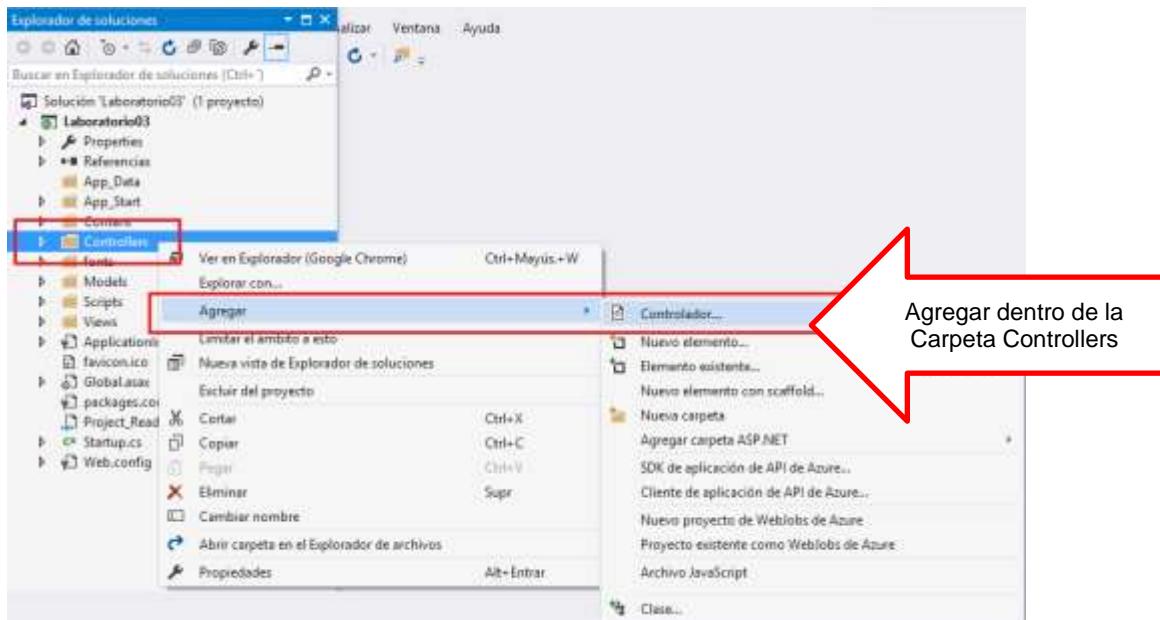
    public partial class Negocios2017Entities : DbContext
    {
        public Negocios2017Entities()
            : base("name=Negocios2017Entities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

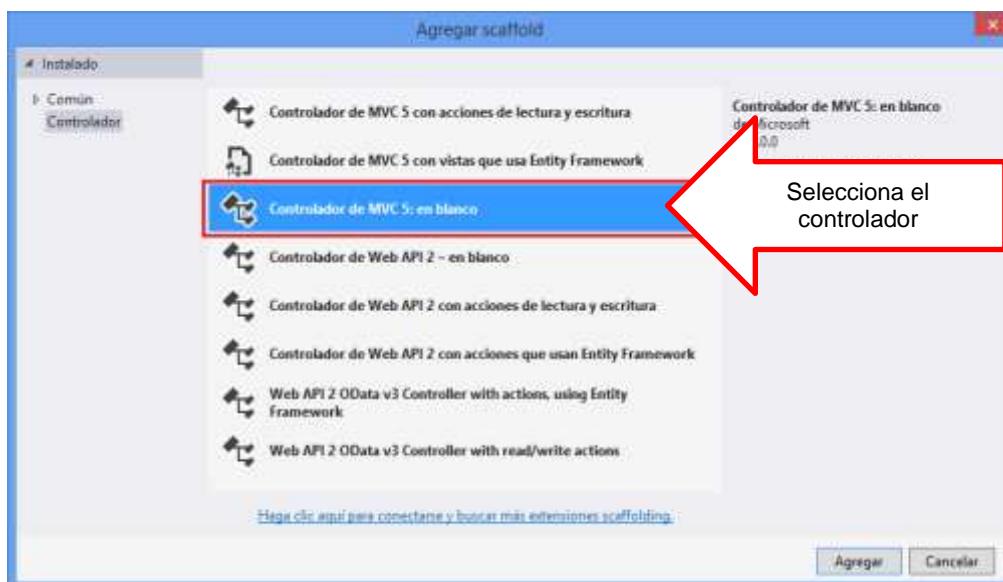
        public virtual DbSet<tb_clientes> tb_clientes { get; set; }
        public virtual DbSet<tb_empleados> tb_empleados { get; set; }
        public virtual DbSet<tb_pedidoscabe> tb_pedidoscabe { get; set; }
        public virtual DbSet<tb_productos> tb_productos { get; set; }
        public virtual DbSet<tb_pedidosdet> tb_pedidosdet { get; set; }
    }
}
  
```

## Agregando el Controlador

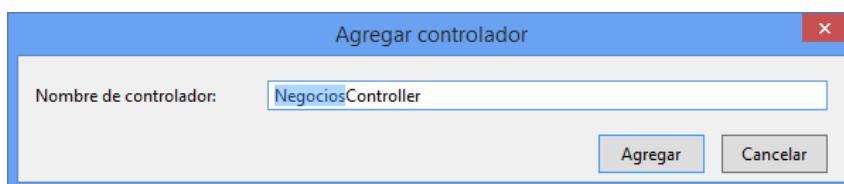
A continuación agregamos un Controlador en el proyecto, tal como se muestra en la figura



Selecciona el tipo de Controlador: en blanco, presiona el botón AGREGAR



Asigne el nombre del controlador:**NegociosController**, presiona el botón AGREGAR



En el controlador NegociosController, defina a db como instancia del Contexto Negocios2017Entities.

```

NegociosController.cs  X
Laboratorio03          Laboratorio03.Controllers.NegociosController      bd
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Laboratorio03.Models;

namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        // instancia del DBContext
        Negocios2017Entities bd = new Negocios2017Entities();

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

```

Dentro del controlador, defina el ActionResult Index(). A través de este método retornamos a la Vista, la lista de los datos de tb\_clientes

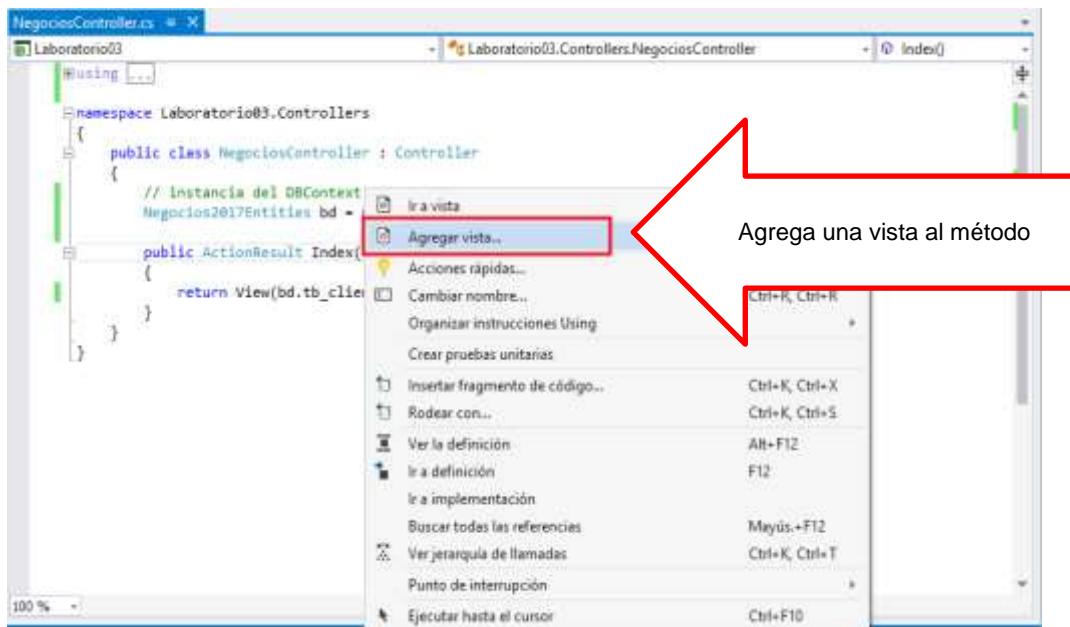
```

NegociosController.cs  X
Laboratorio03          Laboratorio03.Controllers.NegociosController      bd
using ...
namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        // instancia del DBContext
        Negocios2017Entities bd = new Negocios2017Entities();

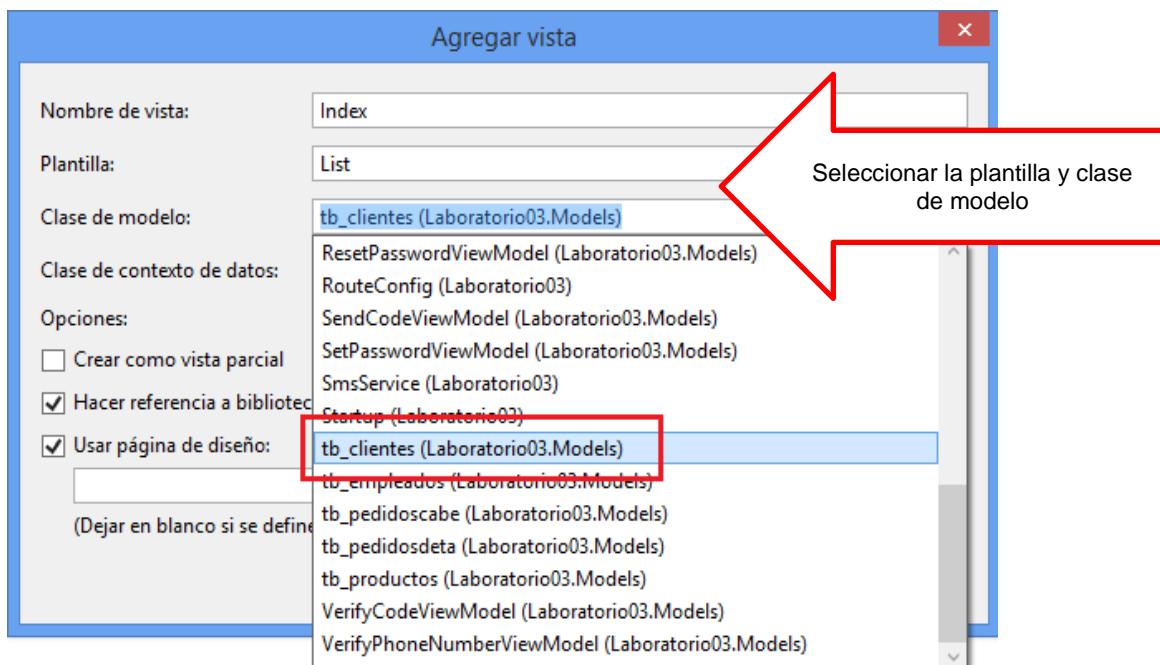
        public ActionResult Index()
        {
            return View(bd.tb_clientes.ToList());
        }
    }
}

```

Agrega una vista al ActionResult, tal como se muestra



En la ventana, selecciona la plantilla List (listado) y la clase de Modelo tb\_clientes, tal como se muestra



Generada la Vista, se imprime el siguiente código.



```

Index.cshtml  X NegociosController.cs
@model IEnumerable<Laboratorio03.Models.tb_clientes>
{
    ViewBag.Title = "Index";
}



## Index



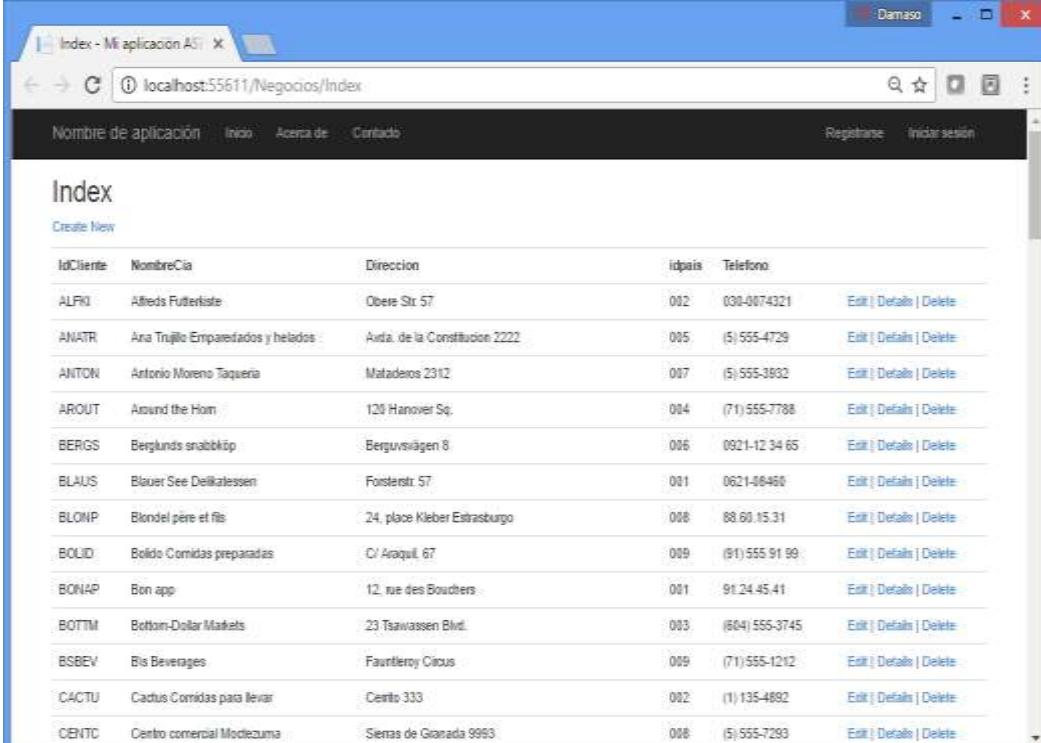
@Html.ActionLink("Create New", "Create")



| @Html.DisplayNameFor(model => model.IdCliente) | @Html.DisplayNameFor(model => model.NombreCia) | @Html.DisplayNameFor(model => model.Direccion) | @Html.DisplayNameFor(model => model.idpais) | @Html.DisplayNameFor(model => model.Telefono) |                                                                         |
|------------------------------------------------|------------------------------------------------|------------------------------------------------|---------------------------------------------|-----------------------------------------------|-------------------------------------------------------------------------|
| @Html.DisplayFor(modelItem => item.IdCliente)  | @Html.DisplayFor(modelItem => item.NombreCia)  | @Html.DisplayFor(modelItem => item.Direccion)  | @Html.DisplayFor(modelItem => item.idpais)  | @Html.DisplayFor(modelItem => item.Telefono)  | <a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a> |


```

Para ejecutar la Vista presiona la combinación Ctrl + F5, visualizando la pagina html



IdCliente	NombreCia	Direccion	idpais	Telefono	
ALFKI	Alfreds Futterkiste	Obere Str. 57	002	030-0874321	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
ANATR	Ara Trujillo Emparedados y helados	Avda. de la Constitución 2222	005	(5) 555-4729	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
ANTON	Antonio Moreno Taquería	Mataderos 2312	007	(5) 555-3932	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
AROUT	Around the Horn	120 Hanover Sq.	004	(71) 555-7788	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BERGS	Berglunds snabbköp	Berguvsvägen 8	006	0921-12 34 65	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BLAUS	Blauer See Delikatessen	Foisternstr. 57	001	0621-08460	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BLONP	Blondel père et fils	24, place Kleber Estrasburgo	008	88.60.15.31	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BOLID	Bolido Comidas preparadas	C/ Araquistán, 67	009	(91) 555 91 99	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BONAP	Bon app	12, rue des Bouchers	001	91.24.45.41	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	003	(604) 555-3745	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BSBEV	B's Beverages	Fountainey Circus	009	(71) 555-1212	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
CACTU	Cactus Comidas para llevar	Cerito 333	002	(1) 135-4892	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
CENTC	Centro comercial Modézuma	Sierras de Granada 9993	008	(5) 555-7293	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

## Laboratorio 3.2

### Aplicación ASP.NET MVC con acceso a datos

En el mismo proyecto ASP.NET MVC realice una consulta a la tabla tb\_clientes filtrando por las iniciales de su nombre (campo nombreCia). En este proceso, codifique la vista utilizando lenguaje Razor.

#### SOLUCION

Defina en el controlador el ActionResult ClientesxNombre, definiendo el parámetro de entrada nom, el cual recibe el valor para realizar el filtro. Aplicamos expresiones Lambda para filtrar los registros.

```

NegociosController.cs > X
Laboratorio03 Controllers NegociosController.cs
using ...
namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        // instancia del DBContext
        Negocios2017Entities bd = new Negocios2017Entities();

        public ActionResult Index()...
        public ActionResult ClientesxNombre(string nom = null)
        {
            ViewBag.nom = nom;

            return View(bd.tb_clientes.Where(x => x.NombreCia.StartsWith(nom,
                StringComparison.CurrentCultureIgnoreCase)).ToList());
        }
    }
}

```

ActionResult que filtra los registros por el campo nombreCia

A continuación agregamos una Vista al ActionResult, tal como se muestra

```

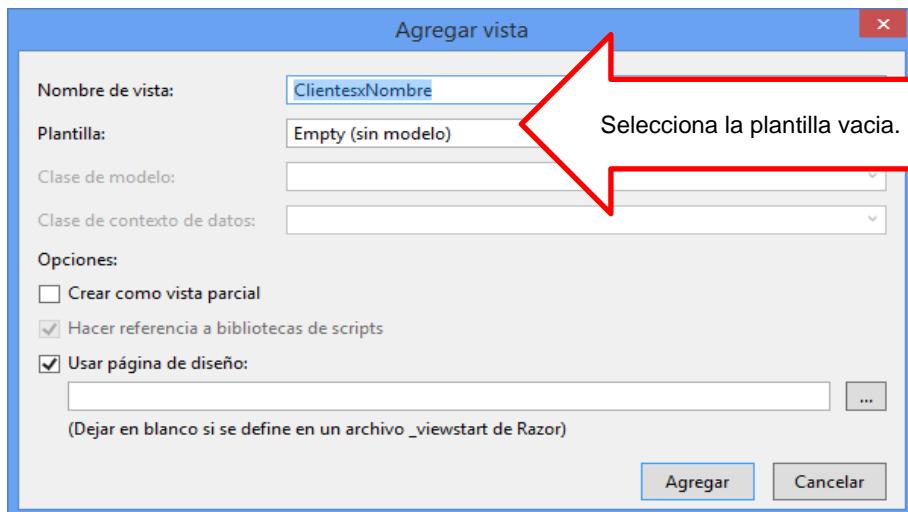
NegociosController.cs > X
Laboratorio03 Controllers NegociosController.cs
using ...
namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        // instancia del DBContext
        Negocios2017Entities bd = new Negocios2017Entities();

        public ActionResult Index()...
        public ActionResult ClientesxNombre(string nom = null)
        {
            return View(bd.tb_clientes.Where(x => x.NombreCia.StartsWith(nom,
                StringComparison.CurrentCultureIgnoreCase)).ToList());
        }
    }
}

```

Agrega una Vista al ActionResult

Para codificar la Vista, no debemos seleccionar la plantilla (Empty), presiona el botón AGREGAR



En la Vista, primero definimos el modelo: IEnumerable de tb\_clientes.  
A continuación defina dentro del formulario, de método Post, un control Text llamado nom, el cual coincide con el parámetro del ActionResult y un control tipo submit para enviar el formulario

```

@model IEnumerable<Laboratorio03.Models.tb_clientes>
@{
    ViewBag.Title = "Consulta de Clientes";
}



## Clientes por Nombre


using (Html.BeginForm("ClientesxNombre", "Negocios", FormMethod.Post))
{
    <span>Nombre del Cliente</span>
    <input name="nom" id="nom" value="@ViewBag.nom" />
    <input type="submit" value="Consulta" />
}

```

Debajo del formulario, defina un table el cual listará los registros de la consulta.

```

ClientesxNombre.cshtml*  NegociosController.cs
@model IEnumerable<Laboratorio03.Models.tb_clientes>
@{ViewBag.Title = "Consulta de Clientes";}



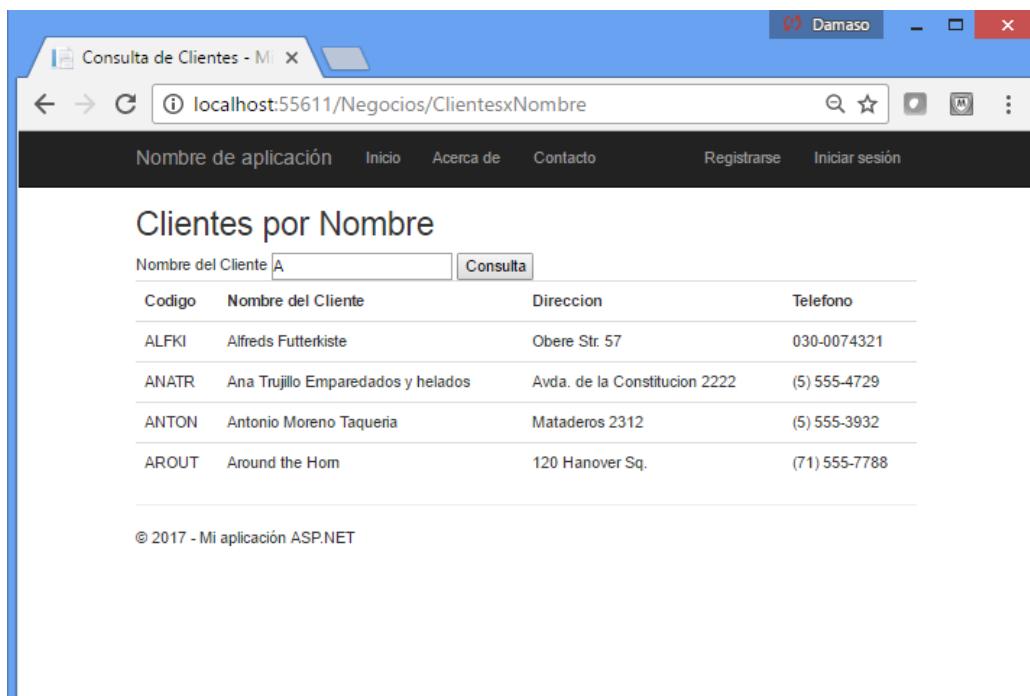
## Clientes por Nombre


@using (Html.BeginForm("ClientesxNombre", "Negocios", FormMethod.Post))
{
    <span>Nombre del Cliente</span>
    <input name="nom" id="nom" value="@ViewBag.nom" />
    <input type="submit" value="Consulta" />
}


| Codigo          | Nombre del Cliente | Direccion       | Telefono       |
|-----------------|--------------------|-----------------|----------------|
| @item.IdCliente | @item.NombreCia    | @item.Direccion | @item.Telefono |


```

Ejecuta la Vista: Ctrl + F5, ingresa la iniciales del nombre, al presionar el botón Consulta, si visualiza los registros coincidentes



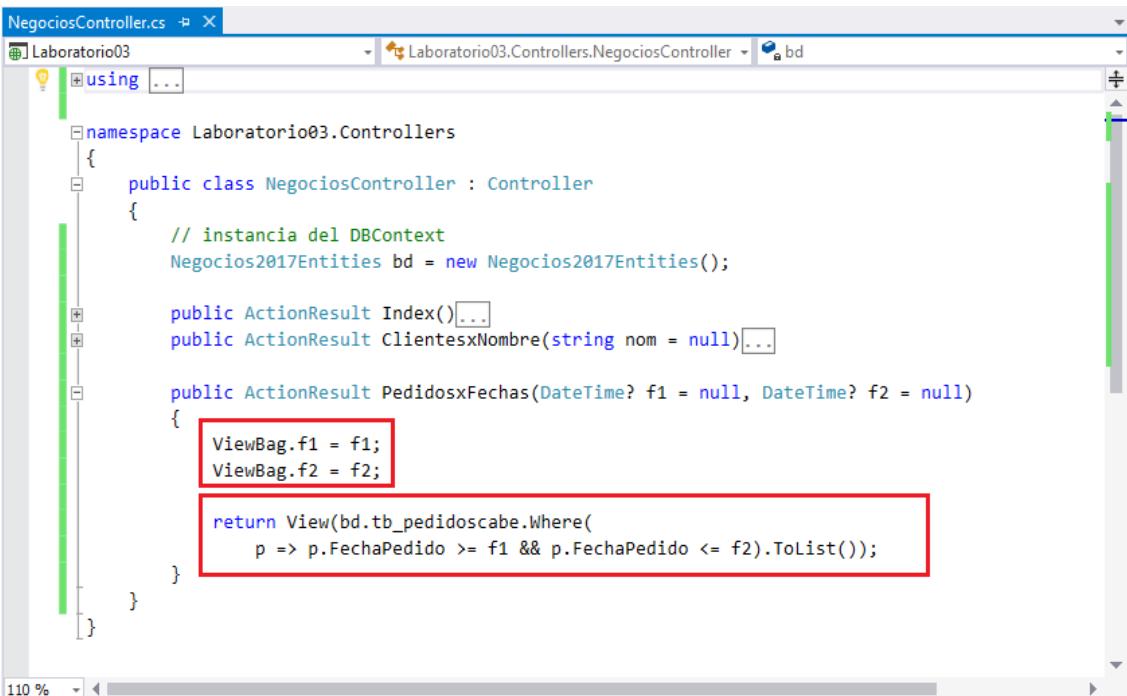
## Laboratorio 3.3

### Aplicación ASP.NET MVC con acceso a datos

En el mismo proyecto ASP.NET MVC realice una consulta a la tabla tb\_pedidoscabe filtrando por un rango de dos fechas del campo FechaPedido. En este proceso, codifique la vista utilizando lenguaje Razor.

#### SOLUCION

Defina en el controlador el ActionResult PedidosxFechas, definiendo el parámetro de entrada f1 y f2 de tipo DateTime, el cual recibe el valor para realizar el filtro. Aplicamos expresiones Lambda para filtrar los registros



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Laboratorio03.Models;

namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        // instancia del DBContext
        Negocios2017Entities bd = new Negocios2017Entities();

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

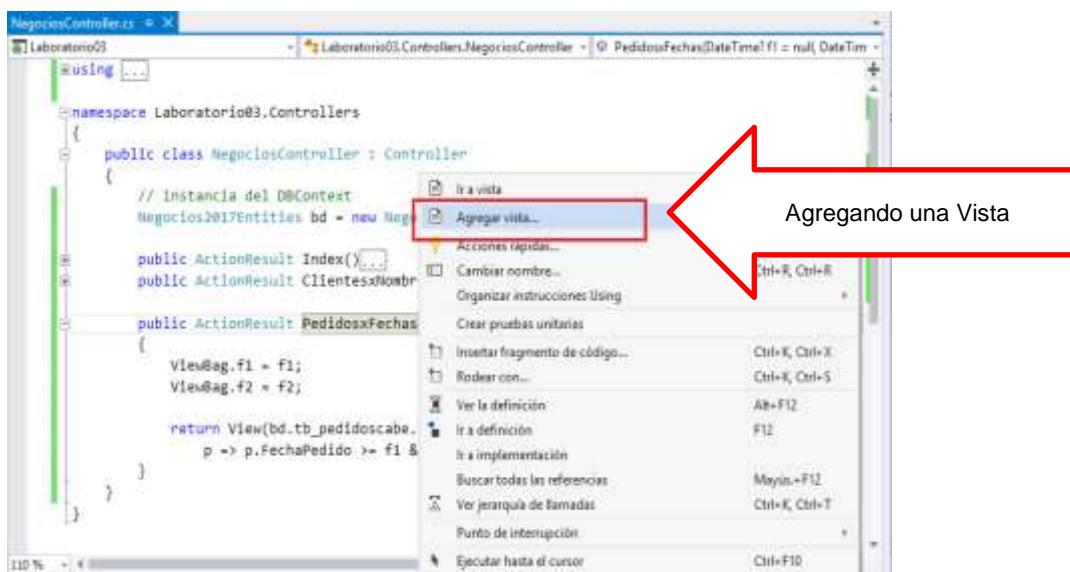
        public ActionResult ClientesNombre(string nom = null)
        {
            ViewBag.nom = nom;
            return View();
        }

        public ActionResult PedidosxFechas(DateTime? f1 = null, DateTime? f2 = null)
        {
            ViewBag.f1 = f1;
            ViewBag.f2 = f2;

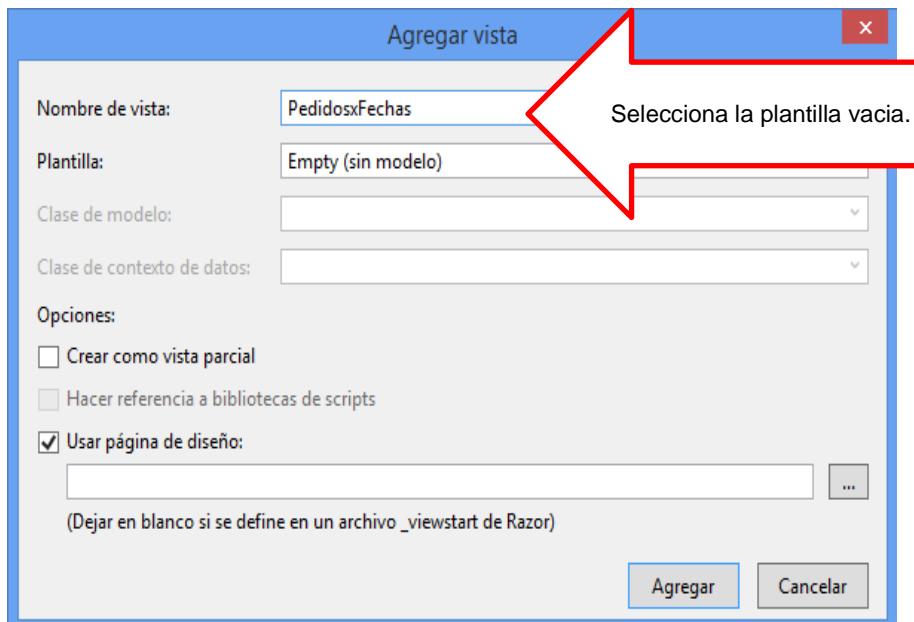
            return View(bd.tb_pedidoscabe.Where(
                p => p.FechaPedido >= f1 && p.FechaPedido <= f2).ToList());
        }
    }
}

```

A continuación agregamos la Vista al método.



Para codificar la Vista, no debemos seleccionar la plantilla (Empty), presiona el botón AGREGAR



En la Vista, primero definimos el modelo: IEnumerable de tb\_pedidoscabe. A continuación defina dentro del formulario, de método Post, dos input Text llamado f1 y f2, cuyos nombres coinciden con los parámetros del ActionResult y un control tipo submit para enviar el formulario

```

@model IEnumerable<Laboratorio03.Models.tb_pedidoscabe>
@{
    ViewBag.Title = "PedidosxFechas";
}



## Consulta de Pedidos entre Fechas



@using (Html.BeginForm("PedidosxFechas", "Negocios", FormMethod.Post))
{
    <span>Ingrese Fecha 1</span>
    <input name="f1" id="f1" value="@ViewBag.f1" />
    <input type="submit" value="Consulta" /><br />

    <span>Ingrese Fecha 2</span>
    <input name="f2" id="f2" value="@ViewBag.f2" />
}

```

Debajo del Formulario, defina la lista donde su visualiza los registros de la consulta

```

PedidosxFechas.cshtml  NegociosController.cs
@model IEnumerable<Laboratorio03.Models.tb_pedidoscabe>
@{ ViewBag.Title = "PedidosxFechas"; }



## Consulta de Pedidos entre Fechas



@using (Html.BeginForm("PedidosxFechas", "Negocios", FormMethod.Post))
{
    <span>Ingrese Fecha 1</span>
    <input name="f1" id="f1" value="@ViewBag.f1" />
    <input type="submit" value="Consulta" /><br />
    <span>Ingrese Fecha 2</span>
    <input name="f2" id="f2" value="@ViewBag.f2" />
}



| Numero de pedido | Fecha Pedido | Cliente | Direccion del Destinatario |
|------------------|--------------|---------|----------------------------|
|------------------|--------------|---------|----------------------------|


```

Ejecuta la Vista: Ctrl + F5, ingresa las fechas, al presionar el botón Consulta, si visualiza los registros coincidentes

Número de pedido	Fecha Pedido	Cliente	Dirección del Destinatario
11098	26/11/2016	Bon app	2817 Milton Dr.
11099	15/11/2016	Perciles Comidas clásicas	Calle Dr. Jorge Cash 321
11100	16/11/2016	Simons bistro	Vinbaelte 34
11101	01/12/2016	Richter Supermarkt	Starenweg 5
11102	02/12/2016	Bon app	12, rue des Bouchers
11103	19/12/2016	Rattlesnake Canyon Grocery	2817 Milton Dr.
11104	20/12/2016	Bon app	2817 Milton Dr.

© 2017 - Mi aplicación ASPNET

## Laboratorio 3.4

### Aplicación ASP.NET MVC con acceso a datos

En el mismo proyecto ASP.NET MVC realice una consulta a la tabla tb\_pedidoscabe filtrando por un determinado cliente, seleccionado desde un control DropDownList. En este proceso, codifique la vista utilizando lenguaje Razor.

#### SOLUCION

Defina en el controlador el ActionResult PedidosxCiente, definiendo el parámetro de entrada id de tipo string, el cual recibe el valor para realizar el filtro. Aplicamos expresiones Lambda para filtrar los registros.

En este proceso envío la lista de los clientes a través del ViewBag.cliente

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Negocios2017Entities;
using System.Data.Entity;

namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        // instancia del DBContext
        Negocios2017Entities bd = new Negocios2017Entities();

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

        public ActionResult ClientesxNombre(string nom = null)
        {
            return View();
        }

        public ActionResult PedidosxFechas(DateTime? f1 = null, DateTime? f2 = null)
        {
            return View();
        }

        public ActionResult PedidosxCiente(string id = null)
        {
            ViewBag.cliente = new SelectList(
                bd.tb_clientes.ToList(), "idcliente", "nombreCia", id);

            return View(bd.tb_pedidoscabe.Where(p => p.IdCliente == id).ToList());
        }
    }
}

```

A continuación agregamos la Vista al método

```

public class NegociosController : Controller
{
    // instancia del DBContext
    Negocios2017Entities bd = new Negocios2017Entities();

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

    public ActionResult ClientesxNombre()
    {
        return View();
    }

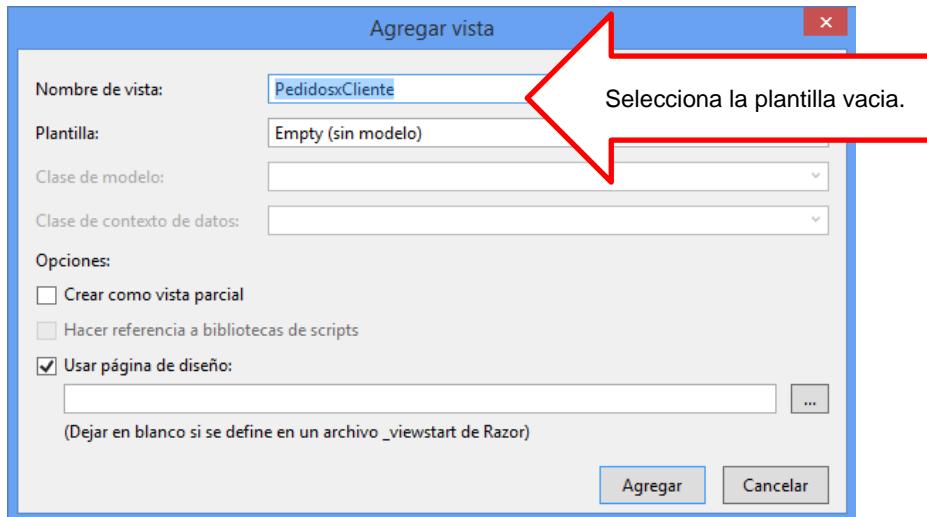
    public ActionResult PedidosxFechas(DateTime? f1 = null, DateTime? f2 = null)
    {
        return View();
    }

    public ActionResult PedidosxCiente(string id = null)
    {
        ViewBag.cliente = new SelectList(
            bd.tb_clientes.ToList(), "idcliente", "nombreCia", id);

        return View(bd.tb_pedidoscabe.Where(p => p.IdCliente == id).ToList());
    }
}

```

Para codificar la Vista, no debemos seleccionar la plantilla (Empty), presiona el botón AGREGAR



En la Vista, primero definimos el modelo: IEnumerable de tb\_pedidoscabe. A continuación defina dentro del formulario, de método Post, un DropDownList donde visualizamos los registros de tb\_clientes y cuyo nombre coincide con el parámetro del ActionResult y un control tipo submit para enviar el formulario

```

PedidosxCiente.cshtml ✘ X NegociosController.cs
@model IEnumerable<Laboratorio03.Models.tb_pedidoscabe>
{
    ViewBag.Title = "PedidosxCiente";
}

<h2>Consulta de Pedidos por Cliente</h2>

@using (Html.BeginForm("PedidosxCiente", "Negocios", FormMethod.Post))
{
    <span>Selecciona un Cliente</span>
    @Html.DropDownList("id", (SelectList)ViewBag.cliente)
    <input type="submit" value="Consulta" />
}

```

Defina el modelo de datos a recibir

Definición del formulario y sus controles

Debajo del Formulario, defina la lista donde su visualiza los registros de la consulta



```

PedidosCliente.cshtml" > X | NegociosController.cs
@model IEnumerable<Laboratorio03.Models.tb_pedidoscabe>
@{ ViewBag.Title = "PedidosxCiente"; }

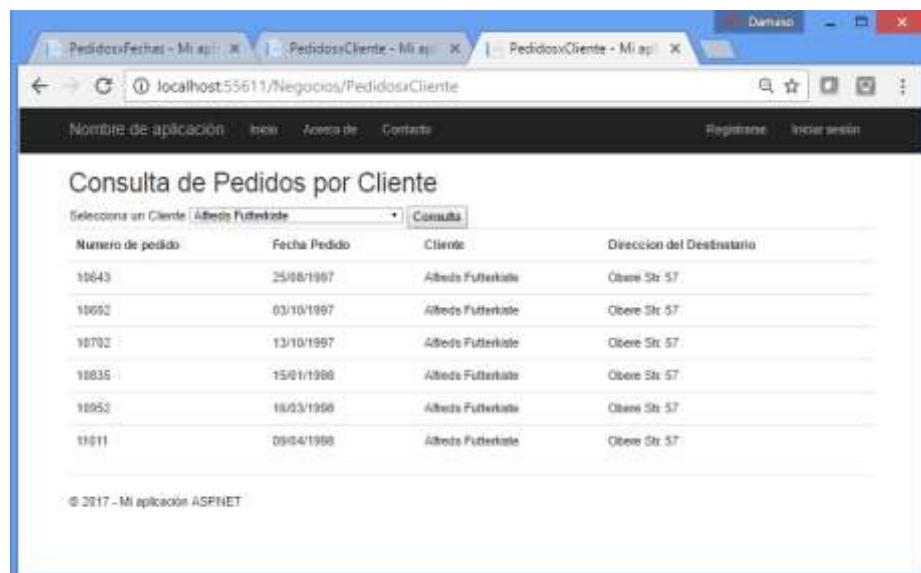
<h2>Consulta de Pedidos por Cliente</h2>

@using (Html.BeginForm("PedidosxCiente", "Negocios", FormMethod.Post))
{
    <span>Selecciona un Cliente</span>
    @Html.DropDownList("id", (SelectList)ViewBag.cliente)
    <input type="submit" value="Consulta" />
}

<table class="table">
    <tr>
        <th>Número de pedido</th>
        <th>Fecha Pedido</th>
        <th>Cliente</th>
        <th>Dirección del Destinatario</th>
    </tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>@item.IdPedido</td>
            <td>@item.FechaPedido.ToString("dd/MM/yyyy")</td>
            <td>@item.tb_clientes.NombreCia</td>
            <td>@item.DireccionDestinatario</td>
        </tr>
    }
</table>

```

Ejecuta la Vista: Ctrl + F5, selecciona el cliente, al presionar el botón Consulta, si visualiza los registros coincidentes



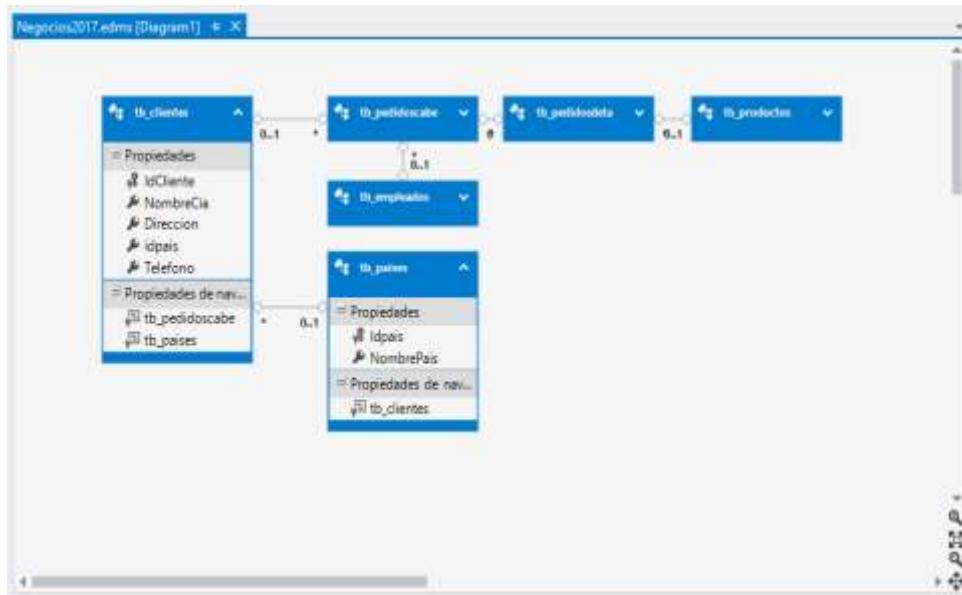
## Laboratorio 3.5

### Aplicación ASP.NET MVC con acceso a datos

En el mismo proyecto ASP.NET MVC realice los procesos CRUD a la tabla tb\_clientes almacenada en la base de datos Negocios2017.

#### SOLUCION

Para implementar el proceso, debemos actualizar el DBContext, agregando la tabla tb\_paises, tal como se muestra



En el Controlador, defina un ActionResult llamado Clientes(), el cual visualiza la lista de los registros de tb\_clientes

```

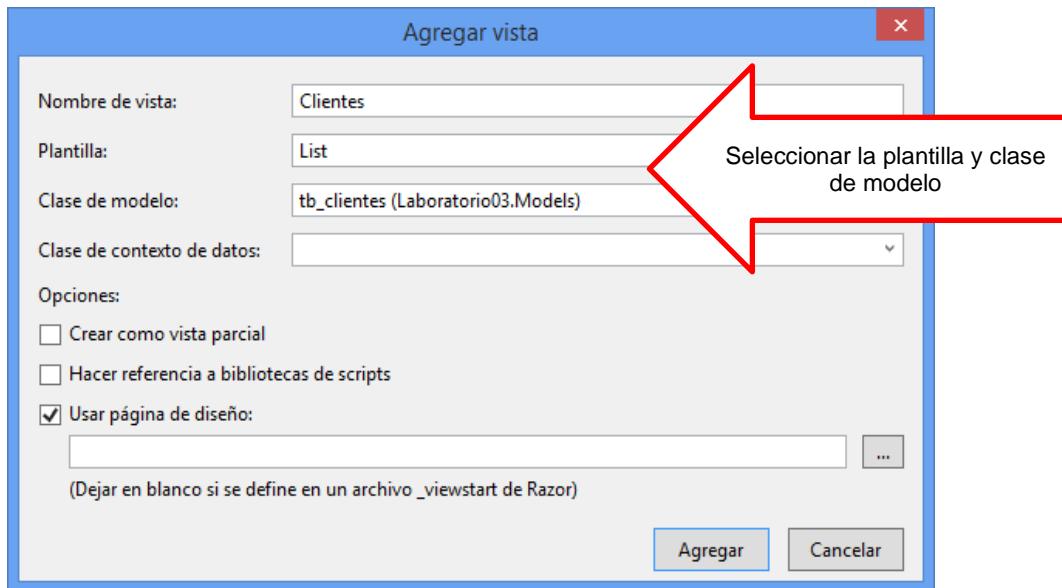
using ...
namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        Negocios2017Entities bd = new Negocios2017Entities();

        public ActionResult Index()...
        public ActionResult ClientesNombre(string nom = null)...
        public ActionResult PedidosxFechas(DateTime? f1 = null, DateTime? f2 = null)...
        public ActionResult PedidosxCliente(string id = null)...

        public ActionResult Clientes()
        {
            return View(bd.tb_clientes.ToList());
        }
    }
}
    
```

Definir el ActionResult Clientes()

En la ventana, selecciona la plantilla List (listado) y la clase de Modelo tb\_clientes, tal como se muestra



Generada la Vista, se imprime el siguiente código. Modifica los ActionLink para dirigir a los métodos del proceso CRUD. En los casos de los métodos EDIT, DETAILS y DELETE, defina el parámetro para realizar el proceso de búsqueda del cliente

```

Cuentas.cshtml ✘ X NegociosController.cs
@model IEnumerable<Laboratorio03.Models.tb_clientes>
@{
    ViewBag.Title = "Cuentas";
}



## Cuentas



| @Html.DisplayNameFor(model => model.IdCliente) | @Html.DisplayNameFor(model => model.NombreCia) | @Html.DisplayNameFor(model => model.Direccion) | @Html.DisplayNameFor(model => model.idpais) | @Html.DisplayNameFor(model => model.Telefono) |
|------------------------------------------------|------------------------------------------------|------------------------------------------------|---------------------------------------------|-----------------------------------------------|
|                                                |                                                |                                                |                                             |                                               |


```

ActionLink para Nuevo Cliente: `@Html.ActionLink("Create New", "NuevoCuenta")`

ActionLink para actualizar un Cliente: `@Html.ActionLink("Edit", "Edit", new { id=item.IdCliente }) | @Html.ActionLink("Details", "Details", new { id=item.IdCliente }) | @Html.ActionLink("Delete", "Delete", new { id=item.IdCliente })`

Ejecuta la Vista Ctrl + F5, donde se lista los registros de tb\_clientes

IDCliente	NombreCia	Dirección	IDPais	Telefono	
ALFKI	Alfreds Futterkiste	Obere Str. 57	001	(0) 691-4231	Edit   Detalle   Delete
ANATR	Alea Trade Empresarios y Relaciones	Avda. de la Constitución 2222	003	(51) 955-4729	Edit   Detalle   Delete
ANTON	Antonio Moreno Taquería	Mättaranta 3312	007	(51) 955-3802	Edit   Detalle   Delete
AROUT	Armand The Rose	120 Harenstein Sq.	004	(71) 555-7788	Edit   Detalle   Delete
BERGS	Berglunds snabbköp	Berguvsgatan 8	006	(91) 555-1248	Edit   Detalle   Delete
BLAUS	Blauer See Delikatessen	Fürstenstr. 17	001	(91) 555-0840	Edit   Detalle   Delete
BLOWF	Blowfish pescadería	24, place Kleber Düsseldorf	008	80-68-15-31	Edit   Detalle   Delete
BOLID	Bolido Corridas preparadas	C/ Acapulco 87	009	(91) 555-85-98	Edit   Detalle   Delete
BONAP	Bon app	12, rue des Blanchers	001	81-24-45-41	Edit   Detalle   Delete
BOTTM	Balkan-Döner Meats	23 Thausenstr. Bvl.	003	(04) 555-3743	Edit   Detalle   Delete
BSBEV	B's Beverages	Fauntleroy Circuit	005	(71) 555-1212	Edit   Detalle   Delete
CACTU	Cactus Comidas para llevar	Cerdo 335	001	(11) 125-4800	Edit   Detalle   Delete
CENTC	Centro comercial Multicenter	Sintra de Oriente 9999	008	(51) 955-7281	Edit   Detalle   Delete

## Proceso Nuevo Cliente

Defina en el Controlador el ActionResult NewCliente() el cual envía a la pagina la estructura de la tabla tb\_clientes y almacenamos en un ViewBag la lista de los países.

```

namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        Negocios2017Entities bd = new Negocios2017Entities();

        public ActionResult Index()...
        public ActionResult ClientesxNombre(string nom = null)...
        public ActionResult PedidosxFechas(DateTime? f1 = null, DateTime? f2 = null)...
        public ActionResult PedidosxCliente(string id = null)...

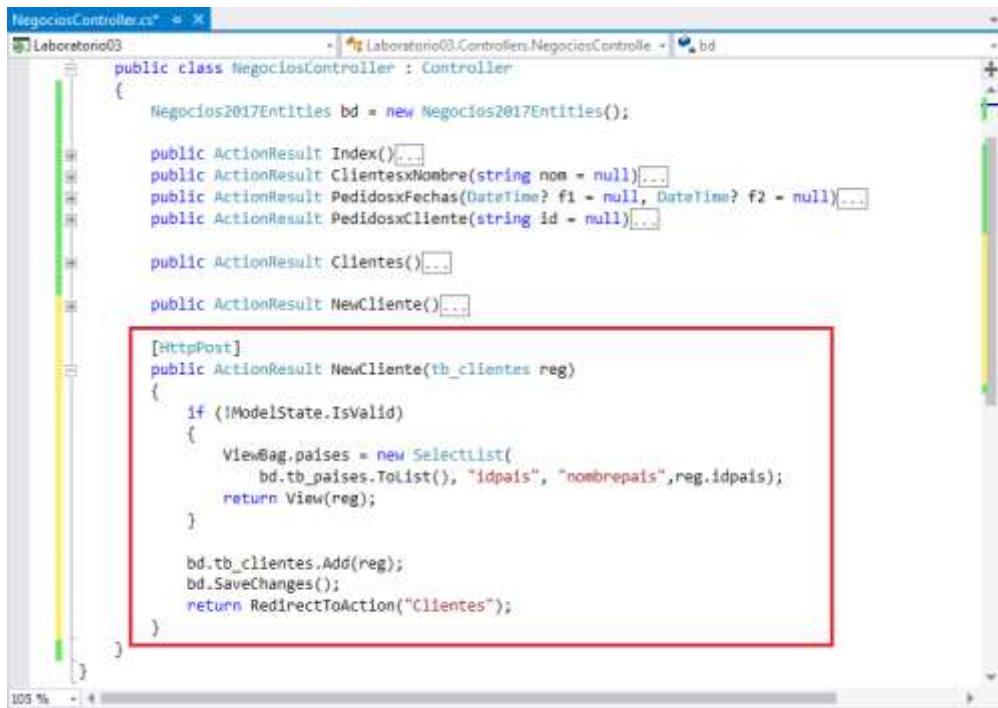
        public ActionResult Clientes()...

        public ActionResult NewCliente()
        {
            ViewBag.paises = new SelectList(bd.tb_paises.ToList(), "idpais", "nombrepais");

            return View(new tb_clientes());
        }
    }
}

```

Implemente el ActionResult NewCliente, de método POST, el cual recibe el registro de tb\_clientes para agregarlo al modelo. En este proceso se ejecuta la validación de los datos.



```

public class NegociosController : Controller
{
    Negocios2017Entities bd = new Negocios2017Entities();

    public ActionResult Index()...
    public ActionResult ClientesxNombre(string nom = null)...
    public ActionResult PedidosxFechas(DateTime? f1 = null, DateTime? f2 = null)...
    public ActionResult PedidosxCiente(string id = null)...

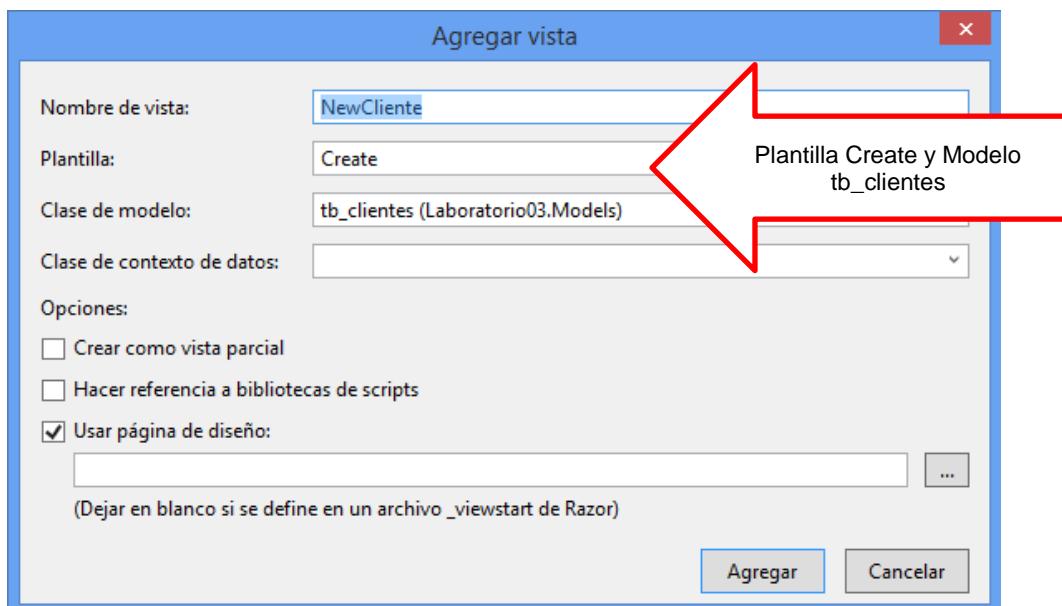
    public ActionResult Clientes()...
    public ActionResult NewCliente()...

    [HttpPost]
    public ActionResult NewCliente(tb_clientes reg)
    {
        if (!ModelState.IsValid)
        {
            ViewBag.paises = new SelectList(
                bd.tb_paises.ToList(), "idpais", "nombrepais", reg.idpais);
            return View(reg);
        }

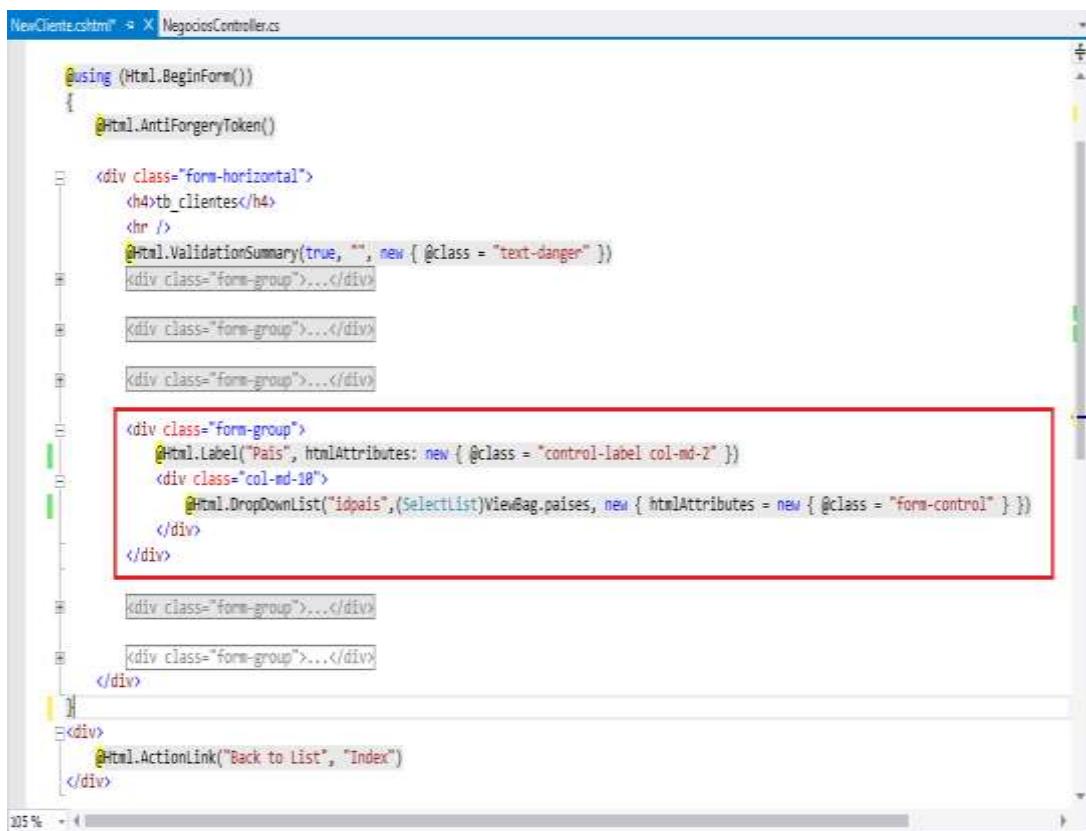
        bd.tb_clientes.Add(reg);
        bd.SaveChanges();
        return RedirectToAction("Clientes");
    }
}

```

Agregar una Vista para el ActionResult, selecciona en la plantilla de tipo Create y el modelo tb\_clientes, tal como se muestra



Al generar la Vista, modificar el contenido de idpais, reemplazando el control por un DropDownList, para seleccionar el país, tal como se muestra.



```
using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>tb_clientes</h4>
        <br />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">...</div>

        <div class="form-group">...</div>

        <div class="form-group">...</div>

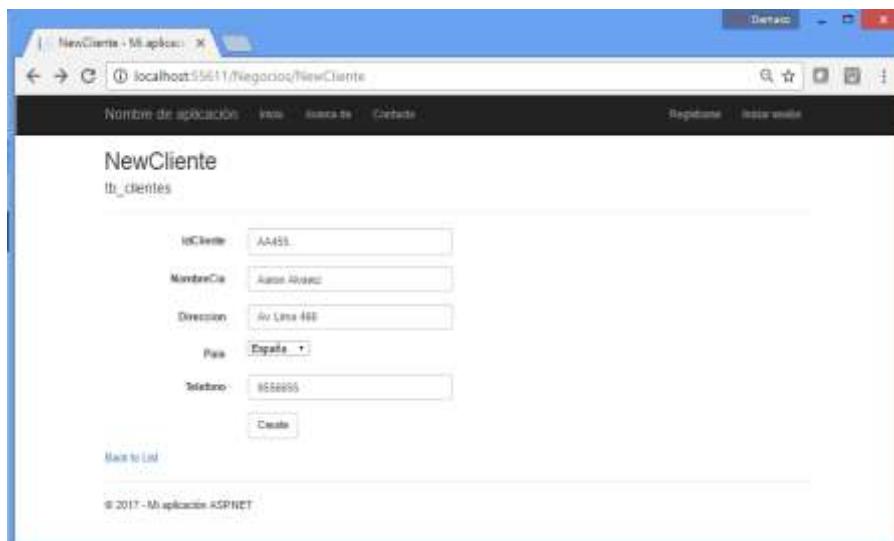
        <div class="form-group">
            @Html.Label("País", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownList("idpais", (SelectList)ViewBag.paises, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>

        <div class="form-group">...</div>

        <div class="form-group">...</div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

Ejecuta la página Ctrl+F5, para agregar un registro de tb\_clientes



## Proceso Details Cliente

Defina en el Controlador el ActionResult Details(id) el cual envía a la pagina los datos del cliente seleccionado por su idCliente.

```

namespace Laboratorio03.Controllers
{
    public class NegociosController : Controller
    {
        Negocios2017Entities bd = new Negocios2017Entities();

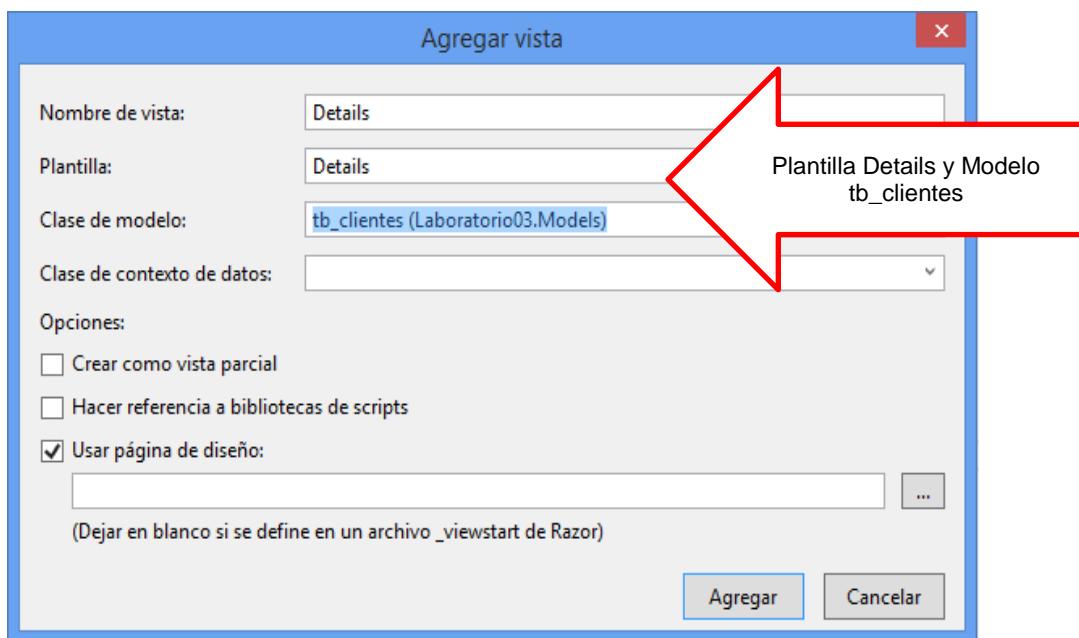
        public ActionResult Index()...
        public ActionResult ClientesxNombre(string nom = null)...
        public ActionResult PedidosxFechas(DateTime? f1 = null, DateTime? f2 = null)...
        public ActionResult PedidosxCliente(string id = null)...

        public ActionResult Clientes()...
        public ActionResult NewCliente()...
        [HttpPost]
        public ActionResult NewCliente(tb_clientes reg)...

        public ActionResult Details(string id)
        {
            return View(bd.tb_clientes.Where(c => c.IdCliente == id).FirstOrDefault());
        }
    }
}

```

Agregar una Vista para el ActionResult, selecciona en la plantilla de tipo Details y el modelo tb\_clientes, tal como se muestra



En el código de la Vista, modifique la estructura para mejorar la presentación de la página

```

Details.cshtml
@model NegociosController.cs
@model Laboratorio03.Models.tb_clientes

@{ViewBag.Title = "Details";}



## Details



<h4>tb_clientes</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>@Html.DisplayNameFor(model => model.IdCliente)</dt>
        <dd>@Html.DisplayFor(model => model.IdCliente)</dd>

        <dt>@Html.DisplayNameFor(model => model.NombreCia)</dt>
        <dd>@Html.DisplayFor(model => model.NombreCia)</dd>

        <dt>@Html.DisplayNameFor(model => model.Direccion)</dt>
        <dd>@Html.DisplayFor(model => model.Direccion)</dd>

        <dt>@Html.DisplayNameFor(model => model.tb_paises.NombrePais )</dt>
        <dd>@Html.DisplayFor(model => model.idpais)</dd>

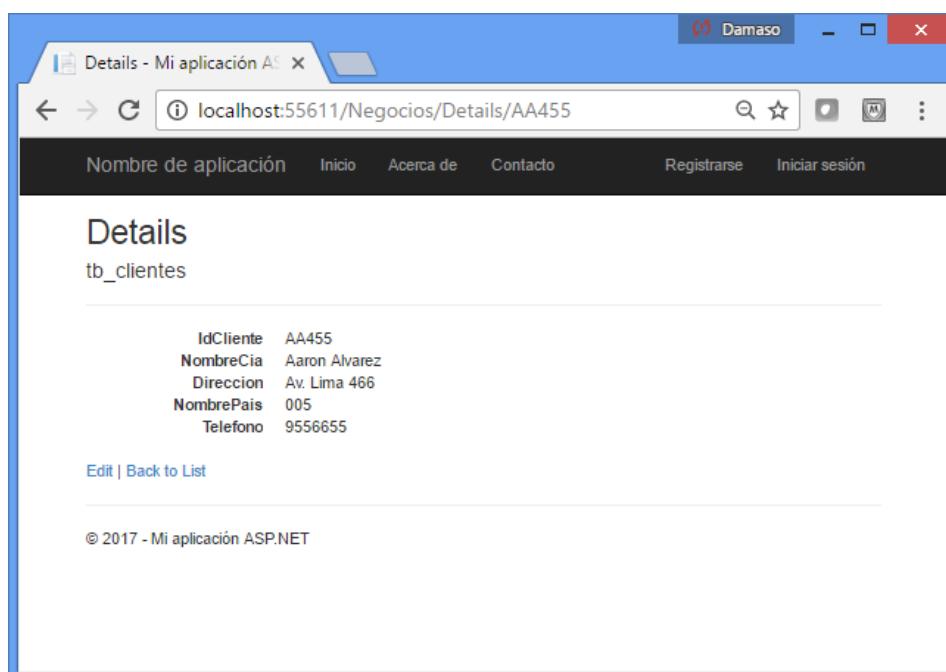
        <dt>@Html.DisplayNameFor(model => model.Telefono)</dt>
        <dd>@Html.DisplayFor(model => model.Telefono)</dd>
    </dl>



@Html.ActionLink("Edit", "Edit", new { id=Model.IdCliente }) |
        @Html.ActionLink("Back to List", "Clientes")

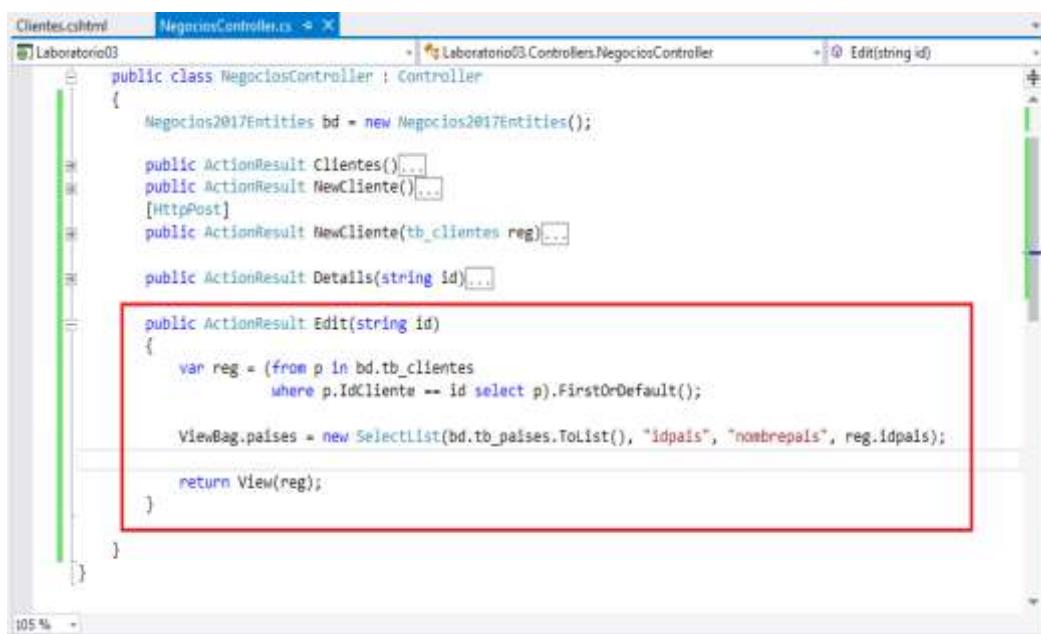

```

Ejecuta la página Clientes(), al seleccionar un cliente, visualizamos sus datos, tal como se muestra.



## Proceso Edit

Defina en el Controlador el ActionResult Edit(id) el cual envía un cliente filtrado por su campo idCliente y almacenamos en un ViewBag la lista de los países.



```

Cínteres.cshtml    NegociosController.cs
Laboratorio03      Laboratorio03.Controllers.NegociosController      Edit(string id)
public class NegociosController : Controller
{
    Negocios2017Entities bd = new Negocios2017Entities();

    public ActionResult Clientes()...
    public ActionResult NewCliente()...
    [HttpPost]
    public ActionResult NewCliente(tb_clientes reg)...

    public ActionResult Details(string id)...

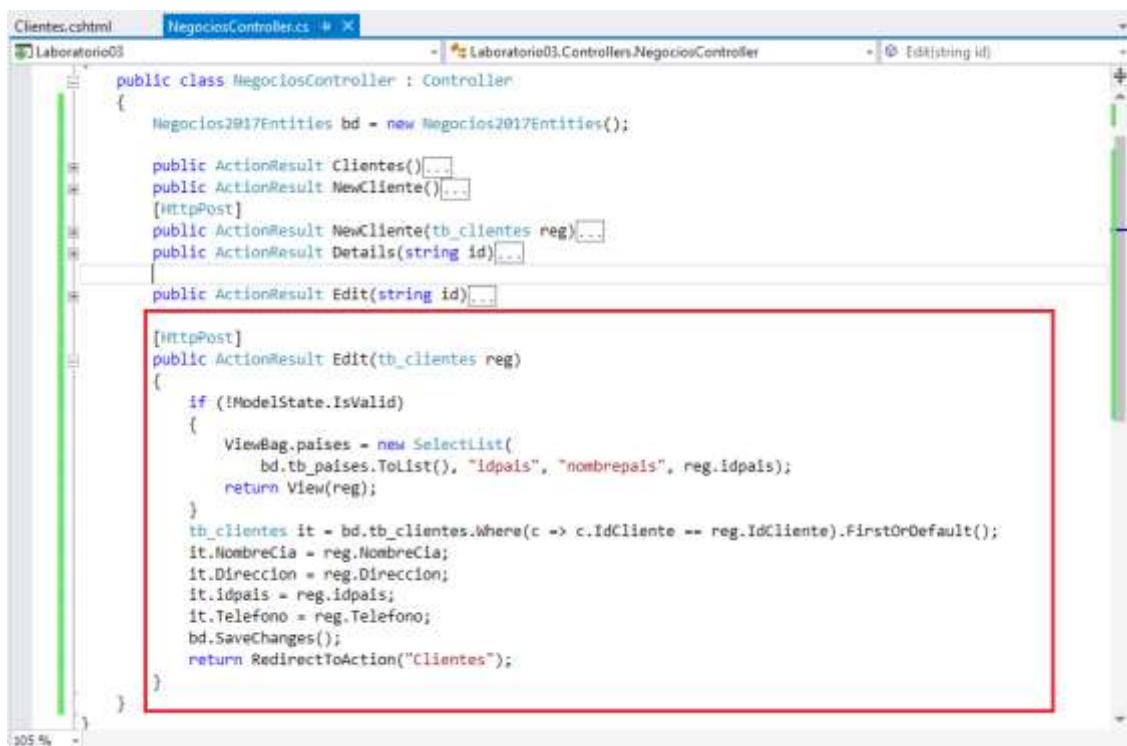
    public ActionResult Edit(string id)
    {
        var reg = (from p in bd.tb_clientes
                   where p.IdCliente == id select p).FirstOrDefault();

        ViewBag.paises = new SelectList(bd.tb_paises.ToList(), "idpais", "nombrepais", reg.idpais);

        return View(reg);
    }
}

```

Defina su método Post, donde recibe los datos del cliente para ser actualizados al modelo, tal como se muestra



```

Cínteres.cshtml    NegociosController.cs
Laboratorio03      Laboratorio03.Controllers.NegociosController      Edit(string id)
public class NegociosController : Controller
{
    Negocios2017Entities bd = new Negocios2017Entities();

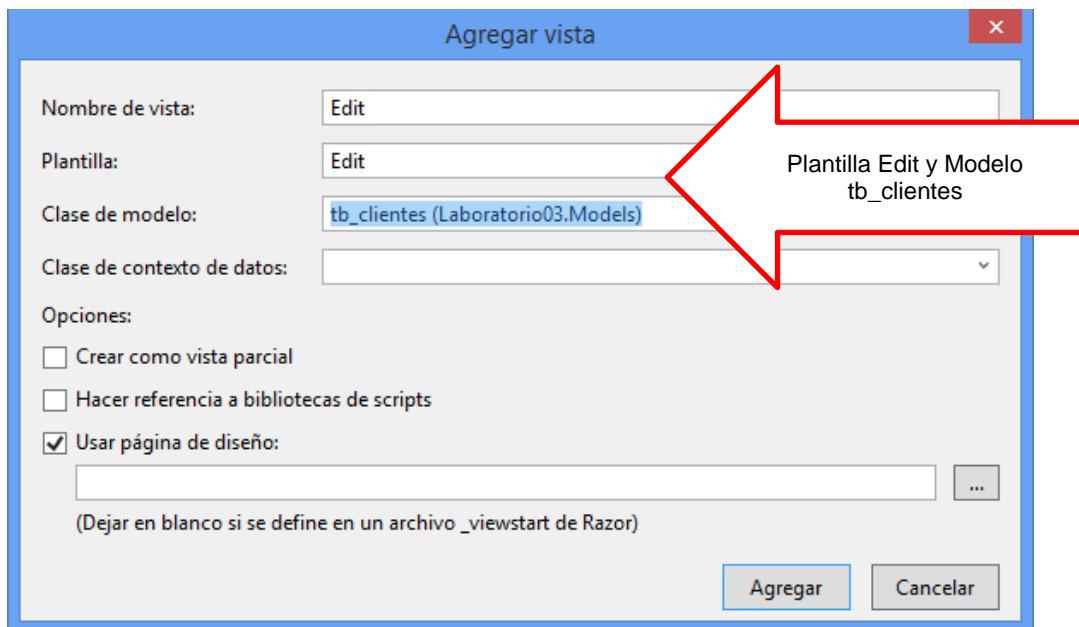
    public ActionResult Clientes()...
    public ActionResult NewCliente()...
    [HttpPost]
    public ActionResult NewCliente(tb_clientes reg)...
    public ActionResult Details(string id)...

    public ActionResult Edit(string id)...

    [HttpPost]
    public ActionResult Edit(tb_clientes reg)
    {
        if (!ModelState.IsValid)
        {
            ViewBag.paises = new SelectList(
                bd.tb_paises.ToList(), "idpais", "nombrepais", reg.idpais);
            return View(reg);
        }
        tb_clientes it = bd.tb_clientes.Where(c => c.IdCliente == reg.IdCliente).FirstOrDefault();
        it.NombreCia = reg.NombreCia;
        it.Direccion = reg.Direccion;
        it.idpais = reg.idpais;
        it.Telefono = reg.Telefono;
        bd.SaveChanges();
        return RedirectToAction("Clientes");
    }
}

```

Agregar una Vista para el ActionResult, selecciona en la plantilla de tipo Edit y el modelo tb\_clientes, tal como se muestra



En el código de la Vista, modifique el contenido del idpais, reemplazandolo por un DropDownList, tal como se muestra

```
Editor.html  X Cuentas.cshtml  NegociosController.cs
@model Laboratorio03.Models.tb_clientes

@{ ViewBag.Title = "Edit"; }

<h2>Edit</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>tb_clientes</h4>
        <br />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">...</div>
        <div class="form-group">...</div>
        <div class="form-group">...</div>

        <div class="form-group">
            @Html.Label("Pais", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownList("idpais", (SelectList)ViewBag.paises, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>

        <div class="form-group">...</div>
        <div class="form-group">...</div>
    </div>
    <div>
        @Html.ActionLink("Back to List", "Cuentas")
    </div>
}
```

Ejecuta la pagina Clientes(), al seleccionar un cliente (Edit) direcccionamos a una segunda página, modifica sus datos. Al presionar el botón Save, los datos serán actualizados.

The screenshot shows a web browser window titled "Edit - Mi aplicación ASP.NET". The address bar displays "localhost:55611/Negocios/Edit/AA455". The page has a header with links for "Nombre de aplicación", "Inicio", "Acerca de", "Contacto", "Registrarse", and "Iniciar sesión". Below the header, the word "Edit" is displayed above a section titled "tb\_clientes". The form contains five input fields: "IdCliente" (value AA455), "NombreCia" (value Aaron Alvarez Garcia), "Direccion" (value Av. Lima 46633), "Pais" (value España), and "Telefono" (value 9556655). A "Save" button is located below the form. At the bottom of the page, there is a link "Back to List" and a copyright notice "© 2017 - Mi aplicación ASP.NET".

# Resumen

- Los objetos en el modelo de datos representan clases que interactúan con una base de datos. Normalmente, se puede pensar en el modelo de datos como el conjunto de las clases creadas por herramientas como Entity Framework (EF)
- La clase principal que es responsable de la interacción con datos como objetos es **System.Data.Entity.DbContext** (a menudo referido como contexto). La clase de contexto gestiona la entidad objetos en tiempo de ejecución, que incluye llenar objetos con datos de una base de datos, el control de cambios, y la persistencia de los datos a la base de datos.
- El método recomendado para trabajar con el contexto es definir una clase que deriva de DbContext y expone DbSet propiedades que representan las colecciones de las entidades especificadas en el contexto. Si está trabajando con el Diseñador del Entity Framework, el contexto se generará automáticamente. Si está trabajando con el Código Primero, normalmente escribe el contexto mismo.
- Los selectores permiten obtener el contenido del documento para ser manipularlo. Al utilizarlos, los selectores retornan un arreglo de objetos que coinciden con los criterios especificados. Este arreglo no es un conjunto de objetos del DOM, son objetos de jQuery con un gran número de funciones y propiedades predefinidas para realizar operaciones con los mismos.
- Entity Framework es un conjunto de herramientas incluidas en ADO.NET que dan soporte para el Desarrollo de Aplicaciones orientadas a datos. Arquitectos y desarrolladores de aplicaciones orientadas a datos se debaten con la necesidad de realizar dos diferentes objetivos.
- Las expresiones lambda son funciones anónimas que devuelven un valor. Se pueden utilizar en cualquier lugar en el que se pueda utilizar un delegado. Son de gran utilidad para escribir métodos breves que se pueden pasar como parámetro.
- Las expresiones lambda se declaran en el momento en el que se van a usar como parámetro para una función.
- Las expresiones lambda son utilizadas ampliamente en Entity Framework. Cuando realizamos una consulta del tipo

```
var consulta = post.Where(p => p.User == "admin");
```
- La cláusula Where tiene como parámetro una expresión lambda. La operación descrita para recuperar los posts del usuario admin se almacenará como una estructura de datos que podemos pasar entre capas como parámetro de métodos. Esta expresión la analizará y ejecutará el proveedor de datos que se está usando y la convertirá en código SQL para realizar la consulta contra la base de datos.
- Los objetos de un contexto del objeto son instancias de tipos de entidad que representan los datos en el origen de datos. Puede modificar, crear y eliminar objetos de un contexto y Entity Framework realizará el seguimiento de los cambios efectuados en estos objetos. Cuando se llama al método SaveChanges, Entity Framework genera y ejecuta comandos que llevan a cabo instrucciones equivalentes de inserción, actualización o eliminación con el origen de datos.
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
  - [http://dawconsblog.blogspot.com/2014/04/tutorial-crud-con-mvc-y-entity-framework\\_14.html](http://dawconsblog.blogspot.com/2014/04/tutorial-crud-con-mvc-y-entity-framework_14.html)
  - <https://translate.google.com.pe/translate?hl=es&sl=en&u=https://msdn.microsoft.com/en-us/data/jj729737.aspx&prev=search>
  - <http://speakingin.net/2007/11/18/aspnet-mvc-framework-primera-parte/>





# TRABAJANDO CON DATOS EN ASP.NET MVC

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla aplicaciones con acceso a datos teniendo implementando procesos de consulta y actualización de datos

## TEMARIO

### Tema 4: Manejo de Vistas (3 horas)

- 4.1 Lenguaje Razor
- 4.2 Vistas parciales
- 4.3 Patrón ViewModel

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando en patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



## 4. MANEJO DE VISTAS

El marco de ASP.NET MVC admite el uso de un motor de vista para generar las vistas (interfaz de usuario). De forma predeterminada, el marco de MVC usa tipos personalizados (ViewPage, ViewMasterPage y ViewUserControl) que heredan de los tipos de página ASP.NET existente (.aspx), página maestra (.master), y control de usuario (.ascx) existentes como vistas.

En el flujo de trabajo típico de una aplicación web de MVC, los métodos de acción de controlador administran una solicitud web de entrada. Estos métodos de acción usan los valores de parámetro de entrada para ejecutar el código de aplicación y recuperar o actualizar los objetos del modelo de datos de una base de datos.

En el modelo Model-View-Controller (MVC), las vistas están pensadas exclusivamente para encapsular la lógica de presentación. No deben contener lógica de aplicación ni código de recuperación de base de datos. El controlador debe administrar toda la lógica de aplicación. Una vista representa la interfaz de usuario adecuada usando los datos que recibe del controlador. Estos datos se pasan a una vista desde un método de acción de controlador usando el método View.

### 4.1 Lenguaje Razor

ASP.NET MVC ha tenido el concepto de motor de vistas (View Engine), las cuales realizan tareas sólo de presentación. No contienen ningún tipo de lógica de negocio y no acceden a datos. Básicamente se limitan a mostrar datos y a solicitar datos nuevos al usuario.

ASP.NET MVC permite separar la sintaxis del servidor, del framework de ASP.NET MVC, es lo que llamamos un motor de vistas de ASP.NET MVC, el cual viene acompañado de un nuevo motor de vistas, llamado Razor.

- Razor es una simple sintaxis de programación para código de servidor embebido en web pages.
- La sintaxis Razor es basada en el framework ASP.NET, parte de Microsoft.NET Framework.
- La sintaxis Razor proporciona todo el poder de ASP.NET, pero es usado como una sintaxis simplificada fácil de aprender.
- Las páginas web Razor puede ser descrito como páginas HTML con dos clases de contenido: contenido HTML y código Razor.
- Las páginas web ASP.NET con sintaxis Razor tiene un archivo de extensión cshtml (Razor para C#) o vbhtml (Razor para VB).

#### **Sintaxis de Razor**

En Razor el símbolo de la arroba (@) marca el inicio de código de servidor; no hay símbolo para indicar que se termina el código de servidor: el motor Razor deduce cuando termina en base al contexto.

Para mostrar una variable de servidor (item.Nombre por ejemplo) simplemente la precedemos de una @. El uso de la estructura repetitiva foreach, en el uso de la llave de cierre, no debe ser precedida de ninguna arroba, Razor ya sabe que esa llave es de servidor y cierra el foreach abierto.

El uso de la @ funciona de dos maneras básicas:

- @expresión: Renderiza la expresión en el navegador. Así @item.Nombre muestra el valor de ítem.Nombre.
- @{ código }: Permite ejecutar un código que no genera salida HTML.

## HTML Helper

La clase HtmlHelper proporciona métodos que ayudan a crear controles HTML mediante programación. Todos los métodos HtmlHelper generan HTML y devuelven el resultado como una cadena.

Los métodos de extensión para la clase HtmlHelper están en el namespace System.Web.Mvc.Html. Estas extensiones añaden métodos de ayuda para la creación de formas, haciendo controles HTML, renderizado vistas parciales, la validación de entrada, y más.

Puede utilizar la clase HtmlHelper en una vista utilizando la propiedad de la vista built-in Html. Por ejemplo, llamar @ Html.ValidationSummary () hace un resumen de los mensajes de validación.

Cuando se utiliza un método de extensión para hacer que un elemento HTML incluya un valor, es importante entender cómo se recupera ese valor. Para los elementos que no sea un elemento de la contraseña (por ejemplo, un cuadro de texto o botón de radio), el valor se recupera en este orden: objeto ModelStateDictionary con el valor del parámetro en el método de extensión; el objeto ViewDataDictionary con un valor en los atributos del elemento HTML.

Para un elemento de la contraseña, el valor del elemento se recupera desde el valor de parámetro en el método de extensión, o la del valor de atributo en el html atributos si no se proporciona el parámetro.

Razor trabaja tanto con paginas aspx (Web Form) así como también con paginas cshtml (MVC)

```
<h1>Code Nugget Example with .ASPx file</h1>

<h3>
    Hello <%=name %>, the year is <%= DateTime.Now.Year %>
</h3>

<p>
    Checkout <a href="/Products/Details/<%=productId %>">this product</a>
</p>
```

```
<h1>Razor Example</h1>

<h3>
    Hello @name, the year is @DateTime.Now.Year
</h3>

<p>
    Checkout <a href="/Products/Details/@productId">this product</a>
</p>
```

Como se mencionó anteriormente, Razor no es un nuevo lenguaje de programación, el equipo de Microsoft buscó fusionar el conocimiento de la programación en VB.NET o C#, con el estándar HTML,

```
<ul id="products">

    @foreach(var p in products) {
        <li>@p.Name (@p.Price)</li>
    }

</ul>
```

Combinar Razor con los Html Helpers nos ayuda a crear de manera rápida y sencilla código e incluso acceder a datos que no necesariamente viene del controlador, tal como se muestra.

```
@{
    var items = new List<SelectListItem>{
        new SelectListItem {Value = "1", Text = "Blue"},
        new SelectListItem {Value = "2", Text = "Red"},
        new SelectListItem {Value = "3", Text = "Green", Selected = true},
        new SelectListItem {Value = "4", Text = "Yellow", Selected = true},
        new SelectListItem {Value = "5", Text = "Black"}
    };
}

@Html.ListBox("myListbox", items, null, 6, true)
```

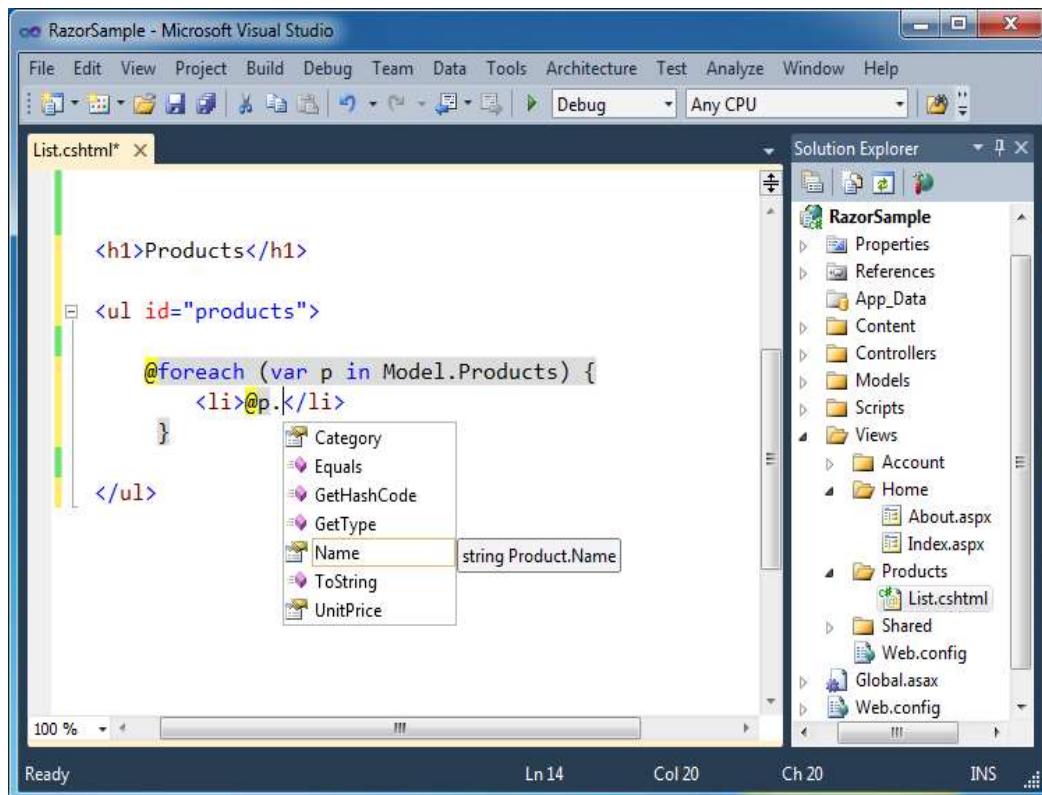
```
@{
    var db = Database.Open("Northwind");
    var data = db.Query("SELECT CategoryId, CategoryName FROM Categories");
    var items = data.Select(i => new SelectListItem {
        Value = i.CategoryId.ToString(),
        Text = i.CategoryName,
        Selected = i.CategoryId == 4 ? true : false
    });
}

@Html.DropDownList("CategoryId", items)
```

```
@{
    var db = Database.Open("Northwind");
    var data = db.Query("SELECT CategoryId, CategoryName FROM Categories");
    var items = data.Select(i => new SelectListItem {
        Value = i.CategoryId.ToString(),
        Text = i.CategoryName
    });
}

@Html.DropDownList("CategoryId", "Please Select One", items, 4, new {@class = "special"})
```

Finalmente, hay un amplio soporte de Intellisense en Visual studio.



### Renderizado de los helpers

Hay HTML Helpers para toda clase de controles de formulario Web: Check Boxes, hidden fields, password boxes, radio buttons, text boxes, text areas, *DropDownList* y *ListBox*.

Hay también un HTML Helper para el elemento Label, los cuales nos asocia a una etiqueta <label> del formulario Web. Cada HTML Helper nos da una forma rápida de crear HTML valido para el lado del cliente.

La siguiente tabla lista los Html Helpers y su correspondiente renderizado HTML:

Helper	HTML Element
Html.CheckBox	<input type="checkbox" />
Html.ActionLink	<a href="" />
Html.DropDownList	<select></select>
Html.Hidden	<input type="hidden" />
Html.Label	<label for="" />
Html.ListBox	<select></select> or <select multiple></select>
Html.Password	<input type="password" />
Html.Radio	<input type="radio" />
Html.TextArea	<textarea></textarea>
Html.TextBox	<input type="text" />

A continuación mostramos una comparativa de código creado con los HTML Helpers de Razor y su equivalencia en HTML

HTML Helper Razor	HTML renderizado
<pre>&lt;form method="post"&gt;     First Name:     @Html.TextBox("firstname")&lt;br /&gt;     Last Name:     @Html.TextBox("lastname",     Request["lastname"])&lt;br /&gt;     Town: @Html.TextBox("town",     Request["town"], new     Dictionary&lt;string, object&gt;() {         "class", "special" })&lt;br /&gt;     Country:     @Html.TextBox("country",     Request["country"], new { size =     50 })&lt;br /&gt;     &lt;input type="submit" /&gt; &lt;/form&gt;</pre>	<pre>&lt;form method="post"&gt;     First Name: &lt;input     id="firstname" name="firstname"     type="text" value="" /&gt;&lt;br /&gt;     Last Name: &lt;input     id="lastname" name="lastname"     type="text" value="" /&gt;&lt;br /&gt;     Town: &lt;input id="town"     name="town" class="special"     type="text" value="" /&gt;&lt;br /&gt;     Country: &lt;input id="country"     name="country" size="50"     type="text" value="" /&gt;&lt;br /&gt;     &lt;input type="submit" /&gt; &lt;/form&gt;</pre>

## 4.2 Vistas parciales

Una vista parcial permite definir una vista que se representará dentro de una vista primaria. Las vistas parciales se implementan como controles de usuario de ASP.NET (.ascx). Cuando se crea una instancia de una vista parcial, obtiene su propia copia del objeto ViewDataDictionary que está disponible para la vista primaria. Por lo tanto, la vista parcial tiene acceso a los datos de la vista primaria. Sin embargo, si la vista parcial actualiza los datos, esas actualizaciones solo afectan al objeto ViewData de la vista parcial. Los datos de la vista primaria no cambian.

### Creando una vista parcial

Para crear una vista parcial se nombra de la siguiente forma: \_NombreVistaParcial, donde el nombre se le debe anteponer el símbolo “\_”.

En la siguiente figura se crea una vista parcial



Una vez creada la vista parcial nos aparecerá vacía y pasamos a generar el código del control que necesitamos.

Para invocar a la vista parcial con los distintos datos:

```
@Html.Partial("_ListPlayerPartial")
@Html.Partial("_ListPlayerPartial", new ViewDataDictionary { valores})
```

Para invocar a una vista parcial, la podemos realizar desde un ActionResult. La diferencia a una acción normal es que se retorna PartialView en lugar de View, y allí estamos definiendo el nombre de la vista parcial (\_Details) y como segundo parámetro el modelo, entonces la definición de la vista parcial.

```
public ActionResult Index(int id)
{
    var detalle = ventasMes
        .Where(c => c.Id == id)
        .Select(c => c.DetalleMes)
        .FirstOrDefault();

    return PartialView("_Details",detalle);
}
```

### 4.3 Patrón ViewModel

Model-View-ViewModel (MVVM) es un patrón de diseño de aplicaciones para desacoplar código de interfaz de usuario y código que no sea de interfaz de usuario. Con MVVM, defines la interfaz de usuario de forma declarativa (por ejemplo, mediante XAML) y usas el marcado de enlace de datos para vincularla a otras capas que contengan datos y comandos de usuario. La infraestructura de enlace de datos proporciona un acoplamiento débil que mantiene sincronizados la interfaz de usuario y los datos vinculados, y que enruta todas las entradas de usuario a los comandos apropiados.

El patrón MVVM organiza el código de tal forma que es posible cambiar partes individuales sin que los cambios afecten a las demás partes. Esto presenta numerosas ventajas, como las siguientes:

- Permite un estilo de codificación exploratorio e iterativo.
- Simplifica las pruebas unitarias.
- Permite aprovechar mejor herramientas de diseño como Expression Blend.
- Admite la colaboración en equipo.

Por el contrario, una aplicación con una estructura más convencional utiliza el enlace de datos únicamente en controles de lista y texto, y responde a la entrada del usuario mediante el control de eventos expuestos por los controles. Los controladores de eventos están estrechamente acoplados a los controles y suelen contener código que manipula la interfaz de usuario directamente. Esto hace que sea difícil o imposible reemplazar un control sin tener que actualizar el código de control de eventos.

#### Ventajas de las capas

Una de las ventajas de esta separación por capas es que permite facilitar la comprensión del código. Esto se debe a que el código de características específicas a menudo es independiente de otro código, lo que facilita su aprendizaje y permite reutilizarlo en otras aplicaciones. Por ejemplo, con la aplicación de muestra de Reversi, si te interesan solo

los conceptos de la interfaz de usuario y no la inteligencia artificial del juego, puedes omitir la capa de modelo. Si quieres ver cómo funciona una animación determinada, puedes buscarla en un único archivo XAML, independiente de todo el código C#.

Otra ventaja importante de la separación de la interfaz de usuario es que facilita la realización de pruebas unitarias automatizadas del código que no corresponde a la interfaz de usuario. Microsoft Visual Studio admite proyectos de pruebas unitarias, que permiten comprobar el diseño del código durante el desarrollo, así como identificar y diagnosticar errores.

En general, una arquitectura estrechamente acoplada dificulta los cambios y el diagnóstico de errores. La principal ventaja de una arquitectura desacoplada es que permite aislar el impacto de los cambios, de forma que es mucho menos arriesgado probar nuevas características, corregir errores e incorporar las contribuciones de los colaboradores.

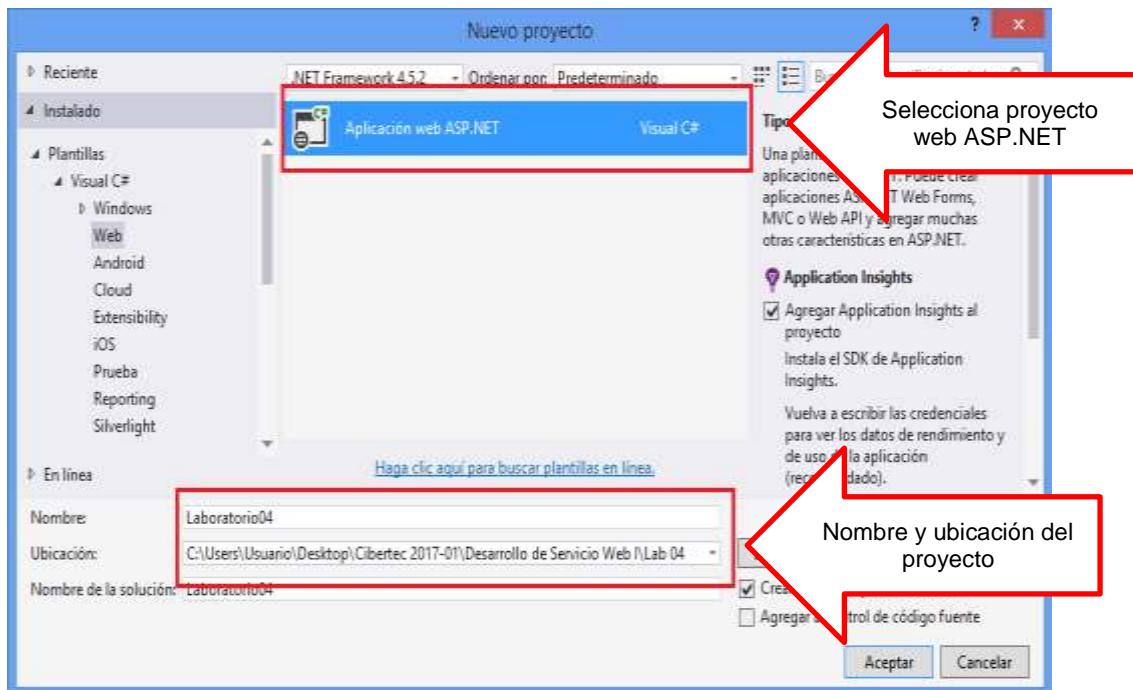
## Laboratorio 4.1

### Aplicación ASP.NET MVC, listado con Vista Parcial

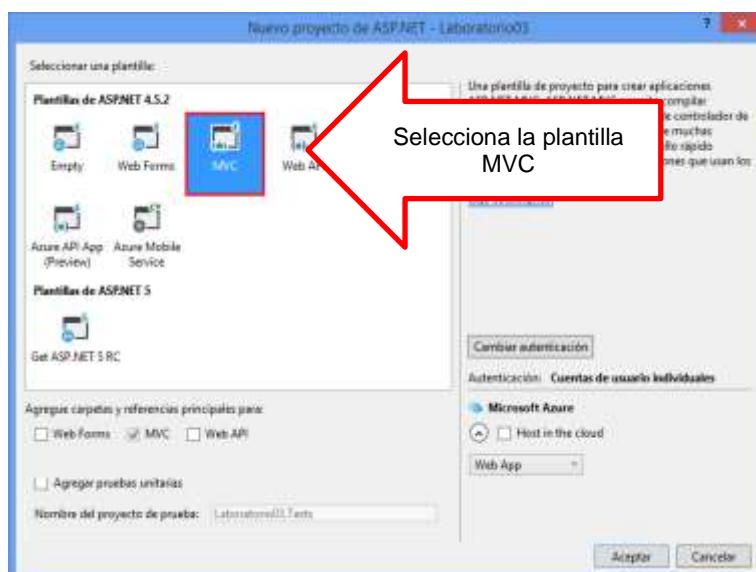
En el mismo proyecto ASP.NET MVC realice los procesos de listado y consulta a la tabla tb\_clientes almacenada en la base de datos Negocios2017.

#### SOLUCION

Crear un nuevo proyecto tipo Aplicación web ASP.NET, tal como se muestra.

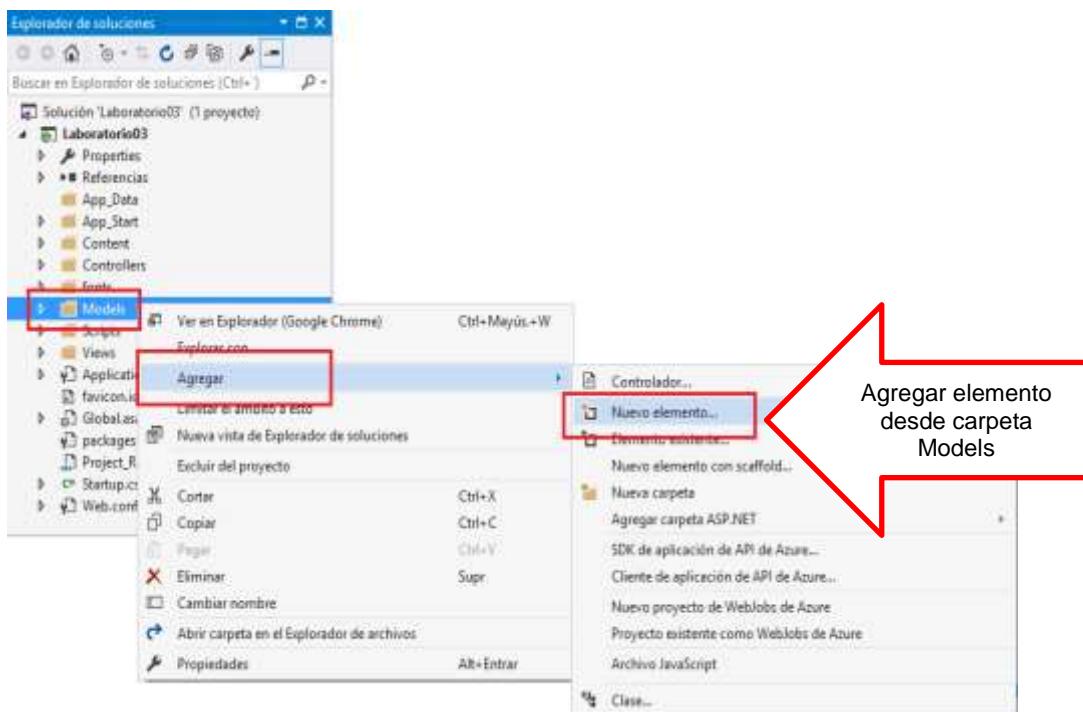


Selecciona la plantilla del proyecto web: MVC, tal como se muestra. Para continuar presionar el botón AGREGAR

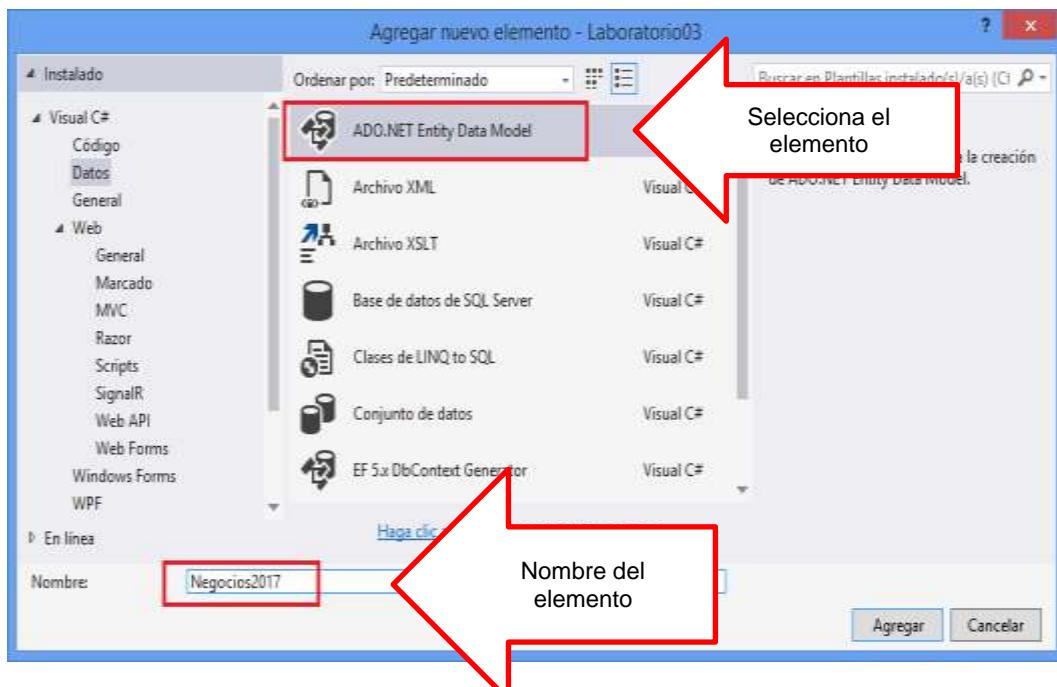


## Modelo de Datos

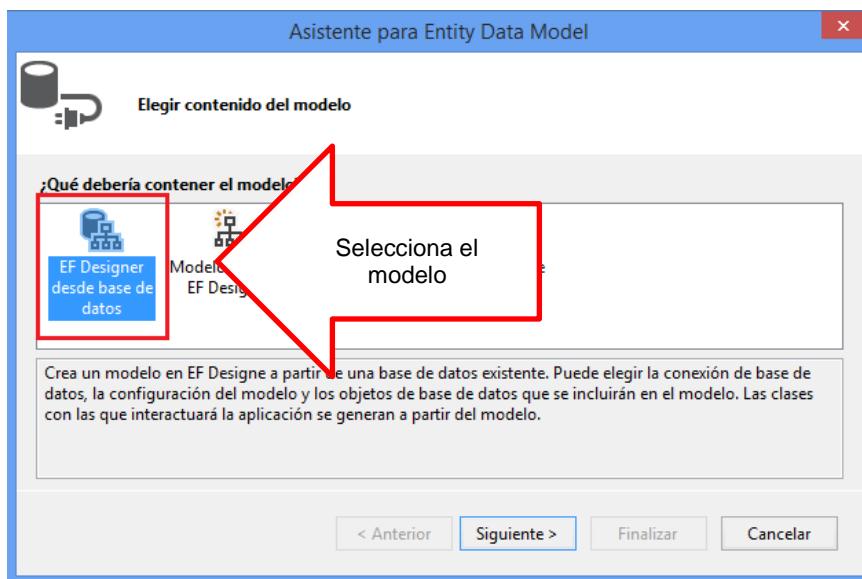
A continuación agregamos el modelo ADO Entity Model. Desde la carpeta **MODELS**, selecciona la opción **AGREGAR → Nuevo elemento**, tal como se muestra.



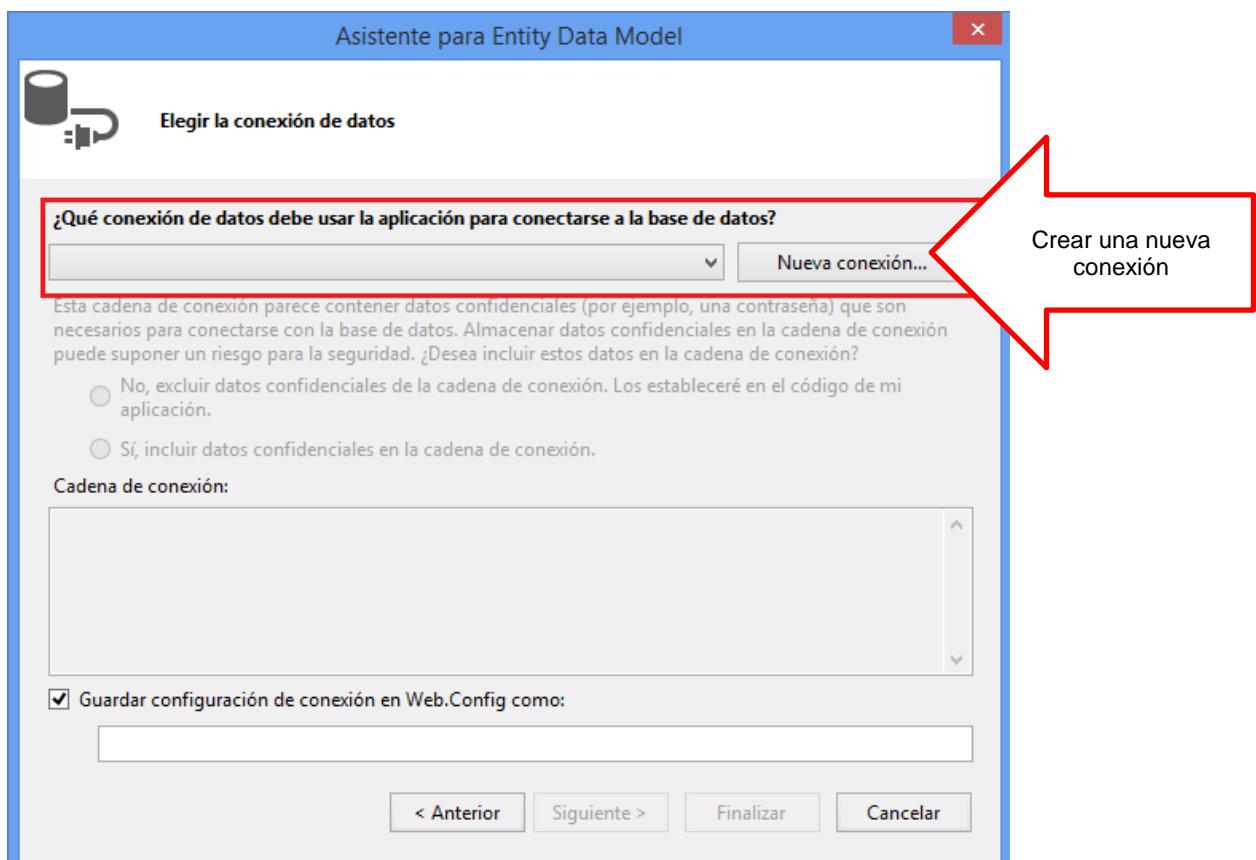
Selecciona, desde la opción Datos, el elemento ADO.NET Entity Data Model, tal como se muestra. Asigne el nombre al elemento: Negocios2017. Presiona el botón AGREGAR



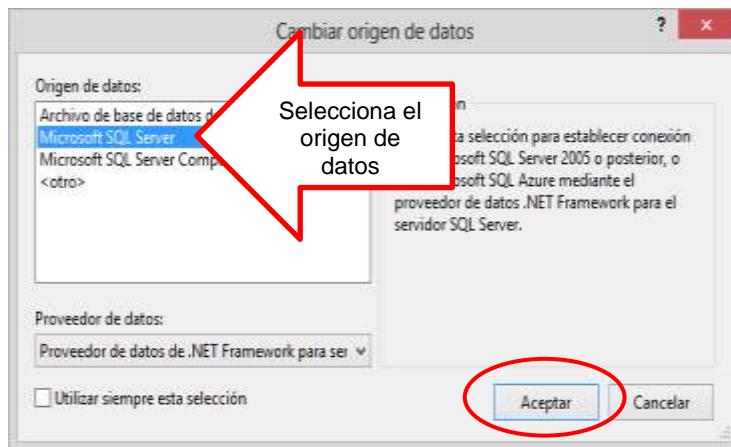
1. Elija el contenido del modelo, para ello selecciona la opción **GENERAR DESDE LA BASE DE DATOS**. Presione el botón Siguiente.



2. Elija la conexión, para crear una nueva conexión, presiona el botón **Nueva conexión**



3. Selecciona el origen de datos: **Microsoft SQL Server**, presiona el botón ACEPTAR, tal como se muestra.

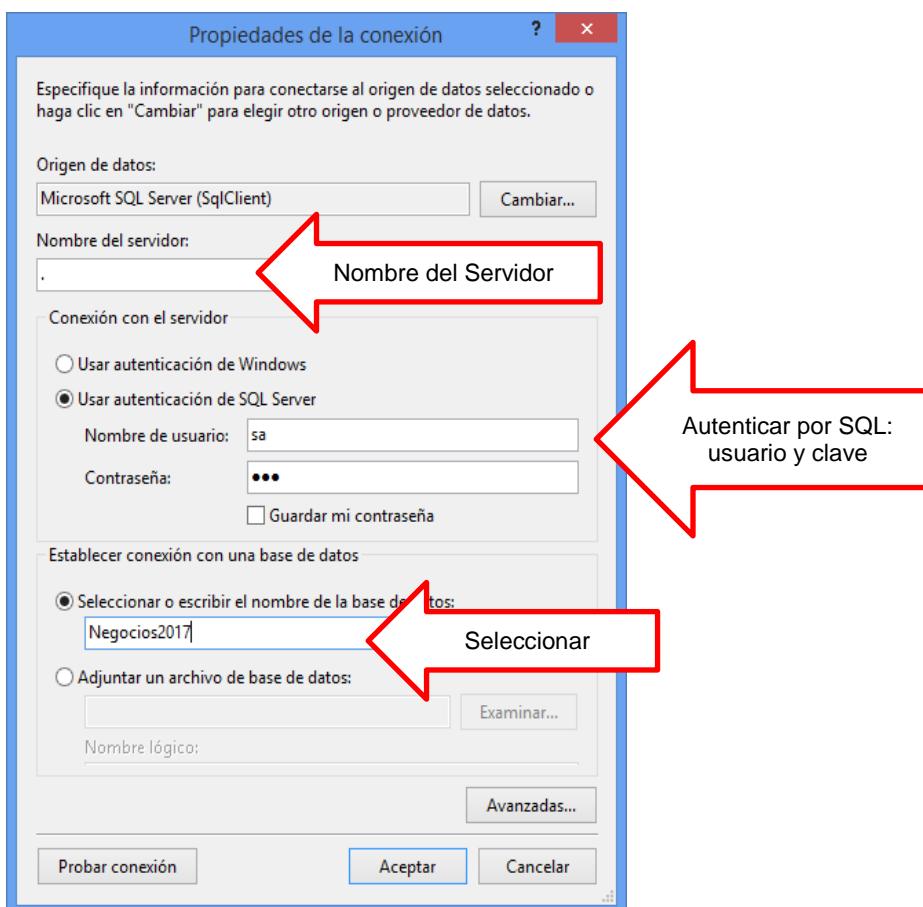


4. Defina las propiedades de la conexión:

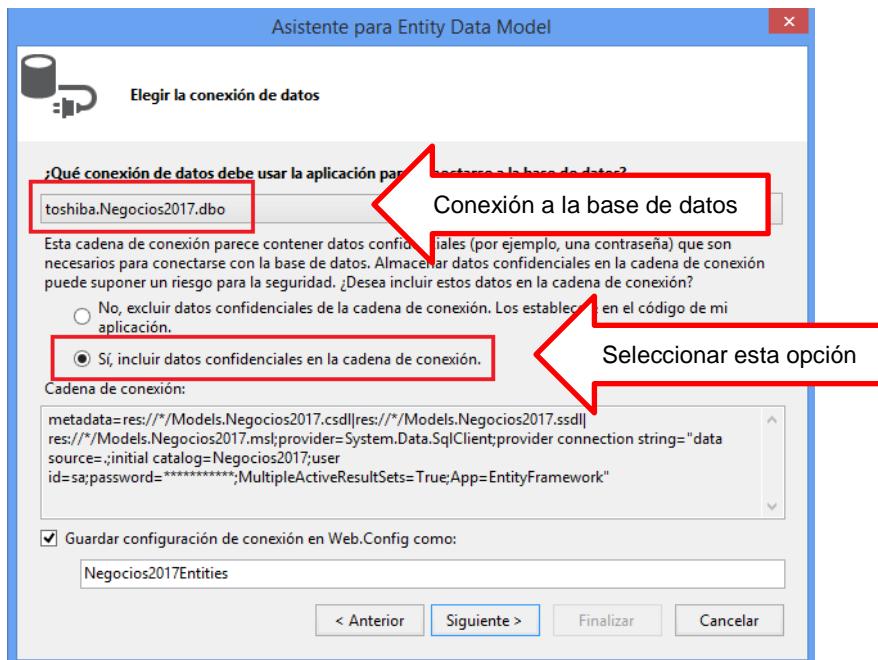
Nombre del servidor,

Autenticación, si es por SQL Server, ingrese el usuario y la contraseña.

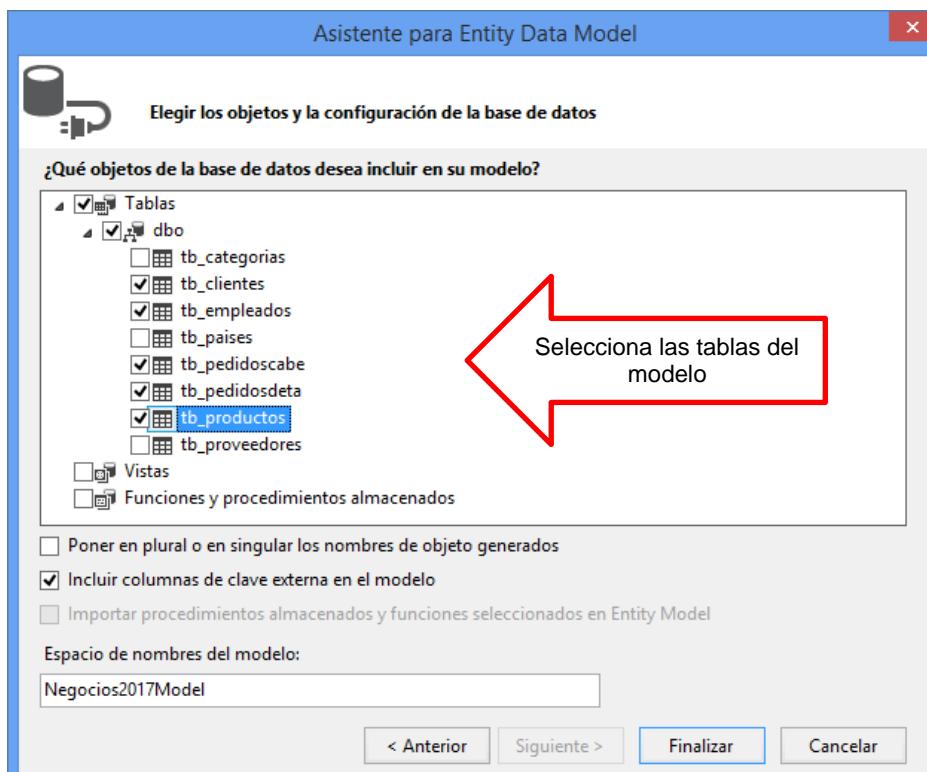
Selecciona la base de datos a trabajar



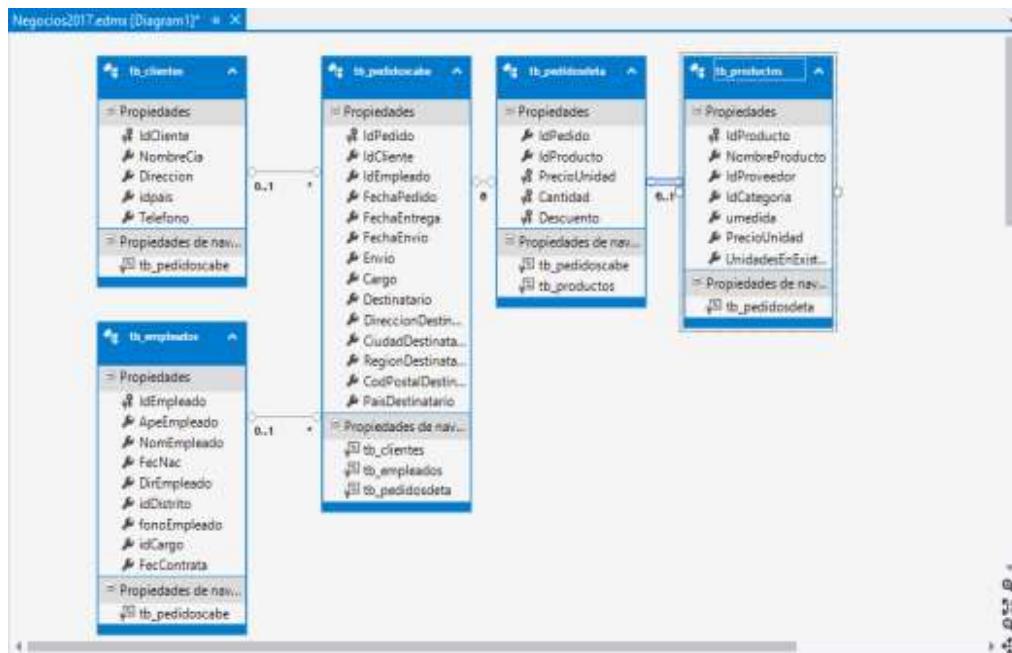
5. Habiendo definido la conexión, ésta se visualiza nueva en la ventana Elegir la conexión de datos, tal como se muestra. A continuación presiona el botón Siguiente.



6. Selecciona las tablas que incluirás en el modelo, tal como se muestra. Al terminar presiona el botón FINALIZAR.

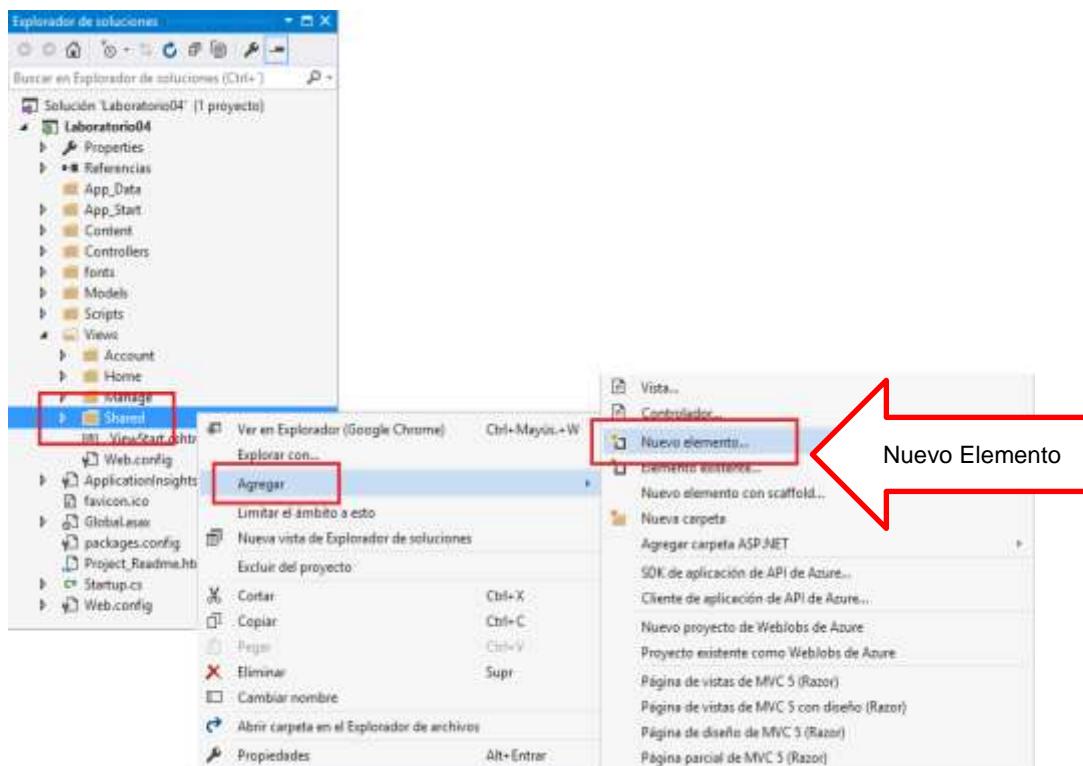


Al finalizar se genera el Diagrama del modelo, tal como se muestra. COMPILE LA SOLUCION (presiona la combinación Ctrl + May + B), a continuación

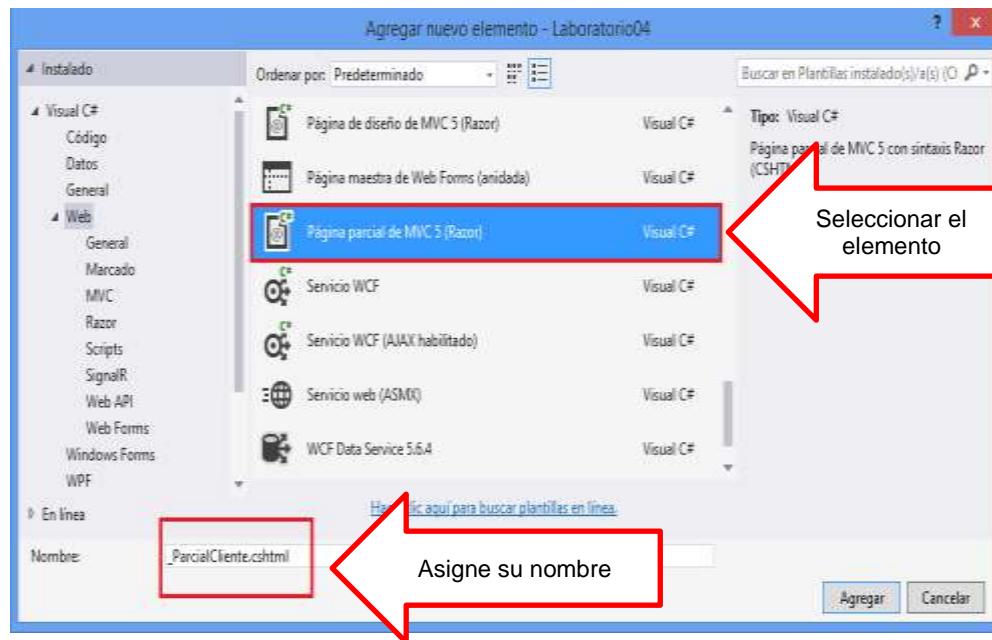


## Creando la Vista Parcial

Desde la carpeta Shared, vamos a agregar una página parcial, para ello selecciona desde la carpeta Agregar →Nuevo Elemento, tal como se muestra



En la ventana Agregar nuevo elemento, selecciona el ítem Página parcial de MVC 5 (Razor), y asigne un nombre, tal como se muestra



En la Vista Parcial, defina la sintaxis para listar los registros de tb\_clientes. Primero defina el modelo de la Vista, luego la sintaxis del listado

```

@model IEnumerable<Laboratorio04.Models.tb_clientes>


| id | Codigo del Cliente | Nombre del Cliente | Direccion | Pais de residencia | Telefono |
|----|--------------------|--------------------|-----------|--------------------|----------|
|----|--------------------|--------------------|-----------|--------------------|----------|


```

## Trabajando con Controlador y su ActionResult

Defina el controlador NegociosController. Dentro del controlador, referencia la carpeta Models (modelo de datos), instancia el Contexto de Datos y crea el ActionResult Cliente, el cual retorna a la Vista la lista de los registros de tb\_clientes

```

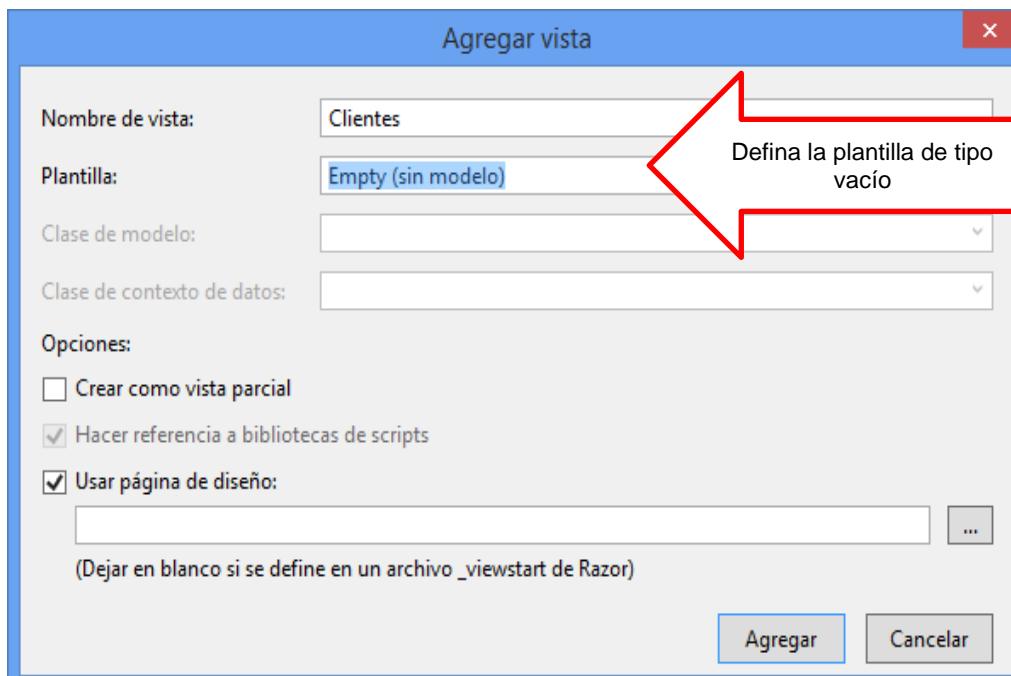
NegociosController.cs*  X
Laboratorio04 Controllers.NegociosController.cs  bd
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Laboratorio04.Models;

namespace Laboratorio04.Controllers
{
    public class NegociosController : Controller
    {
        // instancia del Contexto
        Negocios2017Entities bd = new Negocios2017Entities();

        public ActionResult Clientes()
        {
            return View(bd.tb_clientes.ToList());
        }
    }
}

```

A continuación agregamos una Vista al ActionResult: la plantilla será de tipo Empty. Presiona el botón Agregar



En la Vista, defina el modelo de datos del listado. Para visualizar los registros, invocar la vista parcial \_ParcialCliente, tal como se muestra

The screenshot shows a portion of a C# code editor. A red box highlights the declaration of a model at the top of the file:

```
Clientes.cshtml > X NegociosController.cs
@model IEnumerable<Laboratorio04.Models.tb_clientes>
```

A red arrow points from this box to a call to a partial view named '\_ParcialCliente' located further down in the code:

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



## Listado de Clientes


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

A red arrow points from this call to the text 'Invocar la vista parcial'.

Para ejecutar la vista, presiona la combinación Ctrl + F5, donde se visualiza la lista de registros.

The screenshot shows a web browser window titled 'Cuentas - Mi aplicación'. The address bar shows 'localhost:60482/Negocios/Cuentas'. The page content is titled 'Listado de Clientes' and displays a table of client records. Red boxes highlight the table header and the first few rows of data.

id	Código del Cliente	Nombre del Cliente	Dirección	País de residencia	Teléfono
1	AA455	Aaron Alvarez Garcia	Av. Lima 46633	España	9556655
2	AB123	Abelardo Garcia	Jr. Cusco 1222	Perú	5663232
3	ALFKI	Alfreds Futterkiste	Obere Str. 57	Argentina	030-0074321
4	ANATR	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	España	(5) 555-4729
5	ANTON	Antonio Moreno Taquería	Mataderos 2312	Colombia	(5) 555-3932
6	AROUT	Around the Horn	120 Hanover Sq.	USA	(71) 555-7788
7	BERGS	Berglunds snabbköp	Berguvsvägen 8	Francia	0921-12 34 65
8	BLAUS	Blauer See Delikatessen	Forsterstr. 57	Perú	0621-08460
9	BLONP	Blondel pére et fils	24, place Kleber Estrasburgo	Canadá	88 60 15 31
10	BOLID	Bolido Comidas preparadas	C/ Araquí, 67	China	(91) 555 91 99
11	BONAP	Bon app	12, rue des Bouchers	Perú	91 24 45 41

## Laboratorio 4.2

### Aplicación ASP.NET MVC, paginación con Vista Parcial

En el mismo proyecto ASP.NET MVC realice los procesos de paginación de los registros de la tabla tb\_clientes almacenada en la base de datos Negocios2017

#### Definiendo el ActionResult PaginacionClientes

Defina el ActionResult PaginacionClientes, cuyo parámetro es el número de la página. En dicho proceso, almacenamos el número de la página seleccionada; y definimos las variables reginicio y regfin los cuales indican los registros que se guardarán en la variable lista de tipo List<tb\_clientes> y se visualizarán en la Vista

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Negocios2017Entities;
using System.Web.Mvc;

namespace Laboratorio04.Controllers
{
    public class NegociosController : Controller
    {
        Negocios2017Entities bd = new Negocios2017Entities();

        public ActionResult Clientes()
        {
            return View();
        }

        public ActionResult PaginacionClientes(int? pag)
        {
            int c = bd.tb_clientes.Count();
            ViewBag.numreg = c % 10 != 0 ? c / 10 + 1 : c / 10;

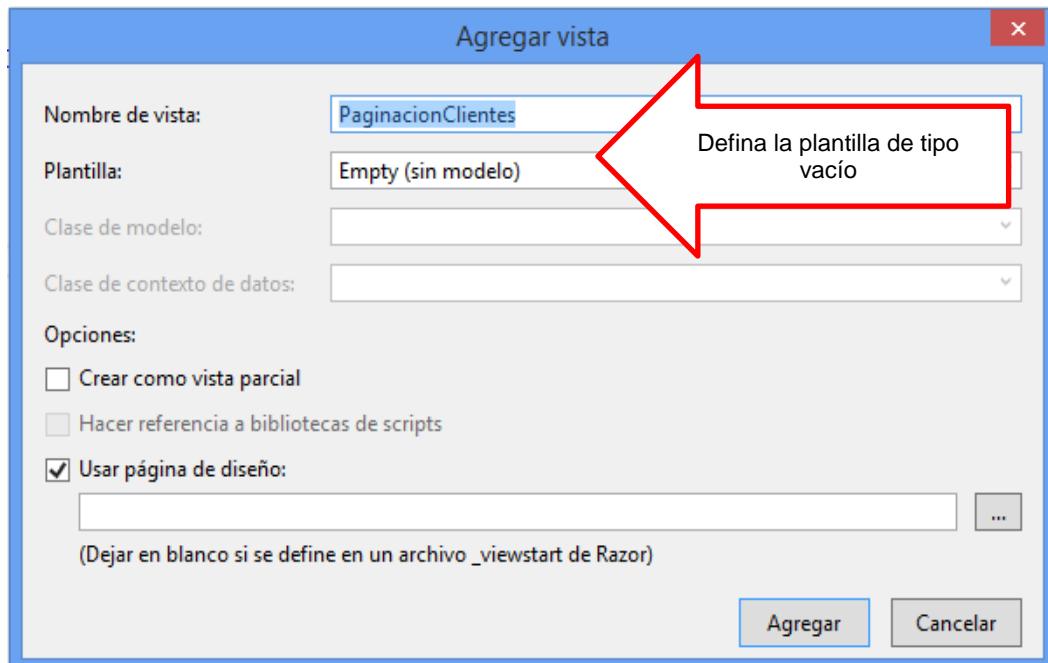
            int pageact = pag == null ? 0 : (int)pag;
            int reginicio = pageact * 10;
            int regfin = reginicio + 10;

            List<tb_clientes> lista = new List<tb_clientes>();
            for (int i = reginicio; i < regfin; i++)
            {
                if (i == c) break;
                lista.Add(bd.tb_clientes.ToList()[i]);
            }

            return View(lista);
        }
    }
}

```

A continuación, agregamos la Vista al ActionResult: plantilla de tipo Empty, tal como se muestra. Presiona el botón AGREGAR



En la vista, agrega un bloque <div>, el cual colocaremos los botones de la paginación, a través del @Html.ActionLink, donde se visualiza el número de la página, direccionalo al ActionResult y enviando el número de la página (pag), tal como se muestra

```

@model IEnumerable<Laboratorio04.Models.tb_clientes>
@{
    ViewBag.Title = "PaginacionClientes";
}



## Paginacion de Clientes


@Html.Partial("_ParcialCliente");



@for (int i = 0; i < (int)ViewBag.numreg; i++)
    {
        @Html.ActionLink((i + 1).ToString(), "PaginacionClientes", new { pag = i },
                        new { @class = "btn btn-primary" });
    }


```

Ejecuta la Vista, donde se visualiza los registros de clientes agrupados de 10 en 10, visualizando la paginación en la parte inferior.

The screenshot shows a web application window titled "PaginacionCientes - Mi aplicación". The address bar displays "localhost:60482/Negocios/PaginacionCientes?pag=0". The page header includes a logo, the title "Nombre de aplicación", and navigation links for "Inicio", "Acerca de", "Contacto", "Registrarse", and "Iniciar sesión". Below the header is a table titled "Paginacion de Clientes" with 10 rows of client data. At the bottom of the table is a blue navigation bar with numbered buttons from 1 to 10, indicating the current page is 1. The table columns are: #, Código del Cliente, Nombre del Cliente, Dirección, País de residencia, and Teléfono.

#	Código del Cliente	Nombre del Cliente	Dirección	País de residencia	Teléfono
1	AA455	Aaron Alvarez Garcia	Av. Lima 46633	España	9556655
2	AB123	Abelardo Garcia	Jr. Cusco 1222	Peru	5663232
3	ALFKI	Alfreds Futterkiste	Obere Str. 57	Argentina	030-0074321
4	ANATR	Ana Trujillo Emparedados y helados	Avda. de la Constitucion 2222	España	(5) 555-4729
5	ANTON	Antonio Moreno Taqueria	Mataderos 2312	Colombia	(5) 555-3932
6	AROUT	Around the Horn	120 Hanover Sq.	USA	(71) 555-7788
7	BERGS	Berglunds snabbköp	Berguvsvägen 8	Francia	0921-12 34 65
8	BLAUS	Blauer See Delikatessen	Forsterstr. 57	Peru	0621-08460
9	BLONP	Blondel père et fils	24, place Kleber Estrasburgo	Canada	88.60.15.31
10	BOLID	Bolido Comidas preparadas	C/ Araquil, 67	China	(91) 555 91 99

## Laboratorio 4.3

### Aplicación ASP.NET MVC, consulta y paginación con Vista Parcial

En el mismo proyecto ASP.NET MVC realice los procesos de paginación de los registros de la tabla tb\_clientes almacenada en la base de datos Negocios2017

#### Definiendo el ActionResult PaginacionClientePais

Defina el ActionResult PaginacionClientesPais, cuyo parámetro es el número de la página y el idpais (filtro de los registros).

En dicho proceso, almacenamos el número de la página seleccionada; y definimos las variables reginicio y regfin los cuales indican los registros que se guardarán en la variable lista de tipo List<tb\_clientes> y se visualizarán en la Vista

```

public class NegociosController : Controller
{
    Negocios2017Entities bd = new Negocios2017Entities();

    public ActionResult Clientes()...
    public ActionResult PaginacionClientePais(string idpais, int ? pag)
    {
        ViewBag.paises = new SelectList(bd.tb_paises.ToList(), "idpais", "nombrepais", idpais);

        List<tb_clientes> filtro = bd.tb_clientes.Where(it => it.idpais == idpais).ToList();

        int c = filtro.Count();
        ViewBag.numreg = c % 10 != 0 ? c / 10 + 1 : c / 10;

        int pageact = pag == null ? 0 : (int)pag;
        int reginicio = pageact * 10;
        int regfin = reginicio + 10;

        List<tb_clientes> lista = new List<tb_clientes>();
        for (int i = reginicio; i < regfin; i++)
        {
            if (i == c) break;
            lista.Add(filtro[i]);
        }

        return View(lista);
    }
}

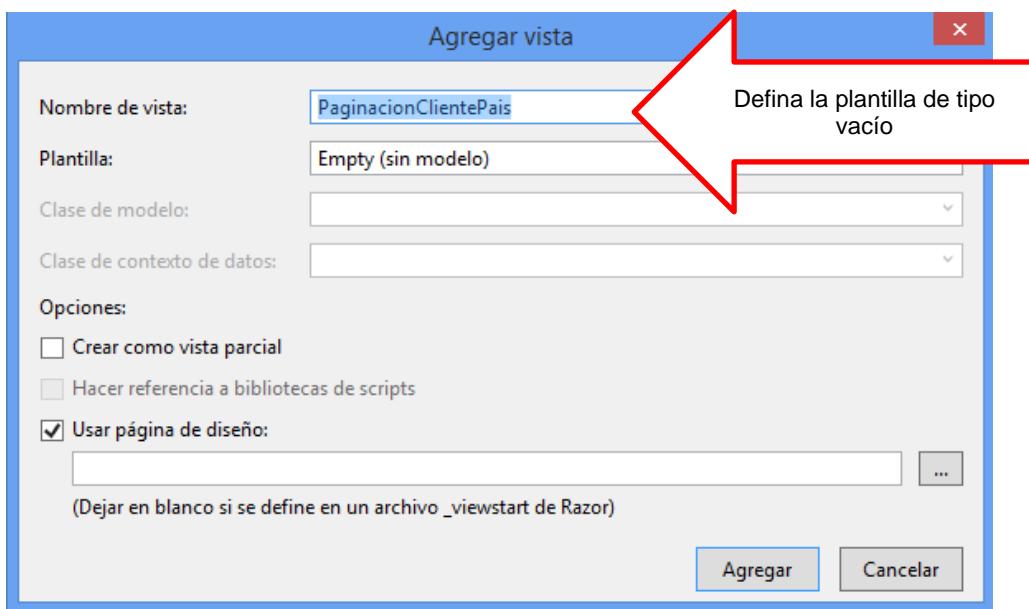
```

ViewBag almacena la lista de países

Filtro almacena los clientes por su condición

Defino la variable lista la cual almacena un grupo de 10 registros, definidos por la variable pag

A continuación, agregamos la Vista al ActionResult: plantilla de tipo Empty, tal como se muestra. Presiona el botón AGREGAR



En la vista, definimos el bloque BeginForm, el cual se visualiza la lista de los países. Al seleccionar un país, listamos los registros de tb\_clientes definido en la vista parcial. Agrega un bloque <div>, el cual colocaremos los botones de la paginación, a través del @Html.ActionLink, donde se visualiza el número de la página, direccinando al ActionResult y enviando el número de la página (pag) y el código del país la cual está almacenada en el ViewBag.idpais, tal como se muestra

```

@model IEnumerable<Laboratorio04.Models.tb_clientes>
@{
    ViewBag.Title = "PaginacionClientePais";
}



## Consulta de Clientes por País


@using (Html.BeginForm("PaginacionClientePais", "Negocios", FormMethod.Post))
{
    <span>Selecciona un País</span>
    @Html.DropDownList("idpais", (SelectList)ViewBag.paises)
    <input type="submit" value="Consulta" />
}

@Html.Partial("_ParcialCliente")



@for (int i = 0; i < (int)ViewBag.numreg; i++)
    {
        @Html.ActionLink((i + 1).ToString(), "PaginacionClientePais", new { pag = i, idpais=ViewBag.idpais }, new { @class = "btn btn-primary" });
    }


```

Ejecuta la Vista, al seleccionar un país se visualiza los registros de clientes agrupados de 10 en 10, visualizando la paginación en la parte inferior

The screenshot shows a web browser window titled "PaginacionClientePais - |" with the URL "localhost:60482/Negocios/PaginacionClientePais". The page has a header with "Nombre de aplicación", "Inicio", "Acerca de", "Contacto", "Registrarse", and "Iniciar sesión". Below the header, the title "Consulta de Clientes por País" is displayed. A dropdown menu "Selecciona un País" is set to "Colombia", and a button "Consulta" is visible. The main content is a table with 10 rows of client data:

id	Código del Cliente	Nombre del Cliente	Dirección	País de residencia	Teléfono
1	ANTON	Antonio Moreno Taquería	Mataderos 2312	Colombia	(5) 555-3932
2	DUMON	Du monde entier	67, rue des Cinquante Otages	Colombia	40.67.89.89
3	GODOS	Godos Cocina Típica	C/ Romero, 33	Colombia	
4	KOENE	Königlich Essen	Mauelstr. 90	Colombia	0555-09876
5	LAUGB	Laughing Bacchus Wine Cellars	1900 Oak St.	Colombia	(604) 555-7293
6	MAGAA	Magazzini Alimentari Riuniti	Via Ludovico il Moro 22	Colombia	035-640231
7	RICAR	Ricardo Adocicados	Av. Copacabana, 267	Colombia	
8	THECR	The Cracker Box	55 Grizzly Peak Rd.	Colombia	(406) 555-8083
9	VICTE	Victuailles en stock	2, rue du Commerce	Colombia	78.32.54.87
10	WILMK	Wilman Kala	Keskuskatu 45	Colombia	90-224 8858

A small blue button labeled "1" is located at the bottom left of the table area, indicating the current page of the pagination.

# Resumen

- En el modelo Model-View-Controller (MVC), las vistas están pensadas exclusivamente para encapsular la lógica de presentación. No deben contener lógica de aplicación ni código de recuperación de base de datos. El controlador debe administrar toda la lógica de aplicación. Una vista representa la interfaz de usuario adecuada usando los datos que recibe del controlador. Estos datos se pasan a una vista desde un método de acción de controlador usando el método View.
- ASP.NET MVC ha tenido el concepto de motor de vistas (View Engine), las cuales realizan tareas sólo de presentación. No contienen ningún tipo de lógica de negocio y no acceden a datos. Básicamente se limitan a mostrar datos y a solicitar datos nuevos al usuario. ASP.NET MVC se ha definido una sintaxis que permite separar la sintaxis de servidor usada, del framework de ASP.NET MVC, es lo que llamamos un motor de vistas de ASP.NET MVC, el cual viene acompañado de un nuevo motor de vistas, llamado Razor.
- La clase HtmlHelper proporciona métodos que ayudan a crear controles HTML mediante programación. Todos los métodos HtmlHelper generan HTML y devuelven el resultado como una cadena. Los métodos de extensión para la clase HtmlHelper están en el namespace System.Web.Mvc.Html. Estas extensiones añaden métodos de ayuda para la creación de formas, haciendo controles HTML, renderizado vistas parciales, la validación de entrada.
- Hay HTML Helpers para toda clase de controles de formulario Web: check boxes, hidden fields, password boxes, radio buttons, text boxes, text areas, DropDownList lists y list boxes.
- Hay también un HTML Helper para el elemento Label, los cuales nos asocian descriptivas de texto a un formulario Web. Cada HTML Helper nos da una forma rápida de crear HTML válido para el lado del cliente.
- Una vista parcial permite definir una vista que se representará dentro de una vista primaria. Las vistas parciales se implementan como controles de usuario de ASP.NET (.ascx). Cuando se crea una instancia de una vista parcial, obtiene su propia copia del objeto ViewDataDictionary que está disponible para la vista primaria. Por lo tanto, la vista parcial tiene acceso a los datos de la vista primaria. Sin embargo, si la vista parcial actualiza los datos, esas actualizaciones solo afectan al objeto ViewData de la vista parcial. Los datos de la vista primaria no cambian
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
  - <https://msdn.microsoft.com/es-es/library/windows/apps/jj883732.aspx>
  - <http://speakingin.net/2011/01/30/asp-net-mvc-3-sintaxis-de-razor-y/>
  - <http://www.julitogtu.com/2013/02/07/asp-net-mvc-cargar-una-vista-parcial-con-ajax/>
  - [https://msdn.microsoft.com/es-pe/library/dd410123\(v=vs.100\).aspx](https://msdn.microsoft.com/es-pe/library/dd410123(v=vs.100).aspx)
  - <http://andresfelipetrujillo.com/2014/08/08/aprendiendo-asp-net-mvc-4-parte-5-vistas/>





# TRABAJANDO CON DATOS EN ASP.NET MVC

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla aplicaciones con acceso a datos teniendo implementando procesos de consulta y actualización de datos

## TEMARIO

### Tema 5: Arquitectura “N” capas orientadas al Dominio (5 horas)

- 5.1 Introducción
- 5.2 Capas de la arquitectura DDD
- 5.3 Implementando las capas de Dominio

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando la arquitectura “n” capas” orientadas al dominio en el patrón de diseño Modelo Vista Controlador
- Los alumnos desarrollan los laboratorios de esta semana

Pag 62



## 5. Arquitectura “N” capas orientadas al Dominio

### 5.1 Introducción

En una aplicación donde el diseño esta orientado al dominio (Domain design Driven o DDD), termino que introdujo Eric Evans en su libro, el dominio debe ser lo más importante de una aplicación, es su corazón.

El dominio es modelo de una aplicación y su comportamiento, donde se definen los procesos y la reglas de negocio al que esta enfocada la aplicación, es la funcionalidad que se puede hablar con un experto del negocio o analista funcional, esta lógica no depende de ninguna tecnología como por ejemplo si los datos se persisten en una base de datos, si esta base de datos es SQL Server, Neo4j, MongoDB o la que sea y lo que es más importante, esta lógica no debe ser reescrita o modificada porque se cambie una tecnología específica en una aplicación.

En un diseño orientado al dominio lo dependiente de la tecnología reside en el exterior como si capas de una cebolla fueran, donde podemos sustituir una capa por otra utilizando otra tecnología y la funcionalidad de la aplicación no se ve comprometida.

La arquitectura en capas es una buena forma de representar un diseño orientado al dominio, abstrayendo cada capa mediante interfaces, de forma que no haya referencias directas entre la implementación real de las capas, lo que nos va a permitir reemplazar capas en el futuro de una forma más segura y menos traumática.

En una arquitectura en capas el dominio reside en la capa más profunda o core, donde no depende de ninguna otra.

*Arquitectura N-Capas con Orientación al Dominio*

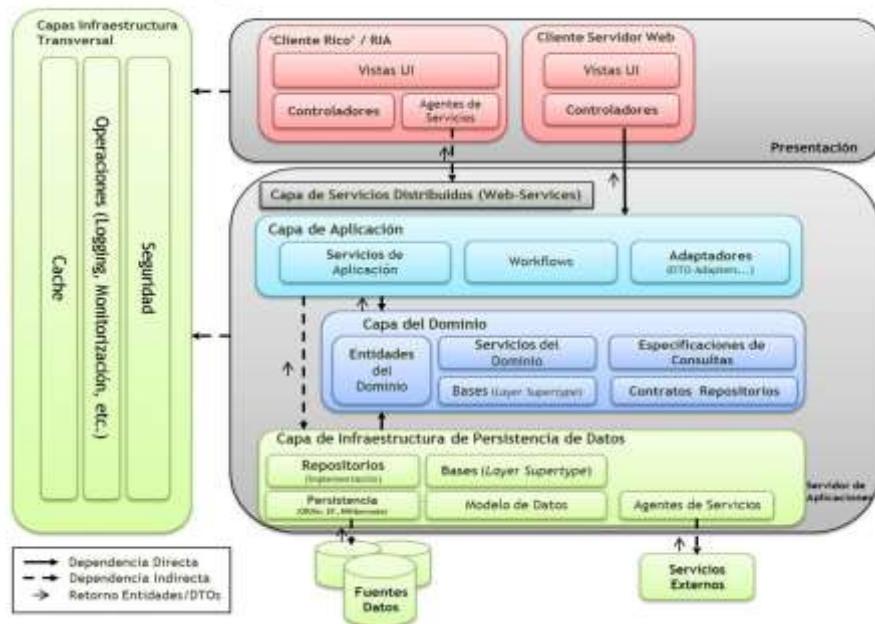


Figura 1: Arquitectura de capas orientadas al dominio  
<http://es.stackoverflow.com/questions/41889/asp-net-mvc-arquitectura-ddd-domain-driven-design>

DDD (Domain Driven Design) además de ser un estilo arquitectural, es una forma que permite desarrollar proyectos durante todo el ciclo de vida del proyecto. Sin embargo, DDD también identifica una serie de patrones de diseño y estilo de Arquitectura concreto.

DDD es una aproximación concreta para diseñar software basado en la importancia del Dominio del Negocio, sus elementos y comportamientos y las relaciones entre ellos. Está muy indicado para diseñar e implementar aplicaciones empresariales complejas donde es fundamental definir un Modelo de Dominio expresado en el propio lenguaje de los expertos del Dominio de negocio real (el llamado Lenguaje Único). El modelo de Dominio puede verse como un marco dentro del cual se debe diseñar la aplicación.

Esta arquitectura permite estructurar de una forma limpia y clara la complejidad de una aplicación empresarial basada en las diferentes capas de la arquitectura, siguiendo el patrón N-Layered y las tendencias de arquitecturas en DDD.

El patrón N-Layered distingue diferentes capas y sub-capas internas en una aplicación, delimitando la situación de los diferentes componentes por su tipología. Por supuesto, esta arquitectura concreta N-Layer se puede personalizar según las necesidades de cada proyecto y/o preferencias de Arquitectura. Simplemente proponemos una Arquitectura marco a seguir que sirva como punto base a ser modificada o adaptada por arquitectos según sus necesidades y requisitos.

En concreto, las capas y sub-capas propuestas para aplicaciones „N-Layered con Orientación al Dominio“ son:

- Capa de Presentación
  - Subcapas de Componentes Visuales (Vistas)
  - Subcapas de Proceso de Interfaz de Usuario (Controladores y similares)
- Capa de Servicios Distribuidos (Servicios-Web)
  - Servicios-Web publicando las Capas de Aplicación y Dominio
- Capa de Aplicación
  - Servicios de Aplicación (Tareas y coordinadores de casos de uso)
  - Adaptadores (Conversores de formatos, etc.)
  - Subcapa de Workflows (Opcional)
  - Clases base de Capa Aplicación (Patrón Layer-Supertype)
- Capa del Modelo de Dominio
  - Entidades del Dominio
  - Servicios del Dominio
  - Especificaciones de Consultas (Opcional)
  - Contratos/Interfaces de Repositorios
  - Clases base del Dominio (Patrón Layer-Supertype)
- Capa de Infraestructura de Acceso a Datos
  - Implementación de Repositorios"
  - Modelo lógico de Datos
  - Clases Base (Patrón Layer-Supertype)
  - Infraestructura tecnología ORM
  - Agentes de Servicios externos

## Interacción en la Arquitectura DDD

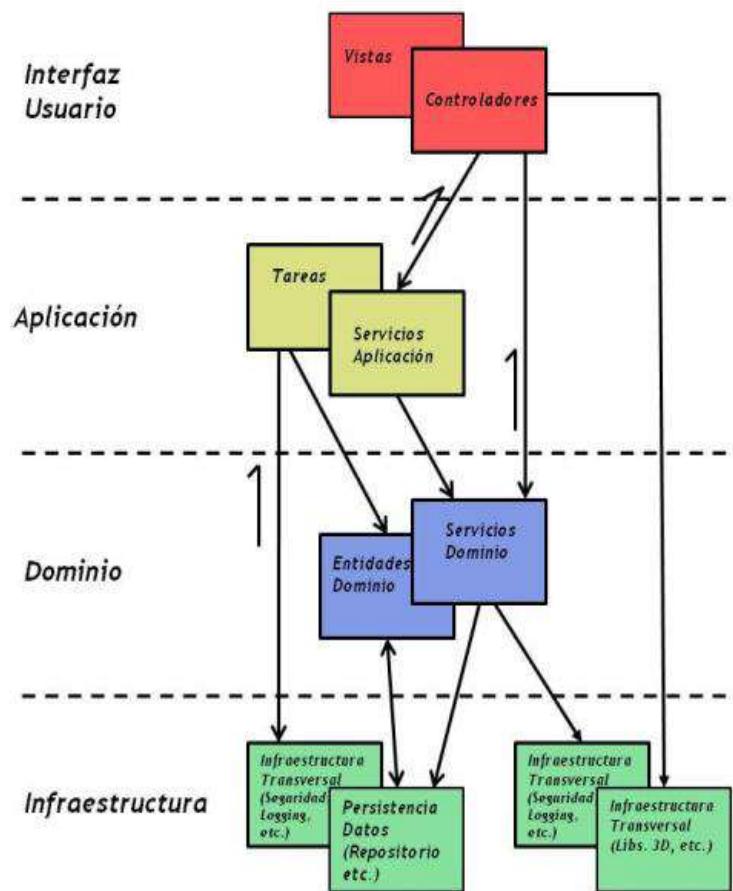


Figura 2: Interacción de la capas

Primeramente, podemos observar que la Capa de Infraestructura que presenta una arquitectura con tendencia DDD, es algo muy amplio y para muchos contextos muy diferentes (Contextos de Servidor y de Cliente).

La Capa de infraestructura contendrá todo lo ligado a tecnología/infraestructura. Ahí se incluyen conceptos fundamentales como Persistencia de Datos (Repositorios, etc.), pasando por aspectos transversales como Seguridad, Logging, Operaciones, etc. e incluso podría llegar a incluirse librerías específicas de capacidades gráficas para UX (librerías 3D, librerías de controles específicos para una tecnología concreta de presentación, etc.). Debido a estas grandes diferencias de contexto y a la importancia del acceso a datos, en nuestra arquitectura propuesta hemos separado explícitamente la Capa de Infraestructura de „Persistencia de Datos” del resto de capas de „Infraestructura Transversal”, que pueden ser utilizadas de forma horizontal/transversal por cualquier capa.

El otro aspecto interesante que adelantábamos anteriormente, es el hecho de que el acceso a algunas capas no es con un único camino ordenado por diferentes capas.

Concretamente podremos acceder directamente a las capas de Aplicación, de Dominio y de Infraestructura Transversal siempre que lo necesitemos. Por ejemplo, podríamos acceder directamente desde una Capa de Presentación Web (no necesita interfaces remotos de tipo Servicio-Web) a las capas inferiores que necesitemos (Aplicación, Dominio, y algunos aspectos de Infraestructura Transversal). Sin embargo, para llegar a la „Capa de Persistencia de Datos” y sus objetos Repositorios (puede recordar en algunos aspectos a la Capa de Acceso a Datos (DAL) tradicional, pero no es lo mismo),

es recomendable que siempre se acceda a través de los objetos de coordinación (Servicios) de la Capa de Aplicación, puesto que es la parte que los orquesta.

Queremos resaltar que la implementación y uso de todas estas capas debe ser algo flexible. Relativo al diagrama, probablemente deberían existir más combinaciones de flechas (accesos). Y sobre todo, no tiene por qué ser utilizado de forma exactamente igual en todas las aplicaciones.



## 5.2 Capas de la arquitectura DDD

1. UserInterface: Esta será nuestra capa de presentación. Aquí pondremos nuestro proyecto MVC, ASP, o lo que utilicemos como front-end de nuestra aplicación.
2. Application: Esta capa es la que nos sirve como nexo de unión de nuestro sistema. Desde ella se controla y manipula el domain. Por ejemplo, dada una persona se requiere almacenar información de un hijo que acaba de nacer. Esta capa se encargará de: llamar a los repositorios del domain para obtener la información de esa persona, instanciar los servicios del domain y demás componentes necesarios, y por ultimo persistir los cambios en nuestra base de datos.  
En esta capa también crearíamos interfaces e implementaciones para servicios, DTOs etc, en caso de que fuese necesario.
3. Domain: En domain podemos ver que hay 3 proyectos:
  - a. Entities: En el cual tendremos nuestras entidades. Es decir, es una clase donde se definen los atributos de la entidad.  
Una entidad tiene una clave que es única y la diferencia de cualquier otra entidad. Por ejemplo, para una clase Persona, podríamos tener los siguientes atributos: Nombre, Apellidos y fecha de nacimiento, en ellos tendríamos la información para la clase Persona.  
Una entidad no sólo se ha de ver como una simple clase de datos, sino que se ha de interpretar como una unidad de comportamiento, en la cual, además de las propiedades antes descritas, debe tener métodos como por ejemplo Edad(), el cual a través de la fecha de nacimiento tiene que ser capaz de calcular la edad. Esto es muy importante, ya que si las entidades las utilizamos simplemente como clases de datos estaremos cayendo en el antipatrón de modelo de datos anémico.
  - b. Domain: En este proyecto tendremos los métodos que no se ciñen a una entidad, sino que lo hacen a varias. Es decir, operaciones que engloben varias entidades.
  - c. Repositories: Aquí expondremos la colección de métodos que consumiremos desde nuestra capa application. En los repositories se va a instanciar las

entidades de nuestro dominio, pero no las implementa. Para eso ya tenemos la capa de infrastructure. Por ejemplo New, Update, Delete...

4. Infrastructure: Esta será la capa donde implementaremos repositorios, es decir, donde estarán las querys que ejecutaremos sobre la base de datos.

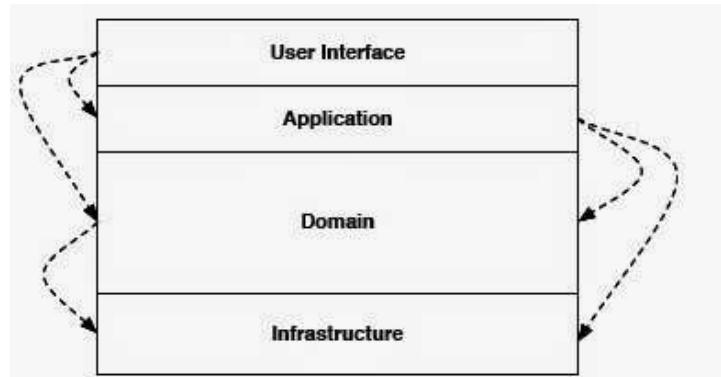


Figura 3: Capas en una aplicación MVC  
<http://nfanjul.blogspot.pe/2014/09/arquitectura-ddd-y-sus-capas.html>

### 5.3 Implementando la arquitectura DDD

Al utilizar ASP.NET MVC se aprovecha todas las ventajas que ofrece para la capa de presentación (HTML 5 + Razor), además de Entity Framework 4.5 Code First, data scaffolding y todos los beneficios que ofrece la aplicación de un patrón de diseño.

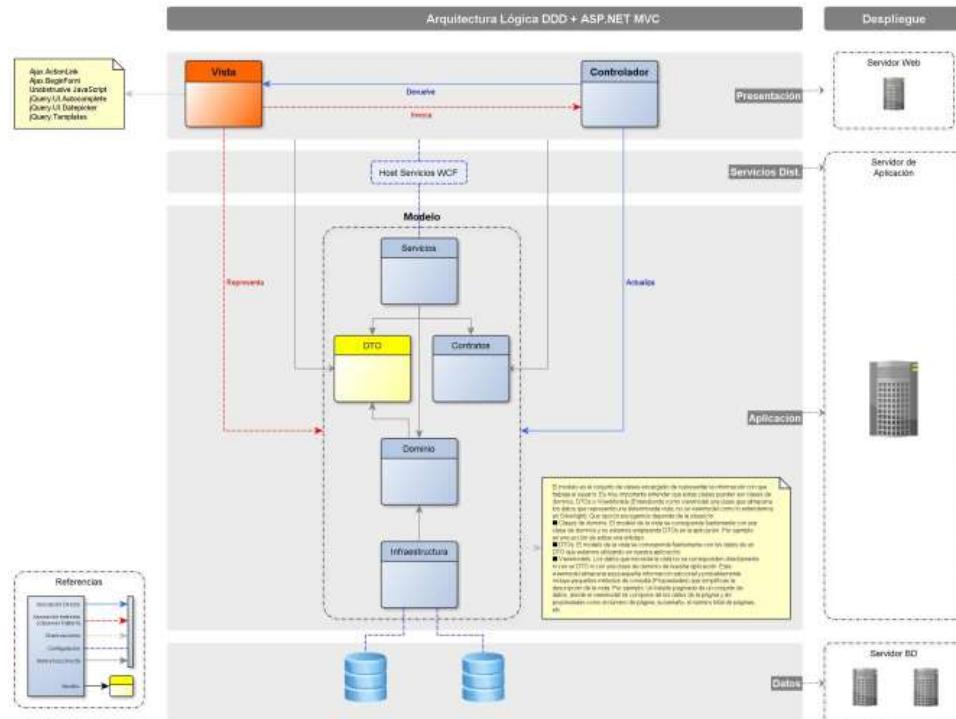


Figura4: Implementando Capas + aplicación MVC  
<https://jjestruch.wordpress.com/2012/02/21/arquitectura-ddd-domain-driven-design-asp-net-mvc/>

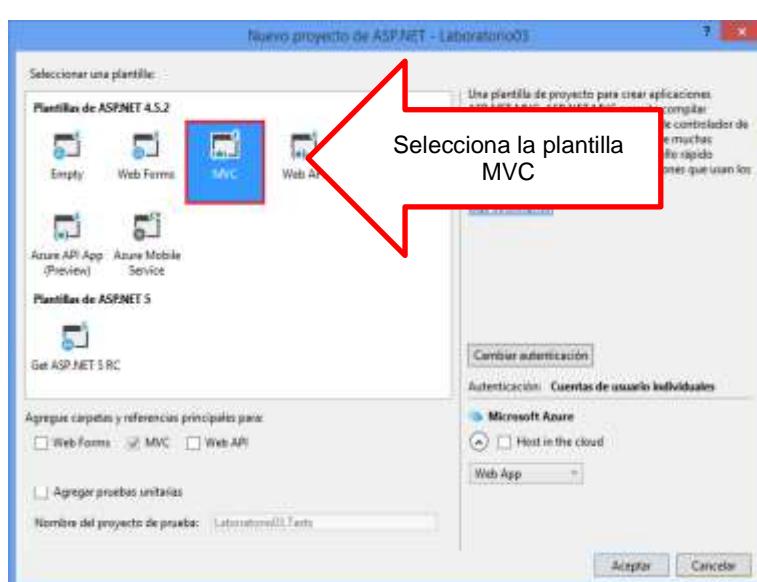
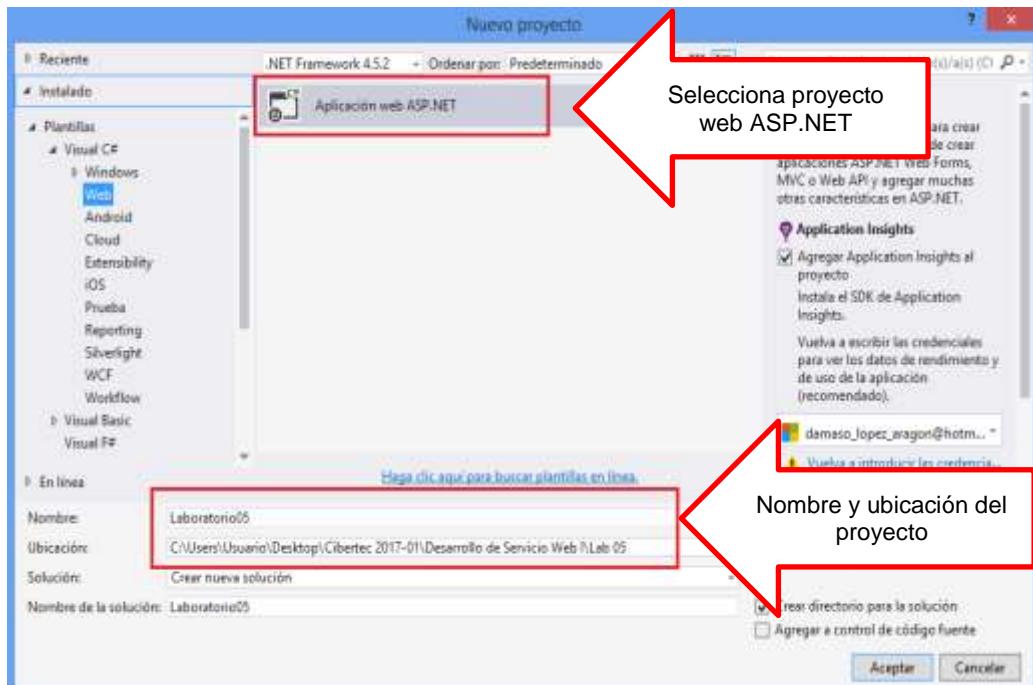
## Laboratorio 5.1

### Implementando consulta y actualización en ASP.NET MVC utilizando “N” capas

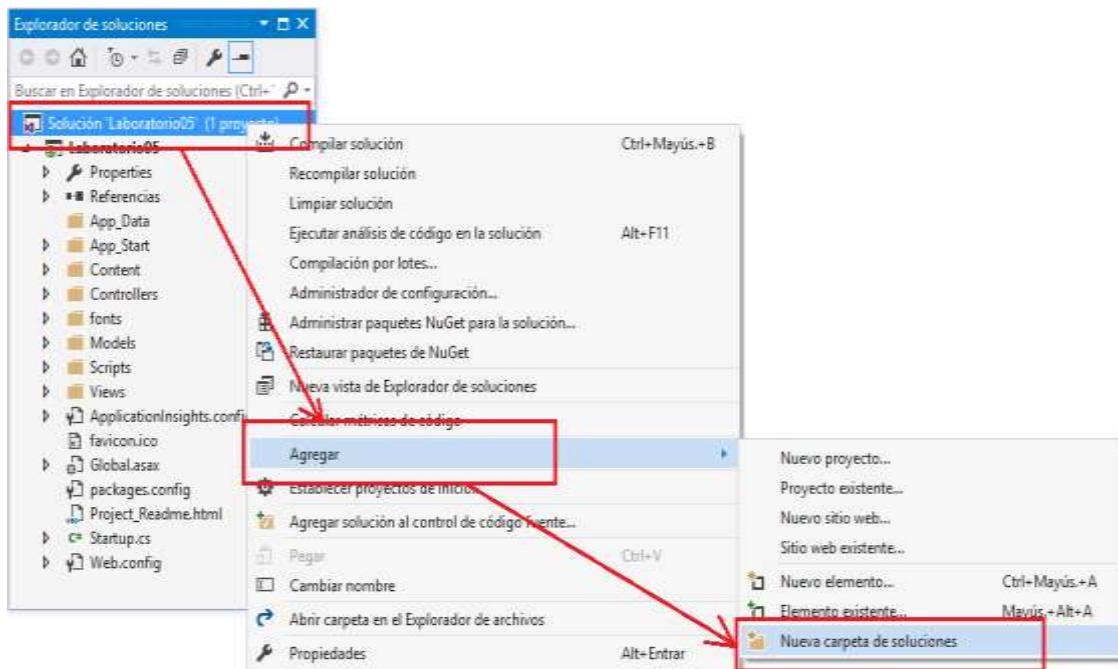
Implemente un proyecto ASP.NET MVC donde permita realizar las operaciones de consulta y actualización de datos utilizando arquitectura “N” capas.

#### SOLUCION

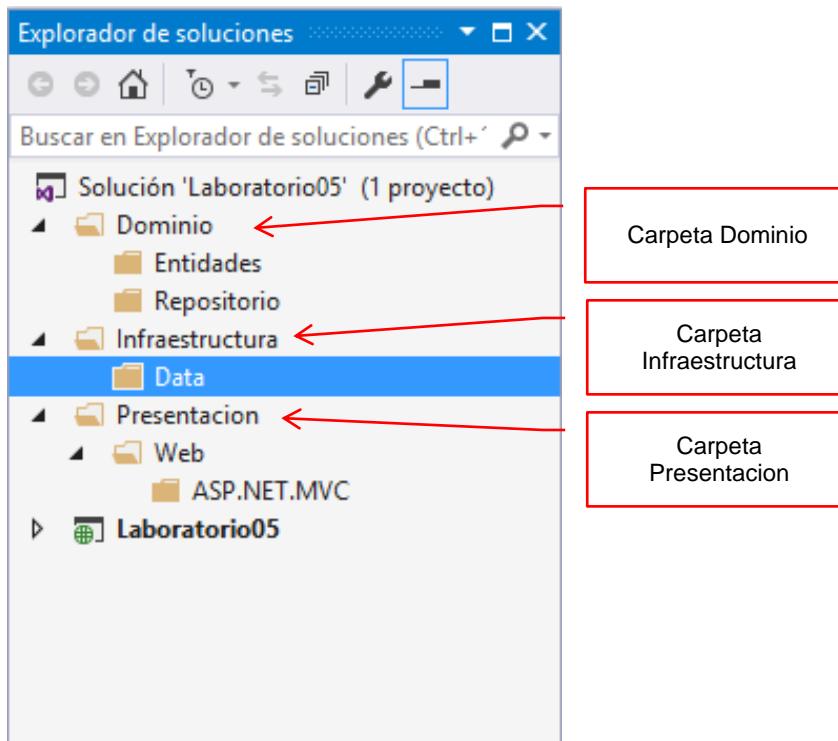
Crear un nuevo proyecto tipo Aplicación web ASP.NET, tal como se muestra.



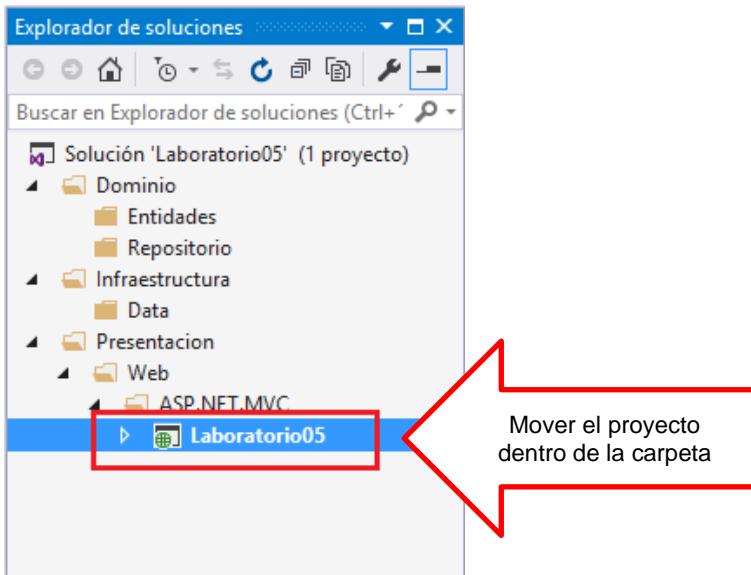
Para definir la arquitecturas de Capas de Dominio, agregamos las siguientes carpetas de solución a la Solucion Laboratorio05, tal como se muestra.



En la solución agregamos las siguientes carpetas y subcarpetas en la Solucion, tal como se muestra

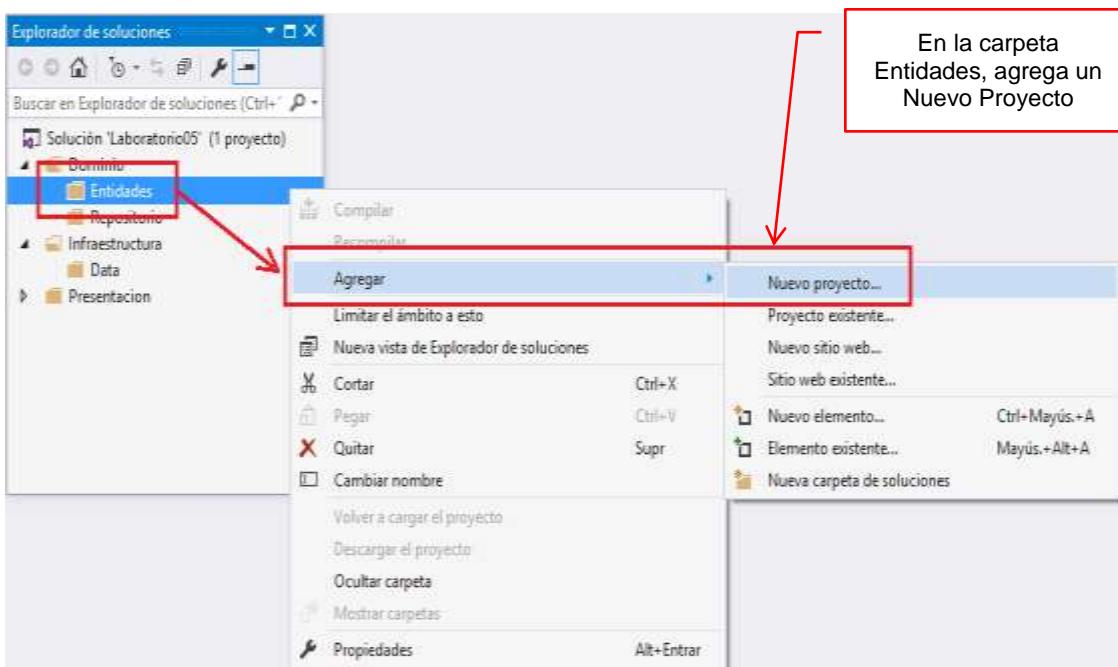


A continuación movemos el proyecto Laboratorio05 dentro de la carpeta ASP.NET.MVC, tal como se muestra

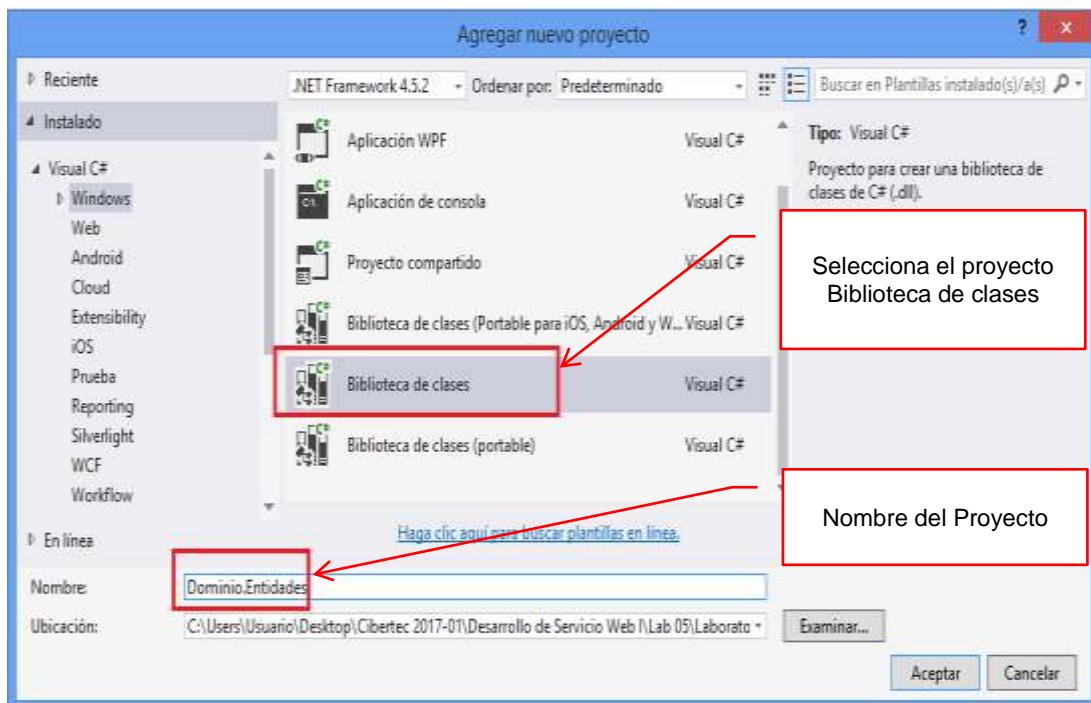


### Trabajando con la Entidades

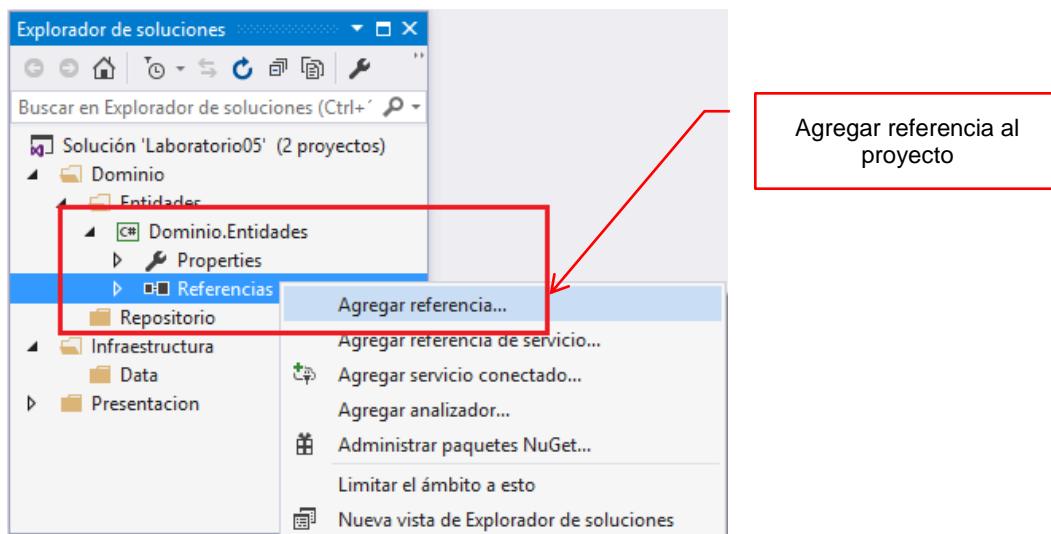
Para el desarrollo de la aplicación, debemos definir las entidades del proyecto. Para ello agrega dentro de la carpeta Entidades un Nuevo Proyecto, tal como se muestra.



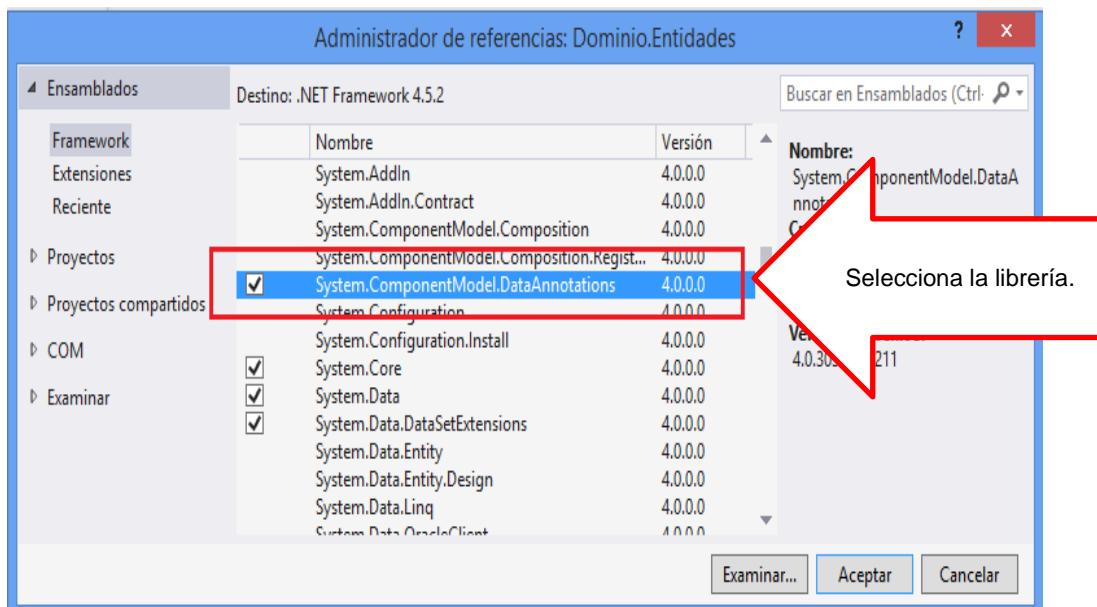
Selecciona el proyecto Biblioteca de clases, asigne el nombre Dominio.Entidades



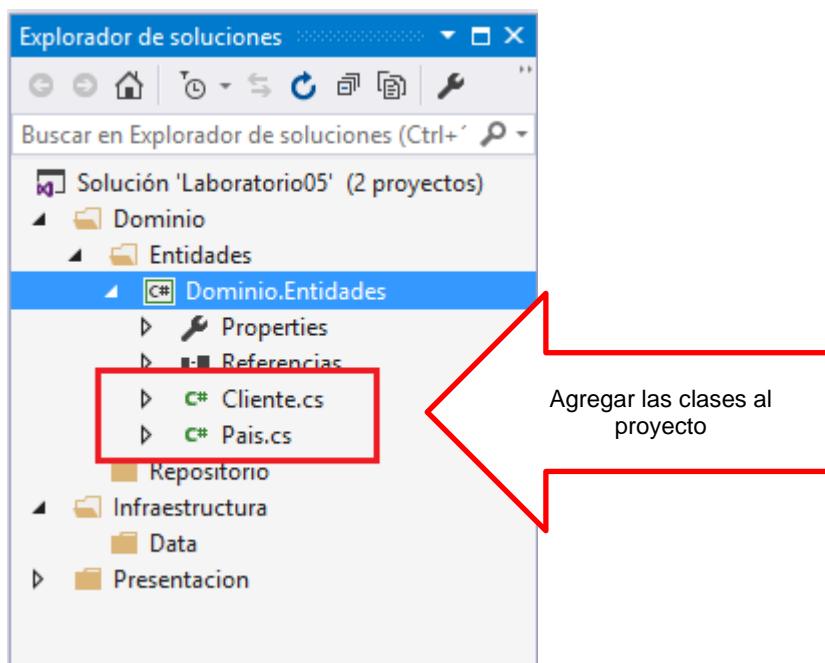
Para agregar la librería de notaciones, agregar una referencia



Selecciona la librería DataAnnotations, para validar los datos. Presionar el botón ACEPTAR



A continuación, agrega, en el proyecto Dominio.Entidades, las clases Cliente y País, tal como se muestra



Abrir el archivo País.cs y definir su estructura de datos, tal como se muestra.

```

Pais.cs* # X
C# Dominio.Entidades          ↗ Dominio.Entidades.Pais      ↗ idpais
[using System;]
[using System.Collections.Generic;]
[using System.Linq;]
[using System.Text;]
[using System.Threading.Tasks;]
[using System.ComponentModel;]

namespace Dominio.Entidades
{
    public class País
    {
        [DisplayName("Codigo")]
        public string idpais { get; set; }

        [DisplayName("País")]
        public string nombrepais { get; set; }
    }
}

```

Abrir el archivo Cliente.cs, importar las librerías de validaciones y notaciones. Defina la estructura de datos, tal como se muestra

```

Cliente.cs* # X
C# Dominio.Entidades          ↗ Dominio.Entidades.Cliente      ↗ idpais
[using System;]
[using System.Collections.Generic;]
[using System.Linq;]
[using System.Text;]
[using System.Threading.Tasks;]
[using System.ComponentModel;]
[using System.ComponentModel.DataAnnotations;]

namespace Dominio.Entidades
{
    public class Cliente
    {
        [DisplayName("Codigo Cliente")]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el codigo")]
        public string idcliente { get; set; }

        [DisplayName("Nombre del Cliente")]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el nombre")]
        public string nombreCia { get; set; }

        public string Direccion { get; set; }

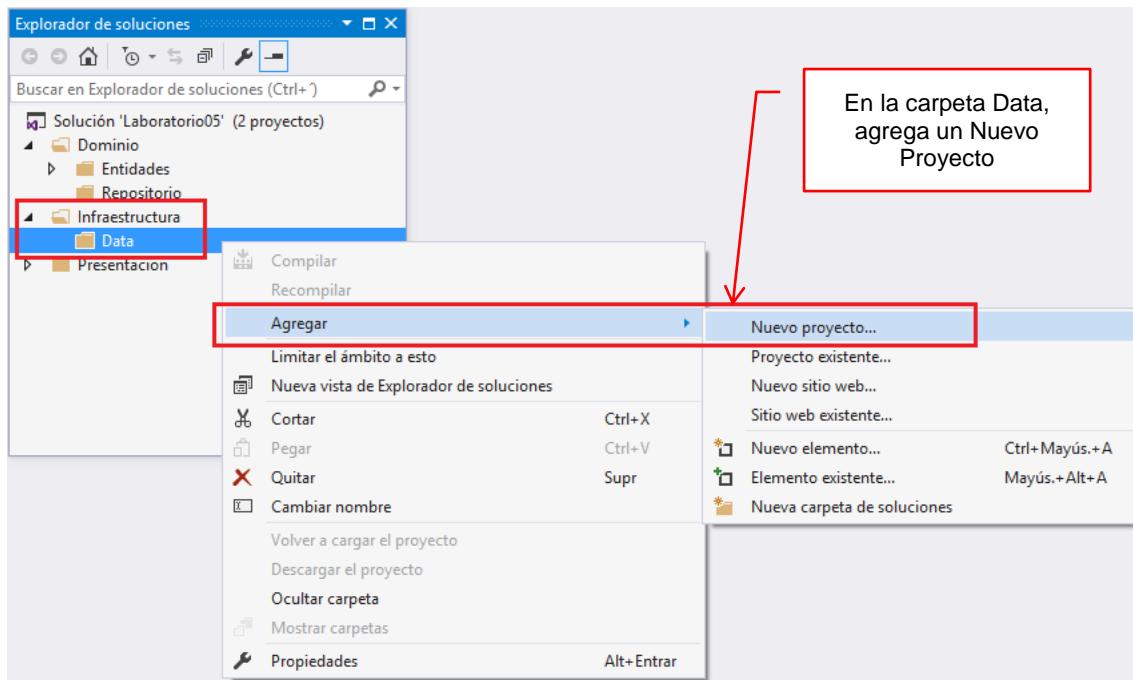
        [DisplayName("País")]
        public string idpais { get; set; }

        public string Telefono { get; set; }
    }
}

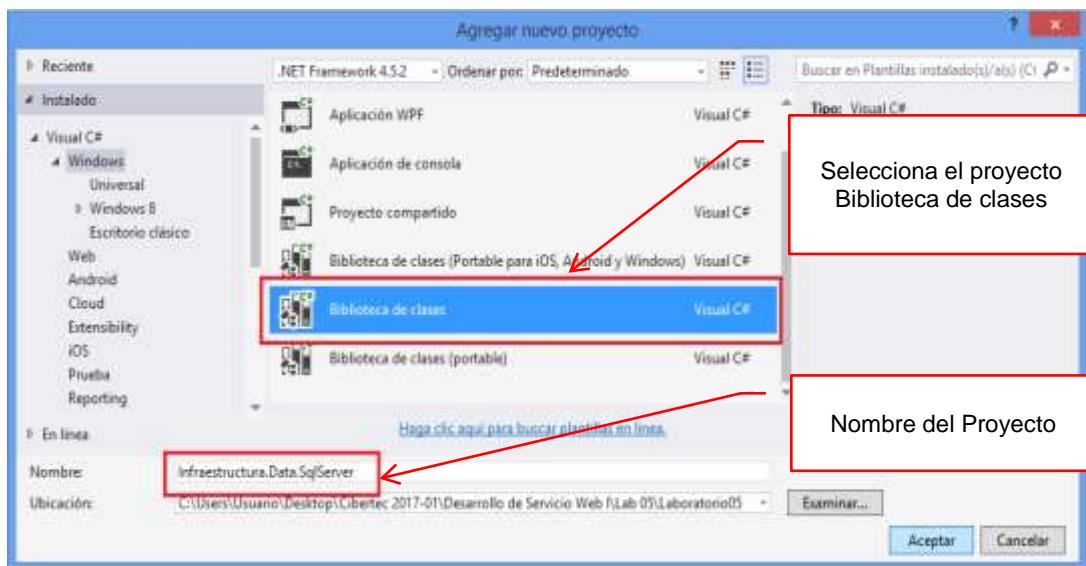
```

## Trabajando con el Acceso de datos

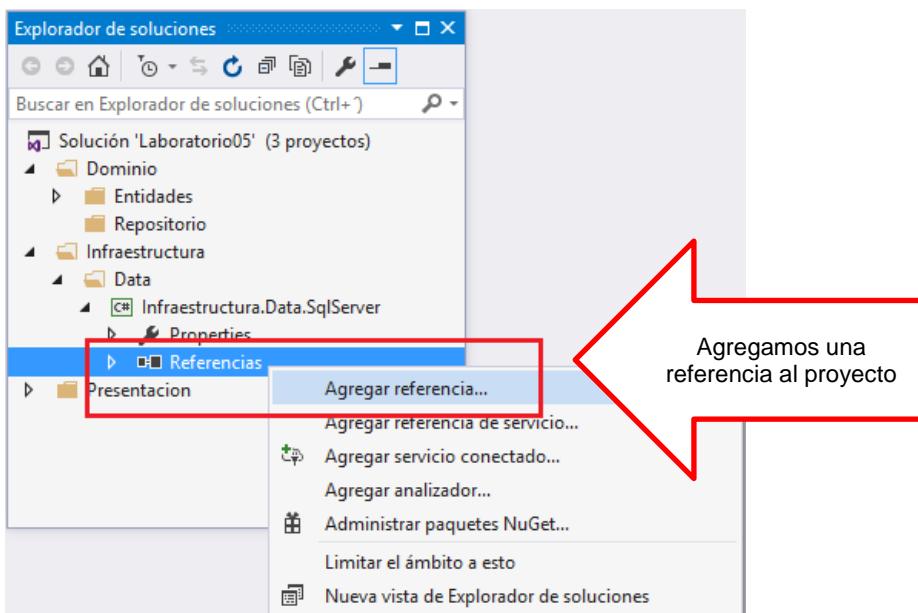
Para el desarrollo de la aplicación, debemos definir el proyecto para Acceso a Datos. Para ello agrega dentro de la carpeta Data un Nuevo Proyecto, tal como se muestra



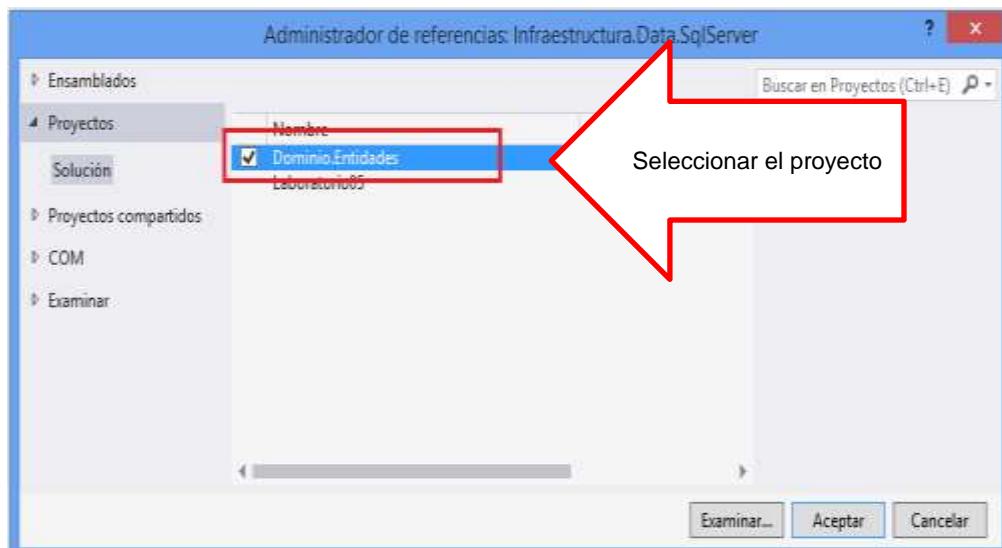
Selecciona el proyecto Biblioteca de Clase y asigne el nombre Infraestructura.Data.SqlServer, tal como se muestra



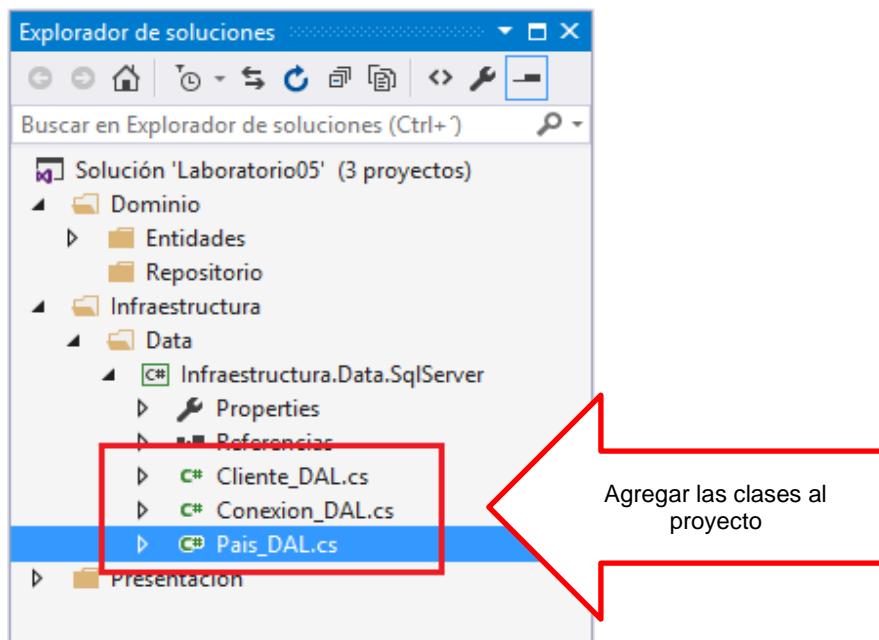
A continuación agregamos la referencia al proyecto



En la opción Proyecto, seleccionamos el proyecto Dominio.Entidades, para referenciar las clases de la librería, tal como se muestra



A continuación, agregamos al proyecto las clases, tal como se muestra.



### Trabajando con la clase Conexión\_DAL

Importar las librerías de trabajo. Luego instanciamos la clase SqlConnection definiendo la conexión a la base de datos.

Finalizando, defina una propiedad de solo lectura: getcn(), la cual retorna la conexión de la base de datos, tal como se muestra.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
namespace Infraestructura.Data.SqlServer
{
    public class Conexion_DAL
    {
        SqlConnection cn = new SqlConnection("server=.; DataBase=Negocios2017; uid=sa; pwd=sql");
        public SqlConnection getcn
        {
            get { return cn; }
        }
    }
}

```

## Trabajando con la clase País\_DAL

Para implementar la clase, primero importamos las librerías. A continuación instanciamos la conexión a la base de datos y definimos el método listado() el cual retorna la lista de los registros de tb\_paises

```

Pais.DAL.cs* 萍 X
[Infraestructura.Data.SqlClient] + Infraestructura.Data.SqlClient.Pais.DAL
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using Dominio.Entidades;

namespace Infraestructura.Data.SqlClient
{
    public class País_DAL
    {
        Conexion_DAL cn = new Conexion_DAL();

        public List<País> listado()
        {
        }
    }
}

```

Implementa el método listado() la cual retorna los registros de tb\_paises.

```

Pais.DAL.cs* 萍 X
[Infraestructura.Data.SqlClient] + Infraestructura.Data.SqlClient.Pais.DAL
using ...
namespace Infraestructura.Data.SqlClient
{
    public class País_DAL
    {
        Conexion_DAL cn = new Conexion_DAL();

        public List<País> listado()
        {
            List<País> lista = new List<País>();

            SqlCommand cmd = new SqlCommand("Select * from tb_paises", cn.getcn);
            cn.getcn.Open();
            SqlDataReader dr = cmd.ExecuteReader();

            while (dr.Read())
            {
                País reg = new País();
                reg.idpais = dr.GetString(0);
                reg.nombrepais = dr.GetString(1);
                lista.Add(reg);
            }
            dr.Close(); cn.getcn.Close();

            return lista;
        }
    }
}

```

## Trabajando con la clase Cliente\_DAL

Para implementar la clase, primero importamos las librerías. A continuación instanciamos la conexión a la base de datos y definimos los métodos para listado y actualización de datos, de los registros de tb\_clientes

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using Dominio.Entidades;

namespace Infraestructura.Data.SqlServer
{
    public class Cliente_DAL
    {
        Conexion_DAL cn = new Conexion_DAL();

        public List<Cliente> listado() { }

        public string Agregar(Cliente reg) { }

        public string Actualizar(Cliente reg) { }

        public string Eliminar(Cliente reg) { }

        public Cliente Registro(string id) { }
    }
}

```

Implementa el método listado(), el cual retorna los registros de tb\_clientes

```

public List<Cliente> listado()
{
    List<Cliente> lista = new List<Cliente>();
    SqlCommand cmd = new SqlCommand("Select * from tb_clientes", cn.getcn);
    cn.getcn.Open();
    SqlDataReader dr = cmd.ExecuteReader();

    while (dr.Read())
    {
        Cliente reg = new Cliente();
        reg.idcliente = dr.GetString(0);
        reg.nombreCia = dr.GetString(1);
        reg.Direccion = dr.GetString(2);
        reg.idpais = dr.GetString(3);
        reg.Telefono= dr.GetString(4);
        lista.Add(reg);
    }
    dr.Close(); cn.getcn.Close();
    return lista;
}

```

Implementa el método Agregar(), el cual ejecuta la comando Insert Into para agregar un registro a la tabla tb\_clientes

```

Cliente_DAL.cs  *  NegociosController.cs
Infraestructura.Data.SqlClient
public class Cliente_DAL {
    Conexion_DAL cn = new Conexion_DAL();
    public List<Cliente> listado() ...
    public string Agregar(Cliente reg) {
        string msg = "";
        cn.getcn.Open();
        try {
            SqlCommand cmd = new SqlCommand(
                "insert tb_clientes Values(@id,@nom,@dir,@pais,@fono)", cn.getcn);
            cmd.Parameters.AddWithValue("@id", reg.idcliente);
            cmd.Parameters.AddWithValue("@nom", reg.nombreCia);
            cmd.Parameters.AddWithValue("@dir", reg.Direccion);
            cmd.Parameters.AddWithValue("@pais", reg.idpais);
            cmd.Parameters.AddWithValue("@fono", reg.Telefono);

            int q = cmd.ExecuteNonQuery();
            msg = q.ToString() + " registro agregado";
        }
        catch (SqlException x) {
            msg = x.Message;
        }
        finally {
            cn.getcn.Close();
        }
        return msg;
    }
    public string Actualizar(Cliente reg) ...
    public string Eliminar(Cliente reg) ...
    public Cliente Registro(string id) ...
}

```

Implementa el método Actualizar(), el cual ejecuta la comando Update para actualizar los datos de un registro de la tabla tb\_clientes por su campo idcliente

```

Cliente_DAL.cs  *  NegociosController.cs
Infraestructura.Data.SqlClient
public class Cliente_DAL {
    Conexion_DAL cn = new Conexion_DAL();
    public List<Cliente> listado() ...
    public string Agregar(Cliente reg) ...
    public string Actualizar(Cliente reg) {
        string msg = "";
        cn.getcn.Open();
        try {
            SqlCommand cmd = new SqlCommand(
                "Update tb_clientes Set nombrecia=@nom, Direccion=@dir,Idpais=@pais,Telefono=@fono " +
                "where idcliente=@id", cn.getcn);
            cmd.Parameters.AddWithValue("@id", reg.idcliente);
            cmd.Parameters.AddWithValue("@nom", reg.nombreCia);
            cmd.Parameters.AddWithValue("@dir", reg.Direccion);
            cmd.Parameters.AddWithValue("@pais", reg.idpais);
            cmd.Parameters.AddWithValue("@fono", reg.Telefono);

            int q = cmd.ExecuteNonQuery();
            msg = q.ToString() + " registro actualizado";
        }
        catch (SqlException x) {
            msg = x.Message;
        }
        finally {
            cn.getcn.Close();
        }
        return msg;
    }
    public string Eliminar(Cliente reg) ...
    public Cliente Registro(string id) ...
}

```

Implementa el método Eliminar(), el cual ejecuta la comando Delete para eliminar un registro de la tabla tb\_clientes por su campo idcliente

```

    Cliente_DAL.cs  NegociosController.cs
    Infraestructura.Data.SqlServer  Infraestructura.Data.SqlServer.Cliente_DAL  cn
    public class Cliente DAL
    {
        Conexion_DAL cn = new Conexion DAL();
        public List<Cliente> listado() ...
        public string Agregar(Cliente reg) ...
        public string Actualizar(Cliente reg) ...
        public string Eliminar(Cliente reg)
        {
            string msg = "";
            cn.getcn.Open();
            try
            {
                SqlCommand cmd = new SqlCommand(
                    "Delete tb_clientes where idcliente=@id", cn.getcn);
                cmd.Parameters.AddWithValue("@id", reg.idcliente);

                int q = cmd.ExecuteNonQuery();
                msg = q.ToString() + " registro actualizado";
            }
            catch (SqlException x)
            {
                msg = x.Message;
            }
            finally
            {
                cn.getcn.Close();
            }
            return msg;
        }
        public Cliente Registro(string id) ...
    }

```

Implementa el método Registro(), el cual retorna un registro de la tabla tb\_clientes por su campo idcliente

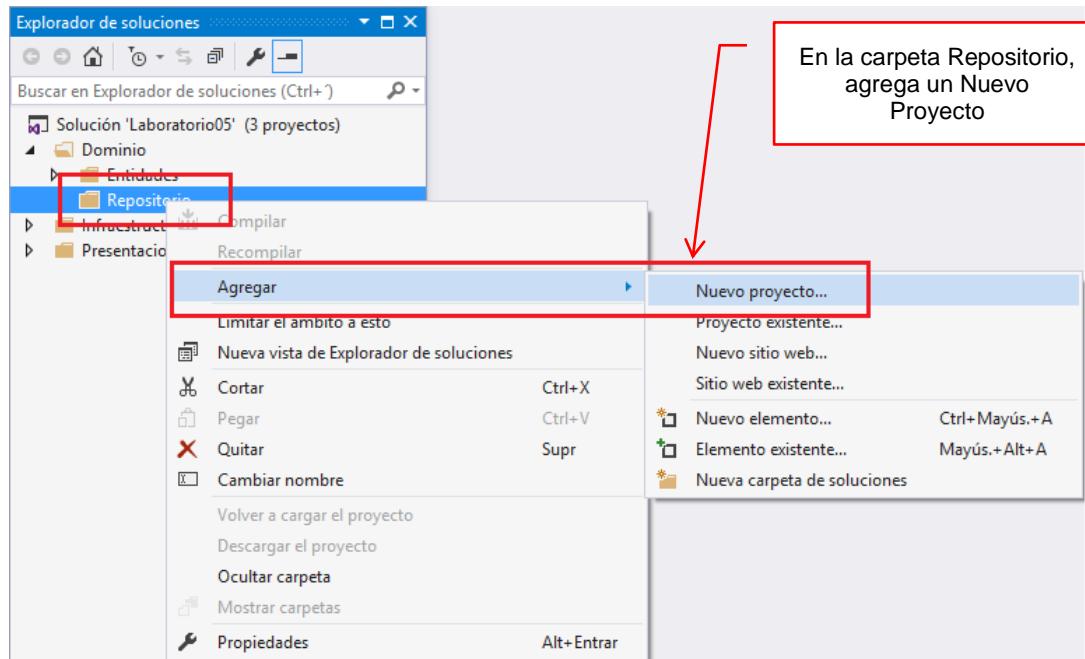
```

    Cliente_DAL.cs  NegociosController.cs
    Infraestructura.Data.SqlServer  Infraestructura.Data.SqlServer.Cliente_DAL  cn
    public class Cliente DAL
    {
        Conexion_DAL cn = new Conexion DAL();
        public List<Cliente> listado() ...
        public string Agregar(Cliente reg) ...
        public string Actualizar(Cliente reg) ...
        public string Eliminar(Cliente reg) ...
        public Cliente Registro(string id)
        {
            return listado().Where(c => c.idcliente == id).FirstOrDefault();
        }
    }

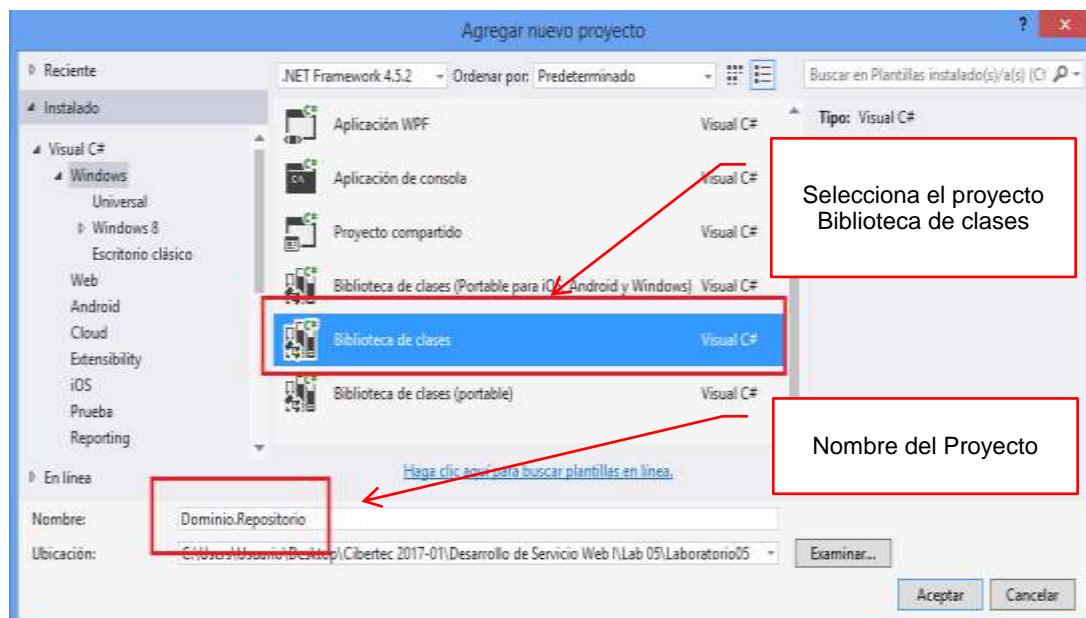
```

## Trabajando con la capa de Repositorio (Negocio)

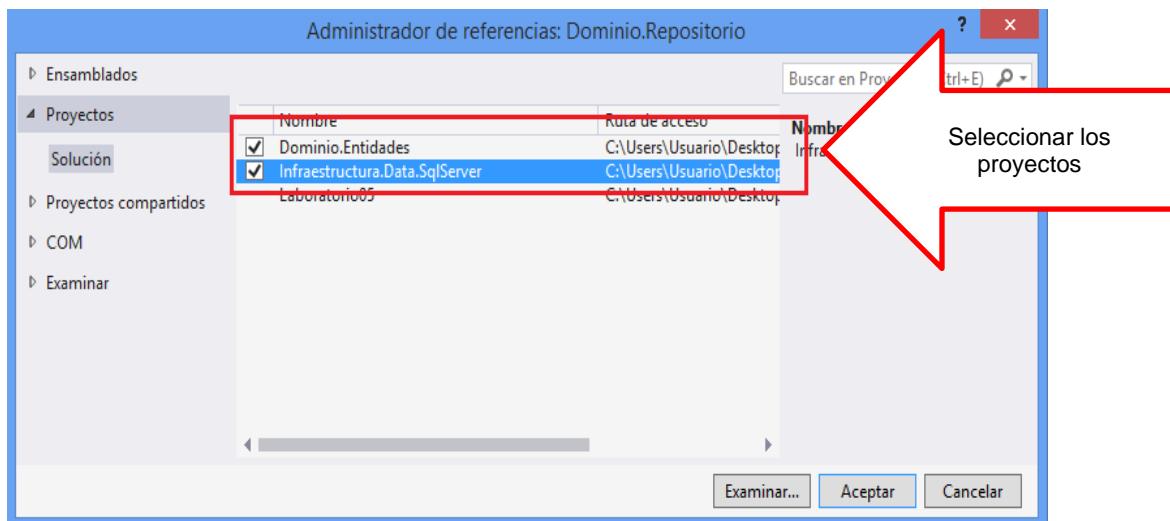
Para el desarrollo de la aplicación, debemos definir el proyecto para Repositorio. Para ello agrega dentro de la carpeta Repositorio un Nuevo Proyecto, tal como se muestra



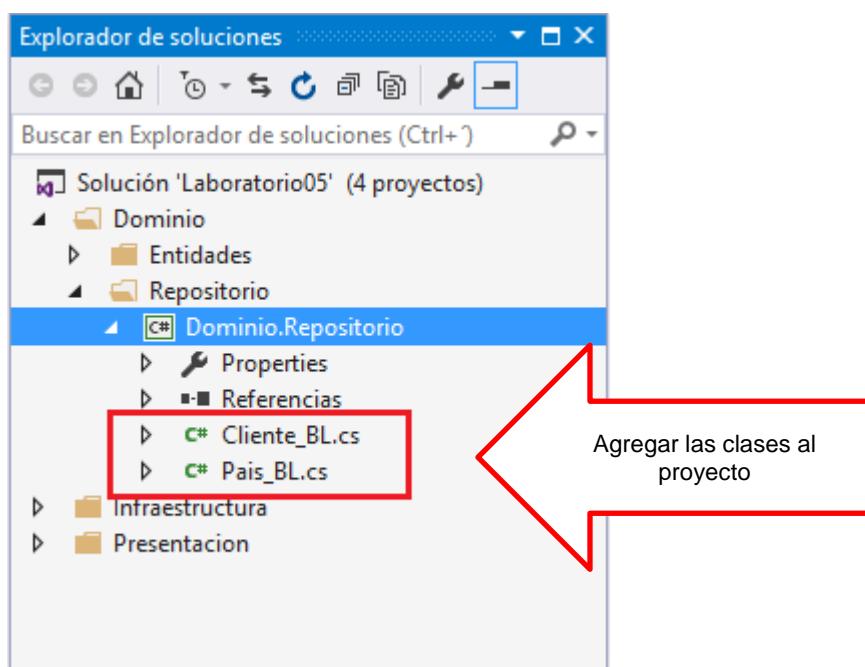
Selecciona el proyecto Biblioteca de Clase y asigne el nombre Dominio.Infraestructura, tal como se muestra



Dentro del proyecto definimos las referencias a los proyectos que utilizamos, tal como se muestra



A continuación, agregamos al proyecto las clases, tal como se muestra



## Trabajando con la clase País\_BL

Para implementar la clase, primero importamos las librerías. A continuación instanciamos la clase País\_DAL y definimos el métodos para listado de tb\_paises

```

Pais_BL.cs  + X
C# Dominio.Repositorio  Dominio.Repositorio.Pais_BL  listado()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Dominio.Entidades;
using Infraestructura.Data.SqlServer;

namespace Dominio.Repositorio
{
    public class País_BL
    {
        País_DAL país = new País_DAL();

        public List<País> listado()
        {
            return país.listado();
        }
    }
}

```

Importar las librerías

Instancia de País\_DAL

Definir e implementa el método listado()

## Trabajando con la clase Cliente\_BL

Para implementar la clase, primero importamos las librerías. A continuación instanciamos la clase País\_DAL y definimos el métodos para listado de tb

```

Cliente_BL.cs  + X
C# Dominio.Repositorio  Dominio.Repositorio.Cliente_BL  clientes
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Dominio.Entidades;
using Infraestructura.Data.SqlServer;
namespace Dominio.Repositorio
{
    public class Cliente_BL
    {
        Cliente_DAL cliente = new Cliente_DAL();

        public List<Cliente> listado(){
            return cliente.listado();
        }

        public string Agregar(Cliente reg){
            return cliente.Agregar(reg);
        }

        public string Actualizar(Cliente reg){
            return cliente.Actualizar(reg);
        }

        public string Eliminar(Cliente reg){
            return cliente.Eliminar(reg);
        }

        public Cliente Registro(string id){
            return cliente.Registro(id);
        }
    }
}

```

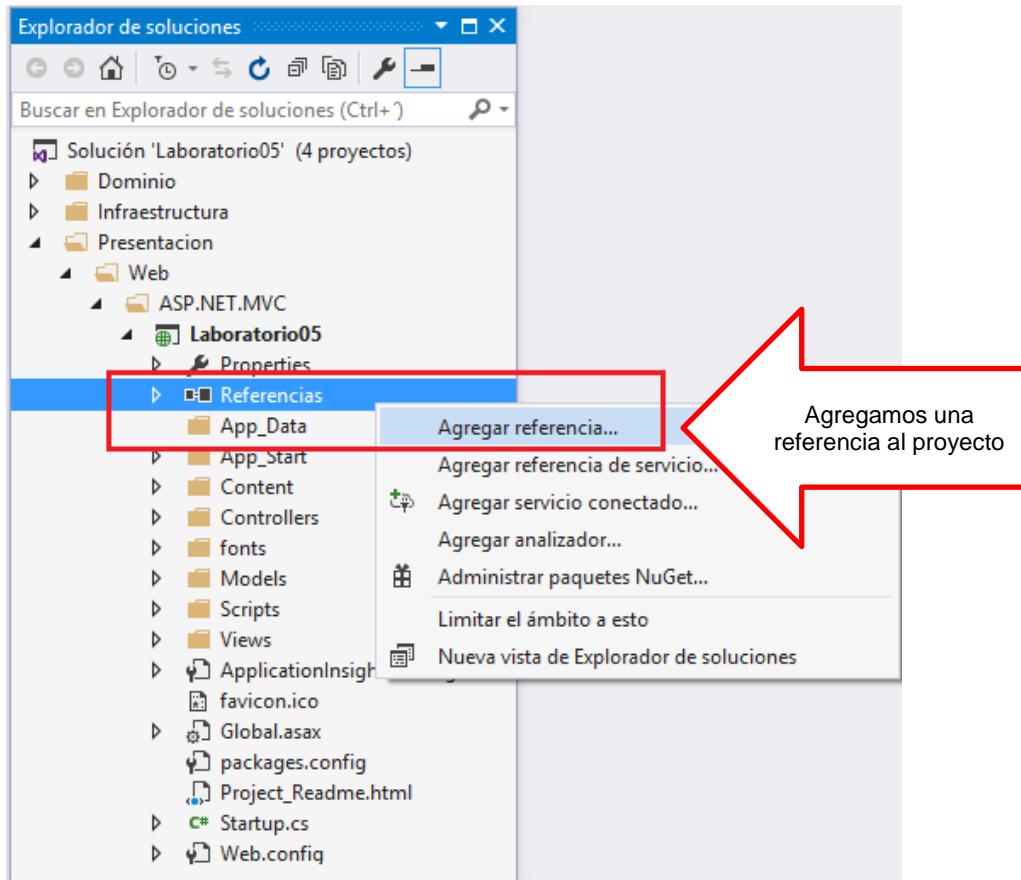
Importar las librerías

Instancia de Cliente\_DAL

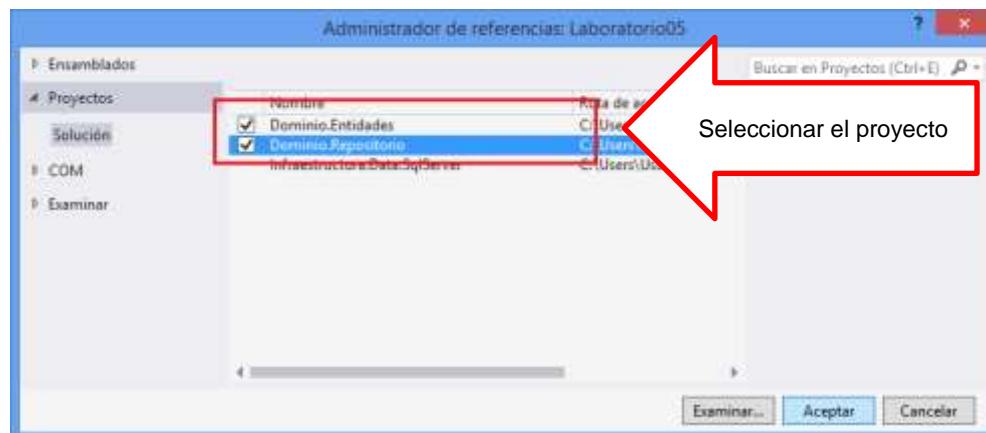
Definir e implementa los métodos de la clase Cliente\_DAL

## Trabajando con el proyecto ASP.NET MVC

En el proyecto Web, agregamos las referencias de las librerías, tal como se muestra

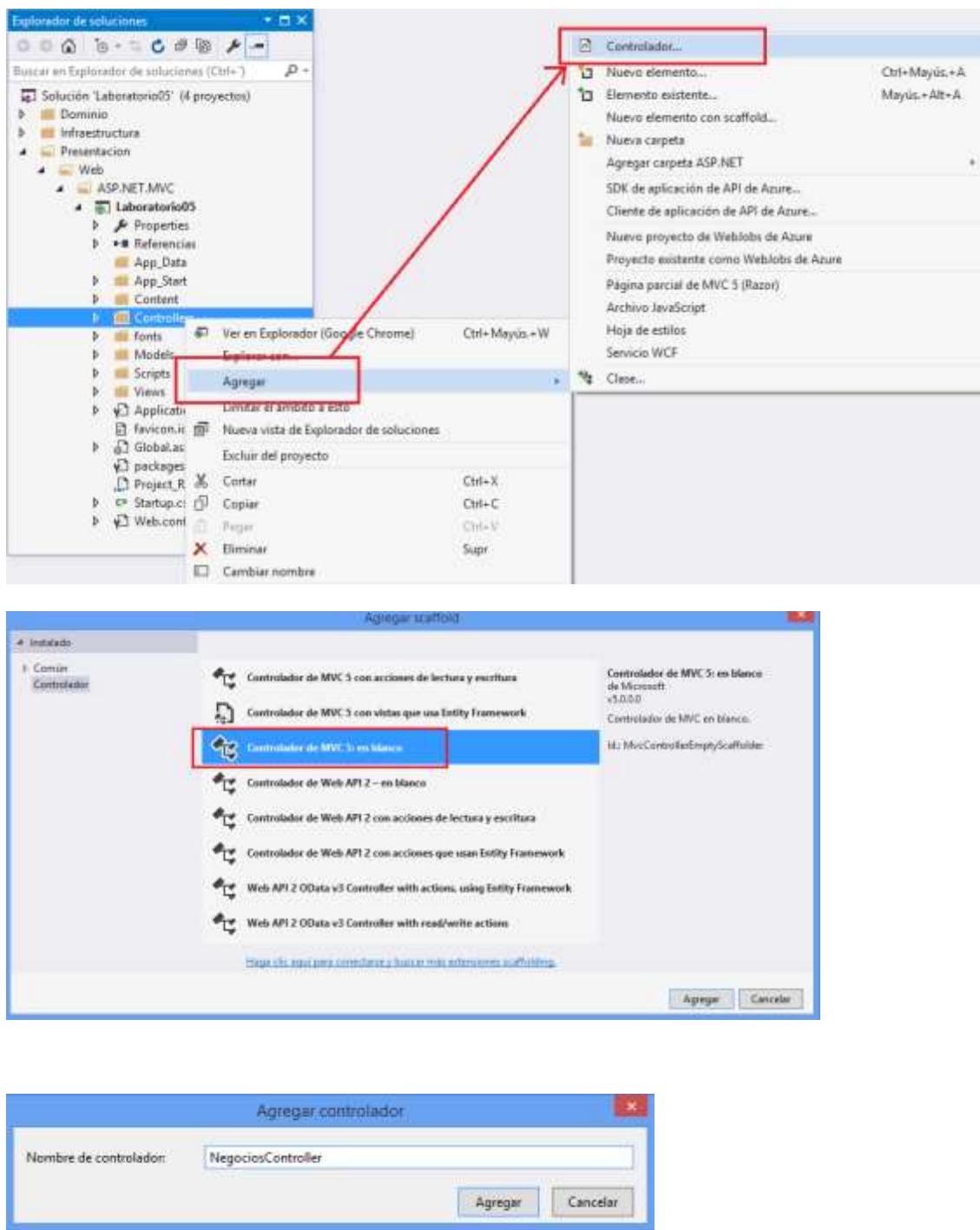


Selecciona los proyecto Dominio.Entidades y Dominio.Repositorio, presiona el botón ACEPTAR



## Agregando el Controlador

En el proyecto Web, agrega un controlador llamado NegociosController, tal como se muestra en la figura



En el controlador, importar las librerías del Dominio, instanciar las clases del Repositorio, tal como se muestra.

```

NegociosController.cs  X
[>#] Laboratorio05          Laboratorio05.Controllers.NegociosController    pais
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Web;
    using System.Web.Mvc;
    using Dominio.Entidades;
    using Dominio.Repositorio;

    namespace Laboratorio05.Controllers
    {
        public class NegociosController : Controller
        {
            Pais_BL pais = new Pais_BL();
            Cliente_BL cliente = new Cliente_BL();
        }
    }

```

Importar las librerías

Instancia las clases del Repositorio

### ActionResult Listado

Defina el ActionResult Listado(), el cual retorna los registro de clientes, tal como se muestra

```

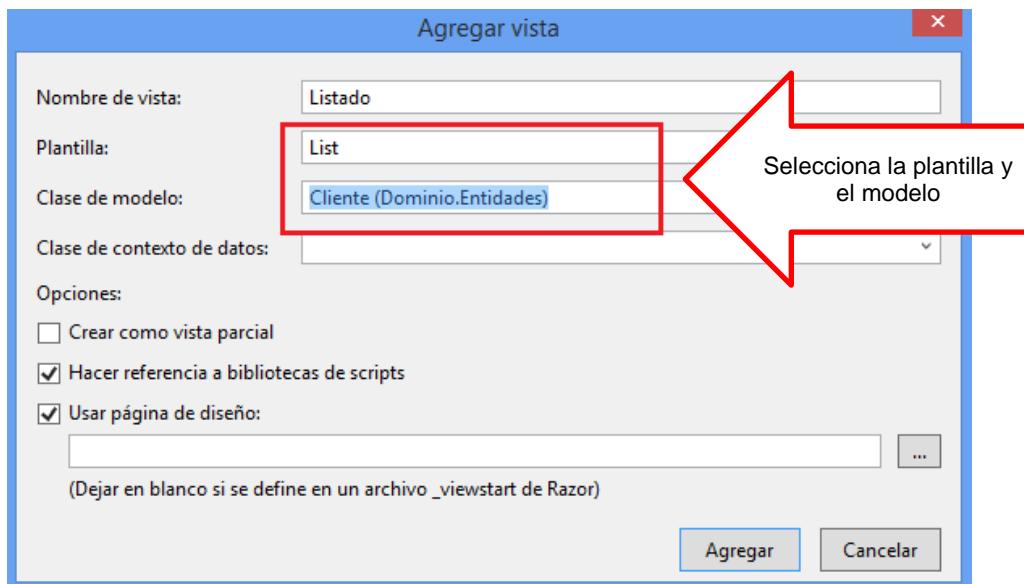
NegociosController.cs  X
[>#] Laboratorio05          Laboratorio05.Controllers.NegociosController    pais
    using ...
    namespace Laboratorio05.Controllers
    {
        public class NegociosController : Controller
        {
            Pais_BL pais = new Pais_BL();
            Cliente_BL cliente = new Cliente_BL();

            public ActionResult Listado()
            {
                return View(cliente.listado());
            }
        }
    }

```

Defina el método donde retorna la lista de Clientes

A continuación agregamos una vista cuya plantilla será tipo List y la clase de modelo es Cliente de Dominio.Entidades, tal como se visualiza



Modificar el diseño de la página, tal como se muestra

```

Listado.cshtml.cs NegocioController.cs
@model IEnumerable<Dominio.Entidades.Cliente>
@{ ViewBag.Title = "Listado";}
<h2>Listado</h2>

<p>
    @Html.ActionLink("Nuevo Cliente", "Create")
</p>
<table class="table">
    <tr>
        <th>@Html.DisplayNameFor(model => model.idcliente)</th>
        <th>@Html.DisplayNameFor(model => model.nombreCia)</th>
        <th>@Html.DisplayNameFor(model => model.Direccion)</th>
        <th>@Html.DisplayNameFor(model => model.idpais)</th>
        <th>@Html.DisplayNameFor(model => model.Telefono)</th>
        <th></th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.idcliente)</td>
            <td>@Html.DisplayFor(modelItem => item.nombreCia)</td>
            <td>@Html.DisplayFor(modelItem => item.Direccion)</td>
            <td>@Html.DisplayFor(modelItem => item.idpais)</td>
            <td>@Html.DisplayFor(modelItem => item.Telefono)</td>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id=item.idcliente }) |
                @Html.ActionLink("Details", "Details", new { id=item.idcliente }) |
                @Html.ActionLink("Delete", "Delete", new { id=item.idcliente })
            </td>
        </tr>
    }
</table>

```

Ejecuta el proyecto, donde se visualiza la lista de los registros de Clientes

Código Cliente	Nombre del Cliente	Dirección	País	Teléfono	
AA555	Aaron Alvarez Gatica	Av. Lima 46533	005	8556633	Edit   Details   Delete
AB123	Abraham Gatica	Jr. Cucus 1222	001	5663232	Edit   Details   Delete
AL101	Alfredo Pineda	Oeste 87	002	038-0974321	Edit   Details   Delete
ANATR	Ana Trujillo Emparedados y helados	Avenida de la Constitución 2222	005	(5) 555-4739	Edit   Details   Delete
ANTON	Antonio Moreno Tiqueros	Mataderos 2312	007	(5) 555-3832	Edit   Details   Delete
ARDOT	Armando the Hare	129 Hareover Sq.	004	(71) 555-7788	Edit   Details   Delete
BERGS	Berglunds snabbköp	Berguvägen 8	006	0621-12 34 65	Edit   Details   Delete
BLAUS	Blauer See Delikatesse	Fosseleit 57	001	0621-08400	Edit   Details   Delete
BLONP	Blondel père et fils	24, place Kleber Estambulga	008	88-60 15 31	Edit   Details   Delete
BOCID	Bolido Comidas preparadas	C/ Arzaké, 67	009	(91) 555 81 99	Edit   Details   Delete
BONAP	Bon app	12, rue des Beaux Arts	001	91-24 45 41	Edit   Details   Delete
BOTTM	Bottino-Dollar Markets	23 Tavassan Blvd.	003	(604) 565-3745	Edit   Details   Delete

## ActionResult Create

Defina el ActionResult Create(), el cual agregar un registro de clientes, tal como se muestra

```

public class NegociosController : Controller
{
    País_BL pais = new País_BL();
    Cliente_BL cliente = new Cliente_BL();

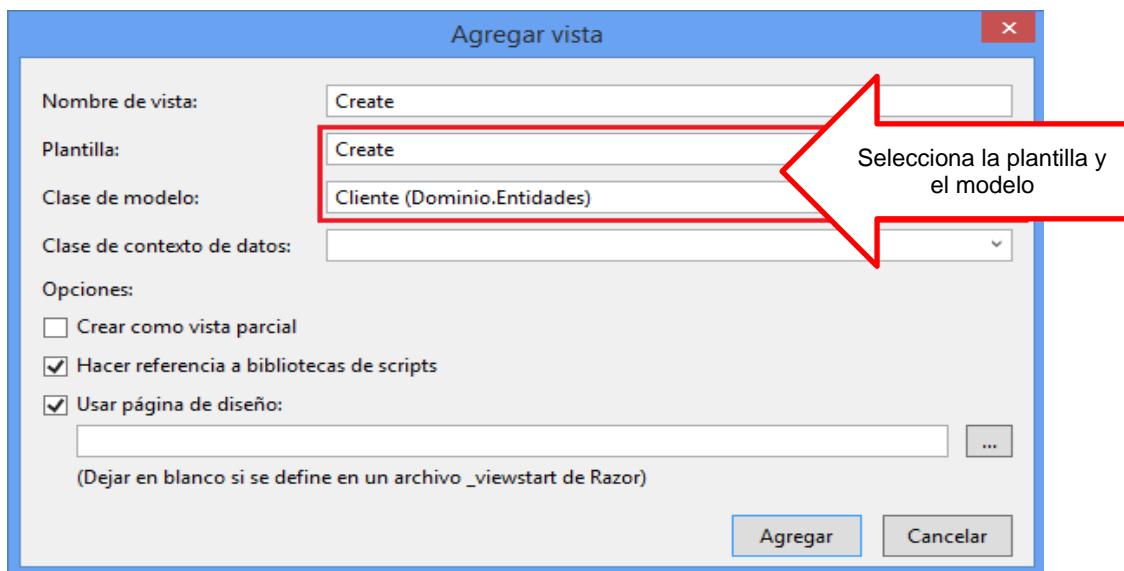
    public ActionResult Listado()
    {
        ViewBag.paises = new SelectList(pais.listado(), "idpais", "nombrepais");
        return View(new Cliente());
    }
    [HttpPost]
    public ActionResult Create(Cliente reg)
    {
        if (!ModelState.IsValid)
        {
            ViewBag.paises = new SelectList(pais.listado(), "idpais", "nombrepais", reg.idpais);
            return View(reg);
        }
        ViewBag.mensaje = cliente.Agregar(reg);
        return RedirectToAction("Listado");
    }
}

```

Método Get, retorna un registro en blanco

Método Post, recibe el registro para ser agregado

A continuación agregamos una vista cuya plantilla será tipo Create y la clase de modelo es Cliente de Dominio.Entidades, tal como se visualiza



Modifica la estructura del campo idpais, reemplazandola por un DropDownList, para seleccionar un pais

```

Create.cshtml
@model Dominio.Entidades.Cliente
@{
    ViewBag.Title = "Create";
}



## Create


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Cliente</h4>
        <br />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">...</div>
        <div class="form-group">...</div>
        <div class="form-group">...</div>

        <div class="form-group">
            @Html.Label("Pais", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownList("idpais", (SelectList)ViewBag.paises, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>

        <div class="form-group">...</div>
        <div class="form-group">...</div>
    </div>
    <div>@Html.ActionLink("Back to List", "list")</div>
}

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

```

Ejecuta la página, ingresa los datos, al presionar el botón Create, se agrega el registro

The screenshot shows a web application window titled 'Create - Mi aplicación ASP.NET'. The URL in the address bar is 'localhost:63877/Negocios/Create'. The page has a header with 'Nombre de aplicación', 'Inicio', 'Acerca de', 'Contacto', 'Registrarse', and 'Iniciar sesión'. Below the header, the word 'Create' is displayed above a form titled 'Cliente'. The form contains the following fields:

- Código Cliente: A0003
- Nombre del Cliente: Juan Garcia
- Dirección: Lima
- País: Perú
- Teléfono: 98623241

At the bottom of the form is a 'Create' button. Below the form, there is a link 'Back to List' and a copyright notice '© 2017 - Mi aplicación ASP.NET'.

### ActionResult Edit

Defina el ActionResult Edit(), el cual modifica los datos de un registro de clientes, tal como se muestra.

```

    NegociosController.cs  Laboratorio05  Laboratorio05.Controllers.NegociosController  País
    public class NegociosController : Controller
    {
        País_BL pais = new País_BL();
        Cliente_BL cliente = new Cliente_BL();

        public ActionResult Listado()...
        public ActionResult Create()...
        [HttpPost]
        public ActionResult Create(Cliente reg)...

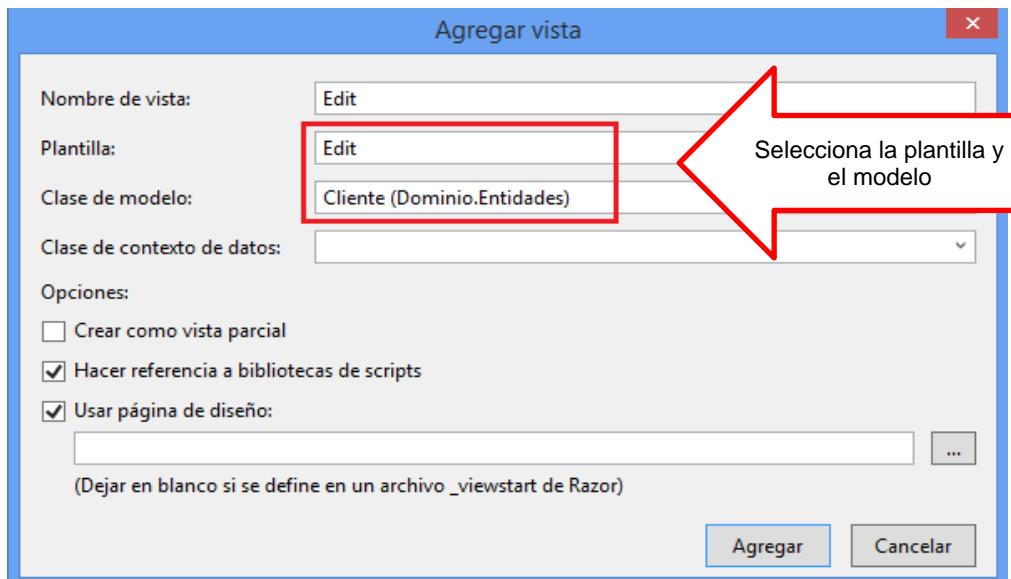
        public ActionResult Edit(string id)
        {
            Cliente reg = cliente.Registro(id);
            ViewBag.paises = new SelectList(pais.listado(), "idpais", "nombrepais", reg.idpais);
            return View(reg);
        }
        [HttpPost]
        public ActionResult Edit(Cliente reg)
        {
            if (!ModelState.IsValid)
            {
                ViewBag.paises = new SelectList(pais.listado(), "idpais", "nombrepais", reg.idpais);
                return View(reg);
            }
            ViewBag.mensaje = cliente.Actualizar(reg);
            return RedirectToAction("Listado");
        }
    }

```

Metodo Get, retorna un registro de tb\_clientes

Metodo Post, recibe el registro para ser actualizado

A continuación agregamos una vista cuya plantilla será tipo Edit y la clase de modelo es Cliente de Dominio.Entidades, tal como se visualiza



Modifica la estructura del campo idpais, reemplazandola por un DropDonwList, para seleccionar un pais

```

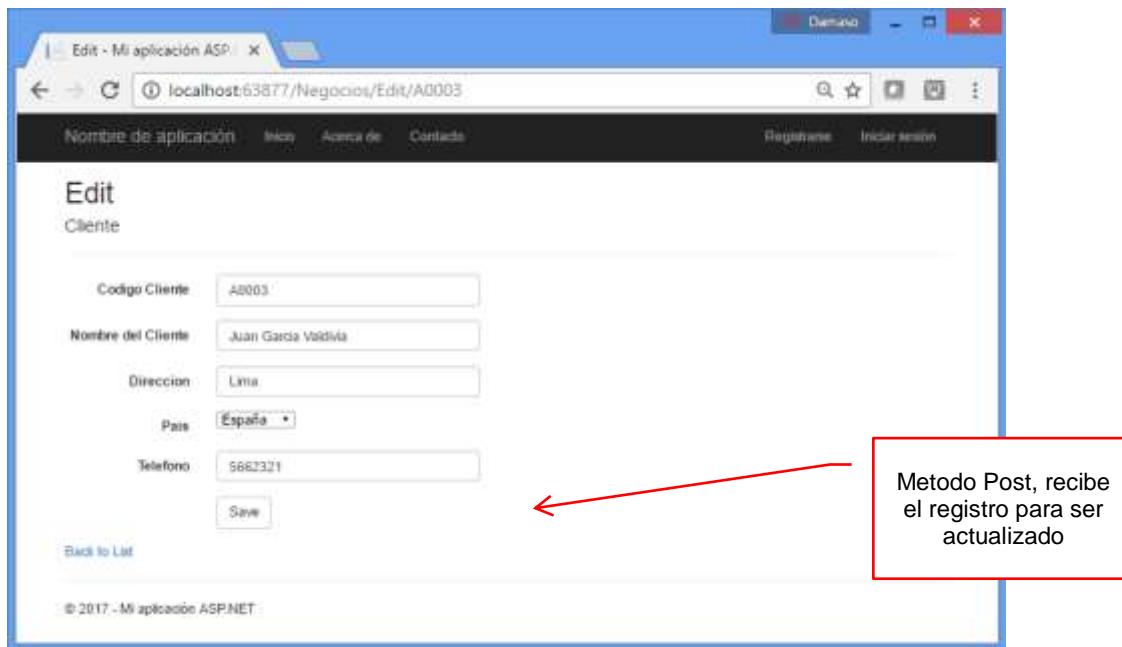
@model Dominio.Entidades.Cliente
@{
    ViewBag.Title = "Edit";
}


## Edit


using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Cliente</h4>
        <br />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">...</div>
        <div class="form-group">...</div>
        <div class="form-group">...</div>
        <div class="form-group">
            @Html.Label("País", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownList("idpais", ViewBag.paises, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>
        <div class="form-group">...</div>
        <div class="form-group">...</div>
    </div>
    <div>
        <div>@Html.ActionLink("Back to list", "Index")</div>
        @section Scripts { @Scripts.Render("~/bundles/jqueryval") }
    </div>
}

```

Ejecuta la página Listado, selecciona un cliente desde el link Edit, modifica los datos, al presionar el botón Edit, se actualiza los datos, visualizando los cambios en la página listado().



## ActionResult Details

Defina el ActionResult Details, el cual retorna el registro de tb\_clientes por su id

```

namespace Laboratorio05.Controllers
{
    public class NegociosController : Controller
    {
        Pais_BL pais = new Pais_BL();
        Cliente_BL cliente = new Cliente_BL();

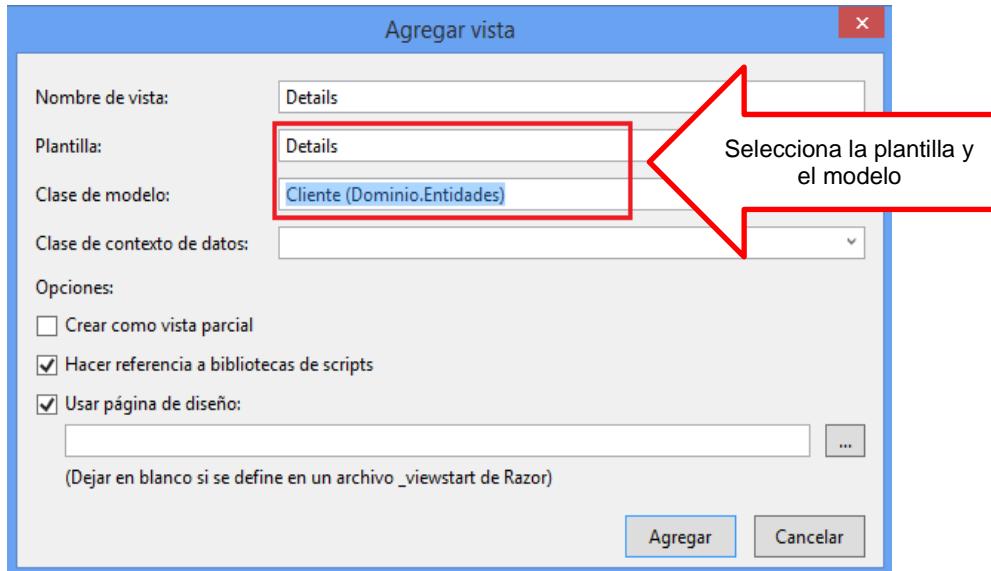
        public ActionResult Listado()...
        public ActionResult Create()...
        [HttpPost]
        public ActionResult Create(Cliente reg)...

        public ActionResult Edit(string id)...
        [HttpPost]
        public ActionResult Edit(Cliente reg)...

        public ActionResult Details(string id)
        {
            return View(cliente.Registro(id));
        }
    }
}

```

A continuación agregamos una vista cuya plantilla será tipo Details y la clase de modelo es Cliente de Dominio.Entidades, tal como se visualiza



Modifica los link de la página, tal como se muestra.

```

@model Dominio.Entidades.Cliente
@{
    ViewBag.Title = "Details";
}



## Details



#### Cliente



---



@Html.DisplayNameFor(model => model.idcliente)
:   @Html.DisplayFor(model => model.idcliente)


@Html.DisplayNameFor(model => model.nombreCia)
:   @Html.DisplayFor(model => model.nombreCia)


@Html.DisplayNameFor(model => model.Direccion)
:   @Html.DisplayFor(model => model.Direccion)


@Html.DisplayNameFor(model => model.idpais)
:   @Html.DisplayFor(model => model.idpais)

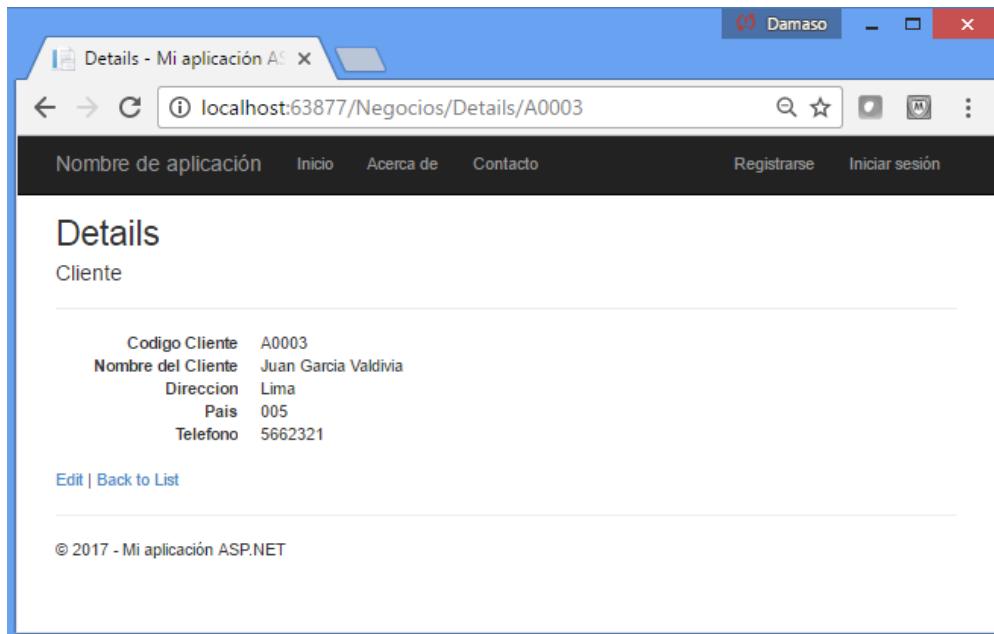

@Html.DisplayNameFor(model => model.Telefono)
:   @Html.DisplayFor(model => model.Telefono)



Edit |
        Back to List


```

Ejecuta la página Listado, selecciona un cliente desde el link Details, para visualizar los datos del cliente seleccionado, al presionar el botón Edit, visualiza los datos para ser editados.



## ActionResult Delete

Defina el ActionResult Delete, el cual elimina el registro de tb\_clientes por su id

```

namespace Laboratorio05.Controllers
{
    public class NegociosController : Controller
    {
        País_BL país = new País_BL();
        Cliente_BL cliente = new Cliente_BL();

        public ActionResult Listado()
        {
            ...
        }

        public ActionResult Create()
        {
            ...
        }

        public ActionResult Create(Cliente reg)
        {
            ...
        }

        public ActionResult Edit(string id)
        {
            ...
        }

        public ActionResult Edit(Cliente reg)
        {
            ...
        }

        public ActionResult Details(string id)
        {
            ...
        }

        public ActionResult Delete(string id)
        {
            Cliente reg = cliente.Registro(id);
            string mensaje = cliente.Eliminar(reg);

            return RedirectToAction("listado");
        }
    }
}

```

## Resumen

- └ El dominio es modelo de una aplicación y su comportamiento, donde se definen los procesos y la reglas de negocio al que esta enfocada la aplicación, es la funcionalidad que se puede hablar con un experto del negocio o analista funcional, esta lógica no depende de ninguna tecnología como por ejemplo si los datos se persisten en una base de datos, si esta base de datos es SQL Server, Neo4j, MongoDB o la que sea y lo que es más importante, esta lógica no debe ser reescrita o modificada porque se cambie una tecnología específica en una aplicación.
- └ DDD es una aproximación concreta para diseñar software basado en la importancia del Dominio del Negocio, sus elementos y comportamientos y las relaciones entre ellos. Está muy indicado para diseñar e implementar aplicaciones empresariales complejas donde es fundamental definir un Modelo de Dominio expresado en el propio lenguaje de los expertos del Dominio de negocio real (el llamado Lenguaje Ubicuo).
- └ Esta arquitectura permite estructurar de una forma limpia y clara la complejidad de una aplicación empresarial basada en las diferentes capas de la arquitectura, siguiendo el patrón N-Layered y las tendencias de arquitecturas en DDD.
- └ Al utilizar ASP.NET MVC se aprovecha todas las ventajas que ofrece para la capa de presentación (HTML 5 + Razor), además de Entity Framework 4.5 Code First, data scaffolding y todos los beneficios que ofrece la aplicación de un patrón de diseño.
- └ Si desea revisar el tema, te publicamos los siguientes enlaces:
  - └ <http://nfanjul.blogspot.pe/2014/09/arquitectura-ddd-y-sus-capas.html>
  - └ <https://pablov.wordpress.com/2010/11/12/arquitectura-de-n-capas-orientada-al-dominio-con-net-4/>
  - └ <http://xurxodeveloper.blogspot.pe/2014/01/ddd-la-logica-de-dominio-es-el-corazon.html>
  - └ <http://www.eltavo.net/2014/08/patrones-implementando-patron.html>
  - └ <http://es.slideshare.net/CesarGomez47/orientaci-19871353>
  - └ <http://elblogdelfrasco.blogspot.pe/2008/10/el-dominio-es-lo-nico-importante.html>





# IMPLEMENTANDO UNA APLICACIÓN E-COMMERCE

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla aplicaciones e-commerce teniendo en cuenta las funciones de compra, validación de usuario, checkout, registro y pago.

## TEMARIO

### Tema 6: Implementando una aplicación e-commerce (06 horas)

- 6.1 Introducción
- 6.2 Proceso del e-commerce
- 6.3 Implementando el proceso del e-commerce (carrito de compras)

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto e-commerce utilizando el patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



## 6. Implementando una aplicación e-commerce “N”

### 6.1 Introducción

El comercio electrónico, o E-commerce, como es conocido en gran cantidad de portales existentes en la web, es definido por el Centro Global de Mercado Electrónico como “cualquier forma de transacción o intercambio de información con fines comerciales en la que las partes interactúan utilizando Tecnologías de la Información y Comunicaciones (TIC), en lugar de hacerlo por intercambio o contacto físico directo”.



*Figura 1: e-commerce*  
<http://beople.es/las-10-tendencias-del-e-commerce/>

Al hablar de E-commerce es requisito indispensable referirse a la tecnología como método y fin de comercialización, puesto que esta es la forma como se imponen las actividades empresariales. El uso de las TIC para promover la comercialización de bienes y servicios dentro de un mercado, conlleva al mejoramiento constante de los procesos de abastecimiento y lleva el mercado local a un enfoque global, permitiendo que las empresas puedan ser eficientes y flexibles en sus operaciones.

La actividad comercial en Internet o comercio electrónico, no difiere mucho de la actividad comercial en otros medios, el comercio real, y se basa en los mismos principios: una oferta, una logística y unos sistemas de pago.



*Figura 2: definición de e-commerce*  
<http://es.slideshare.net/cmcordon/e-commercekcn-alicante169b>

Podría pensarse que tener unas páginas web y una pasarela de pagos podría ser suficiente, pero no es así. Cualquier acción de comercio electrónico debe estar guiada por criterios de rentabilidad o, al menos, de inversión, y por tanto deben destinarse los recursos necesarios para el cumplimiento de los objetivos. Porque si bien se suele decir que poner una tienda virtual en Internet es más barato que hacerlo en el mundo real, las diferencias de coste no son tantas como parece, si se pretende hacerlo bien, claro.

Los objetivos básicos de cualquier sitio web comercial son tres:

- atraer visitantes,

- fidelizarlos y, en consecuencia,
- venderles nuestros productos o servicios

Para poder obtener un beneficio con el que rentabilizar las inversiones realizadas que son las que permiten la realización de los dos primeros objetivos. El equilibrio en la aplicación de los recursos necesarios para el cumplimiento de estos objetivos permitirá traspasar el umbral de rentabilidad y convertir la presencia en Internet en un auténtico negocio.

## 6.2 Proceso del e-commerce

En el comercio electrónico intervienen, al menos, cuatro agentes:

- El proveedor, que ofrece sus productos o servicios a través de Internet.
- El cliente, que adquiere a través de Internet los productos o servicios ofertados por el proveedor.
- El gestor de medios de pago, que establece los mecanismos para que el proveedor reciba el dinero que paga el cliente a cambio de los productos o servicios del proveedor.
- La entidad de certificación, que garantiza mediante un certificado electrónico que los agentes que intervienen en el proceso de la transacción electrónica son quienes dicen ser.

Además de estos agentes suelen intervenir otros que están más relacionados con el suministro de tecnología en Internet (proveedores de hospedaje, diseñadores de páginas web, etc.) que con el propio comercio electrónico.

Básicamente un sistema de comercio electrónico está constituido por unas páginas web que ofrecen un catálogo de productos o servicios. Cuando el cliente localiza un producto que le interesa rellena un formulario con sus datos, los del producto seleccionado y los correspondientes al medio de pago elegido. Al activar el formulario, si el medio de pago elegido ha sido una tarjeta de crédito, se activa la llamada "pasarela de pagos" o TPV (terminal punto de venta) virtual, que no es más que un software desarrollado por entidades financieras que permite la aceptación de pagos por Internet. En ese momento se genera una comunicación que realiza los siguientes pasos: el banco del cliente acepta (o rechaza) la operación, el proveedor y el cliente son informados de este hecho, y a través de las redes bancarias, el dinero del pago es transferido desde la cuenta del cliente a la del proveedor. A partir de ese momento, el proveedor enviará al cliente el artículo que ha comprado.



*Figura 3: proceso de e-commerce*  
<http://www.brainsins.com/es/blog/dropshipping-for-dummies/5916>

Todas estas operaciones se suelen realizar bajo lo que se denomina "servidor seguro", que no es otra cosa que un ordenador verificado por una entidad de certificación y que utiliza un protocolo especial denominado SSL (Secure Sockets Layer), garantizando la

confidencialidad de los datos, o más recientemente con el protocolo SET (Secure Electronic Transaction).

### El carrito de compras en el e-commerce

La aparición y consolidación de las plataformas de e-commerce ha provocado que, en el tramo final del ciclo de compra (en el momento de la transacción), el comprador potencial no interactúe con ninguna persona, sino con un canal web habilitado por la empresa, con el llamado **carrito de compra**.



*Figura 4: carrito de compras*  
<http://www.clarika.com.ar/es/Ecommerce.aspx>

Los carritos de compra son aplicaciones dinámicas que están destinadas a la venta por internet y que si están confeccionadas a medida pueden integrarse fácilmente dentro de websites o portales existentes, donde el cliente busca comodidad para elegir productos (libros, música, videos, comestibles, indumentaria, artículos para el hogar, electrodomésticos, muebles, juguetes, productos industriales, software, hardware, y un largo etc.) -o servicios- de acuerdo a sus características y precios, y simplicidad para comprar.

El desarrollo y programación de un carrito de compras puede realizarse a medida según requerimientos específicos (estos carritos son más fáciles de integrar visualmente a un sitio de internet).

Por otro lado, dentro de las opciones existentes también están los carritos de compra enlatados (en este caso se debe estar seguro de que sus características son compatibles con los requerimientos); open source (código abierto) como osCommerce o una amplia variedad de carritos de compras pagos.

### 6.3 Implementando el proceso del e-commerce



*Figura 5: Implementando el carrito de compras*  
<https://codigofuentenet.wordpress.com/2013/02/05/carrito-de-compras-asp-net-y-c/>

1. Primero a diferencia del carrito del supermercado no es necesario que se tome (crearlo) al principio; no mas llegue a nuestro sitio el cliente. Sino que el cliente puede revisar el catálogo de productos y hasta que este listo a comprar se le asignara un carrito donde introducir su compra.

2. Segundo en el carro de supermercado es capaz de contener una gran cantidad de productos a la vez, nuestro carro de compras debe ser capaz de hacer lo mismo.
3. Tercero el carro de supermercado me permite introducir de un productos varios del mismo, el que programaremos debe ser capaz de hacerlo.
4. Cuarto cuando llega a pagar en el supermercado totaliza su compra a partir de los subtotales de todas sus productos esto lo hace mentalmente, de esta forma decide si dejar algo por que no le alcanza el dinero. Nuestro carro de compras debe ser capaz de mostrarnos el subtotal a partir de cada producto que llevamos y así como en el supermercado me debe de permitir eliminar un producto si no me alcanza el dinero.
5. Y por ultimo en el carro de compras me debe permitir actualizar la cantidad de producto si quiero mas de un producto del mismo tipo o quiero dejar de ese producto uno.

## Laboratorio 6.1

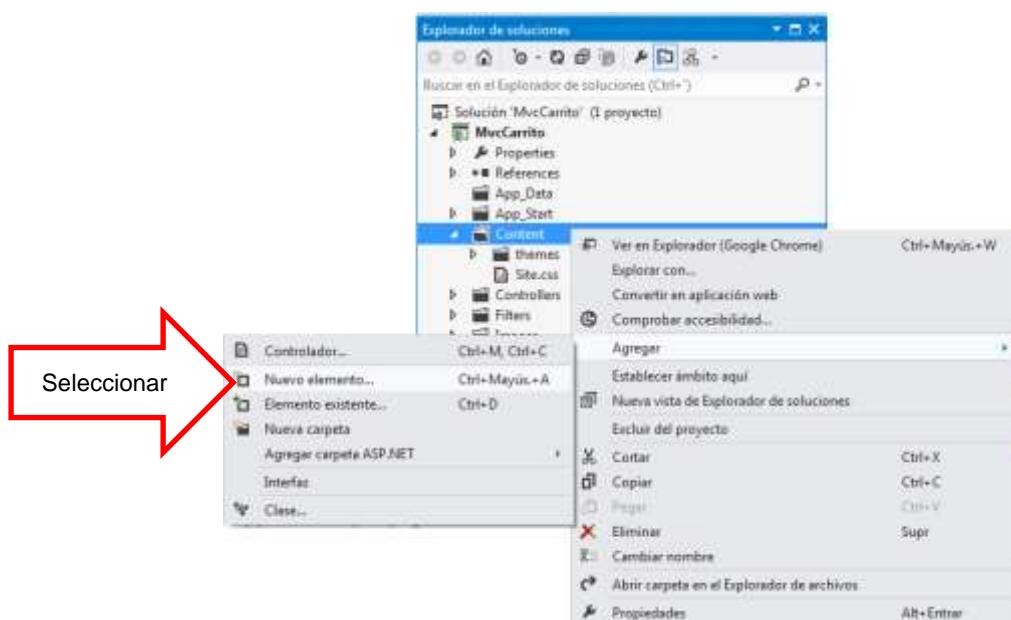
### Implementando el proceso del e-commerce en ASP.NET MVC

Implemente una aplicación ASP.NET MVC que permita registrar las ventas por internet.

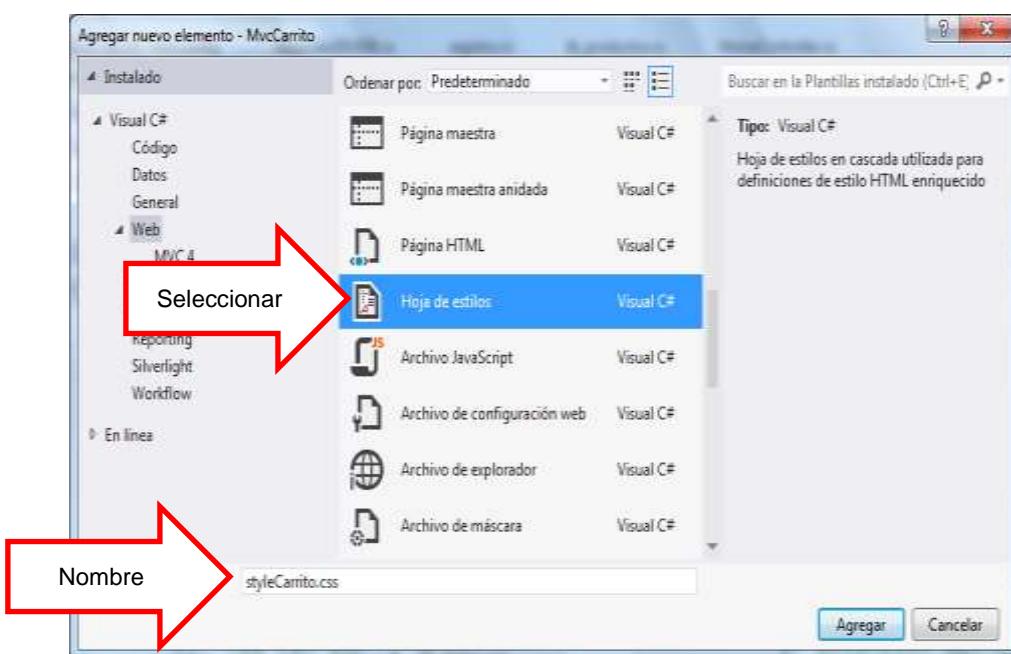
#### Solución.

Creamos un proyecto ASP.NET MVC llamado MvcCarrito

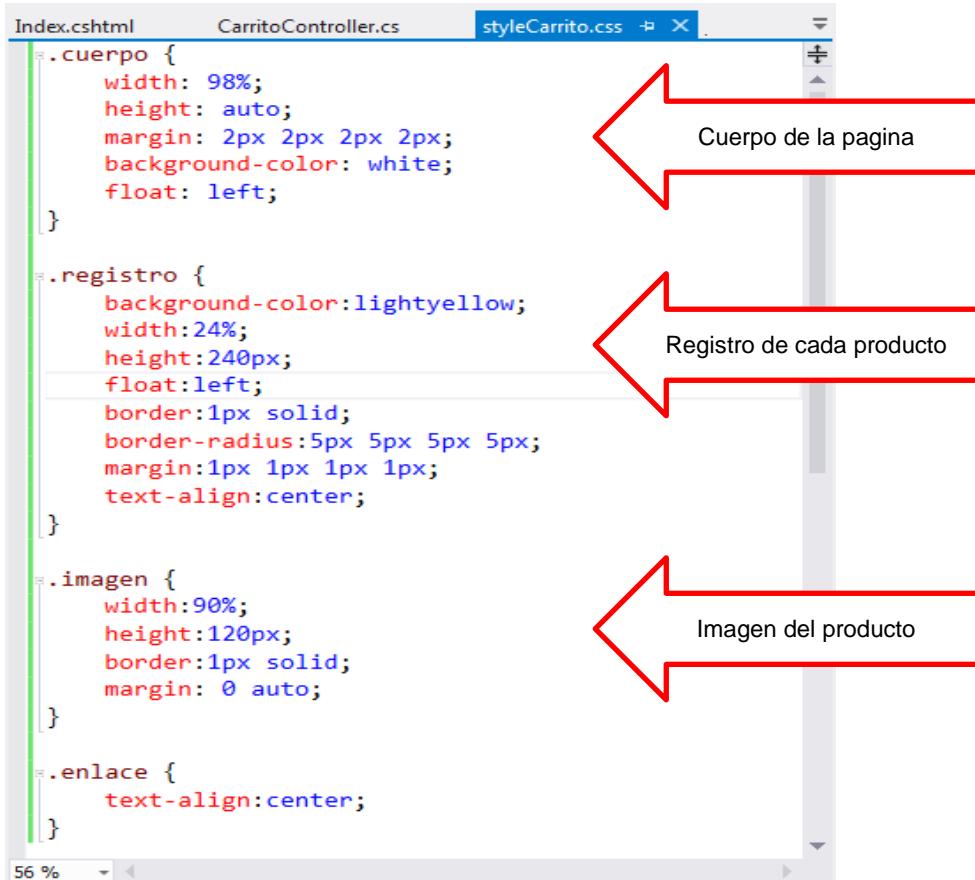
En la carpeta Content vamos a agregar una hoja de Estilo tal como se muestra



En la ventana **Agregar nuevo elemento**, selecciona la plantilla **hoja de estilo** y le asignas el nombre styleCarrito.css, tal como se muestra



Defina las clases en la hoja de estilo tal como se muestra:



```

Index.cshtml      CarritoController.cs      styleCarrito.css
.cuerpo {
    width: 98%;
    height: auto;
    margin: 2px 2px 2px 2px;
    background-color: white;
    float: left;
}

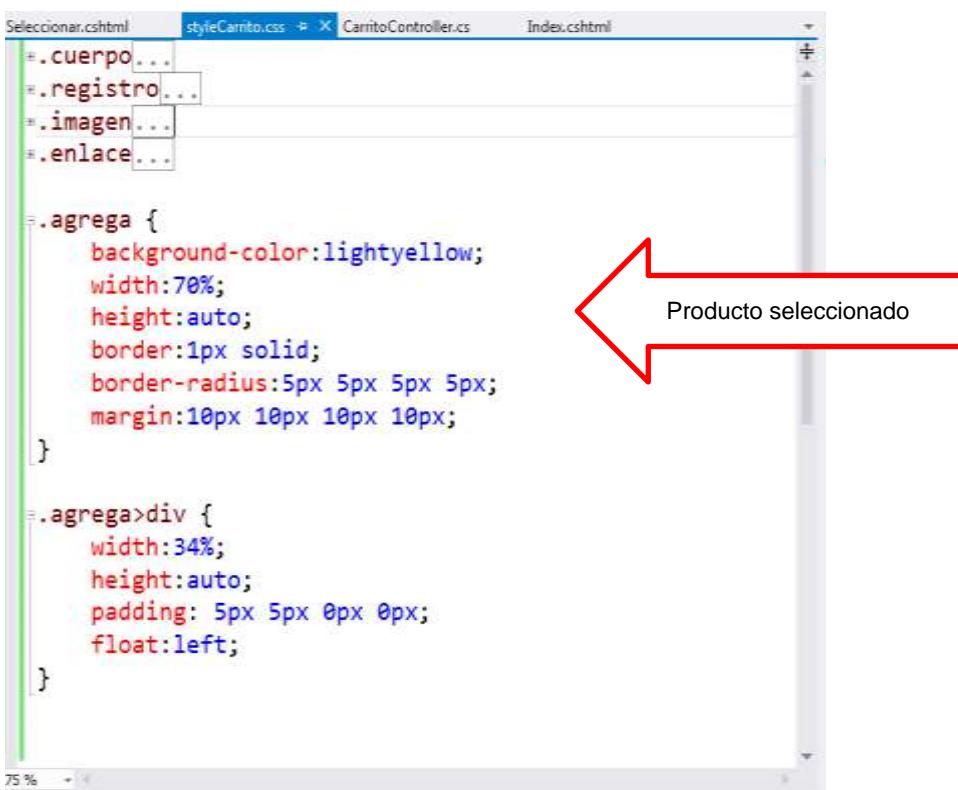
.registro {
    background-color: lightyellow;
    width: 24%;
    height: 240px;
    float: left;
    border: 1px solid;
    border-radius: 5px 5px 5px 5px;
    margin: 1px 1px 1px 1px;
    text-align: center;
}

.imagen {
    width: 90%;
    height: 120px;
    border: 1px solid;
    margin: 0 auto;
}

.enlace {
    text-align: center;
}

```

56 %



```

Seleccionar.cshtml      styleCarrito.css      CarritoController.cs      Index.cshtml
.cuerpo...
.registro...
.imagen...
.enlace...

.agrega {
    background-color: lightyellow;
    width: 70%;
    height: auto;
    border: 1px solid;
    border-radius: 5px 5px 5px 5px;
    margin: 10px 10px 10px 10px;
}

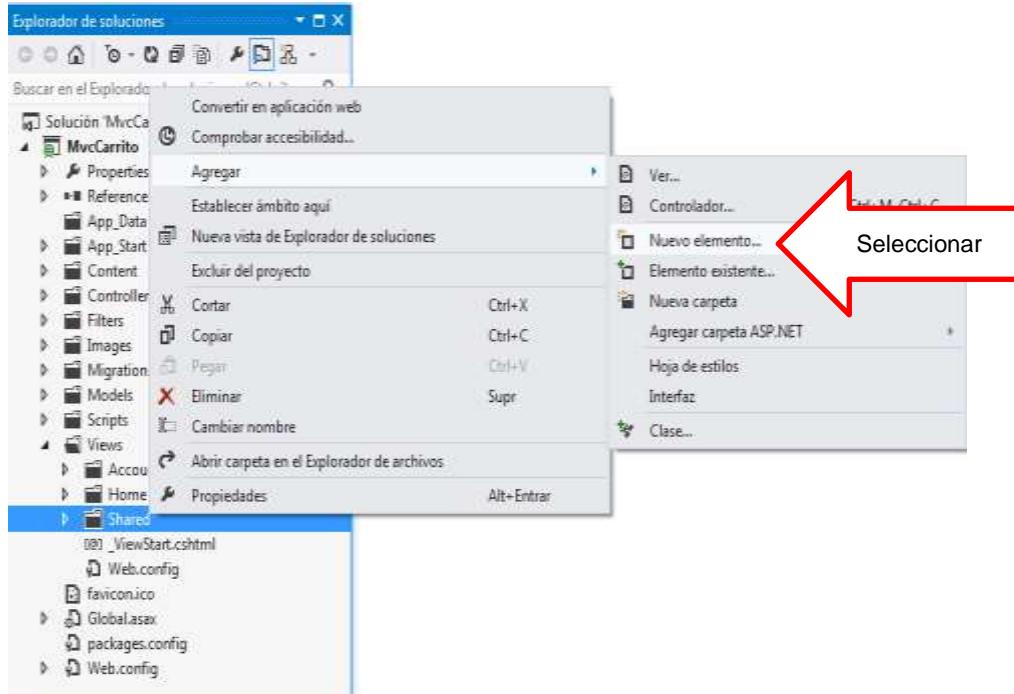
.agrega>div {
    width: 34%;
    height: auto;
    padding: 5px 5px 0px 0px;
    float: left;
}

```

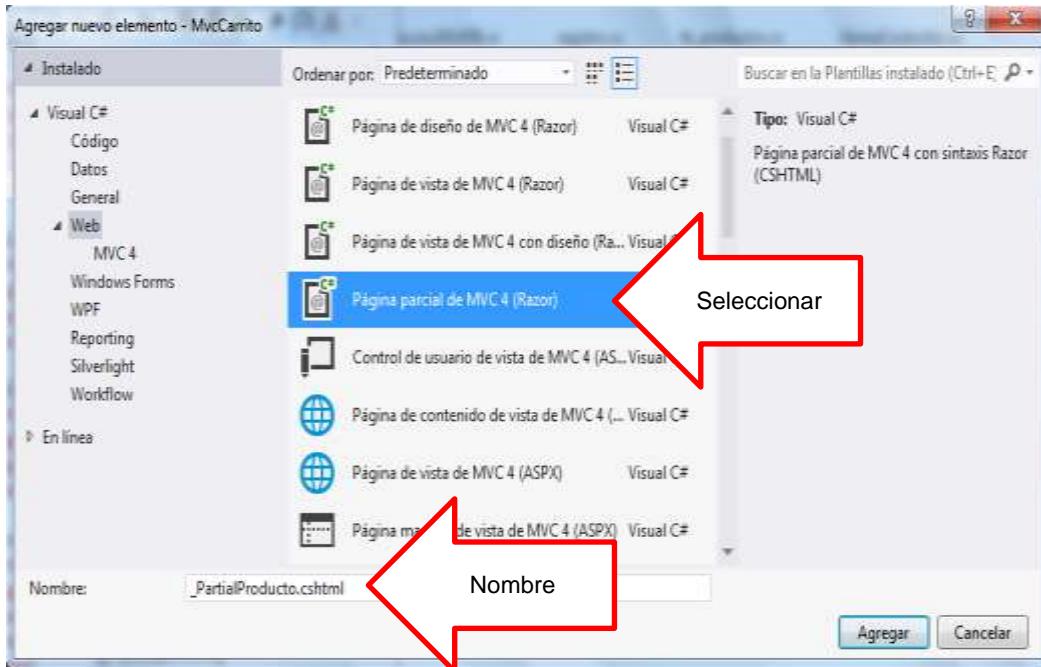
75 %

### Agregando una Vista Parcial

En la carpeta Shared, agrega una página parcial Razor, llamada \_PartialProducto, tal como se muestra



Selecciona la plantilla Página parcial de MVC (Razor) y asigne su nombre



En la clase parcial definimos el diseño de la página para listar los productos, tal como se muestra

```

Index.cshtml    CarritoController.cs    _PartialProducto.cshtml    RouteConfig.cs    Configuration.cs    Web.config    Negocios2014DB.cs
@model IEnumerable<MvcCarrito.Models.tb_productos>
<link href("~/Content/styleCarrito.css" type="text/css" rel="stylesheet" />
<div class="cuerpo">
    @foreach (var reg in Model)
    {
        <div class="registro">
            <table>
                <tr>
                    <td><img class="imagen" src=@Url.Content(string.Format("~/imagenes/{0}.jpg",@reg.idproducto)) /></td>
                </tr>
                <tr>
                    <td>@reg.nombreproducto</td>
                </tr>
                <tr>
                    <td>@reg.preciounidad</td>
                </tr>
                <tr>
                    <td colspan="2" class="enlace">
                        @Html.ActionLink("Seleccionar", "Seleccionar", new { id=@reg.idproducto })
                    </td>
                </tr>
            </table>
        </div>
    }
</div>

```

En la clase parcial `_PartialRegistro` definimos el diseño para visualizar los datos del producto seleccionado, tal como se muestra

```

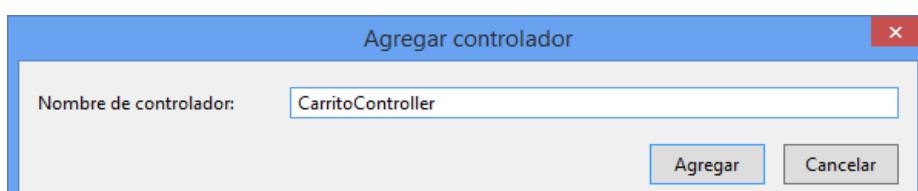
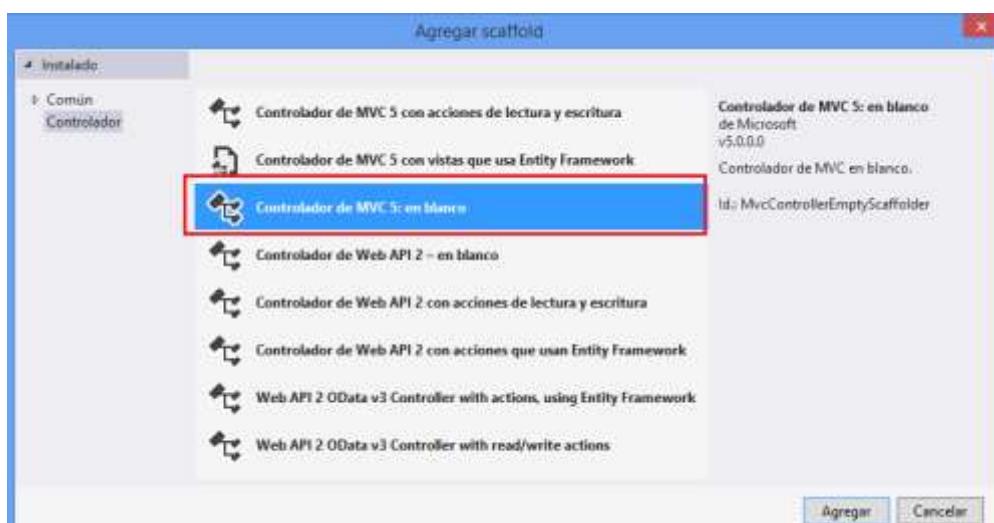
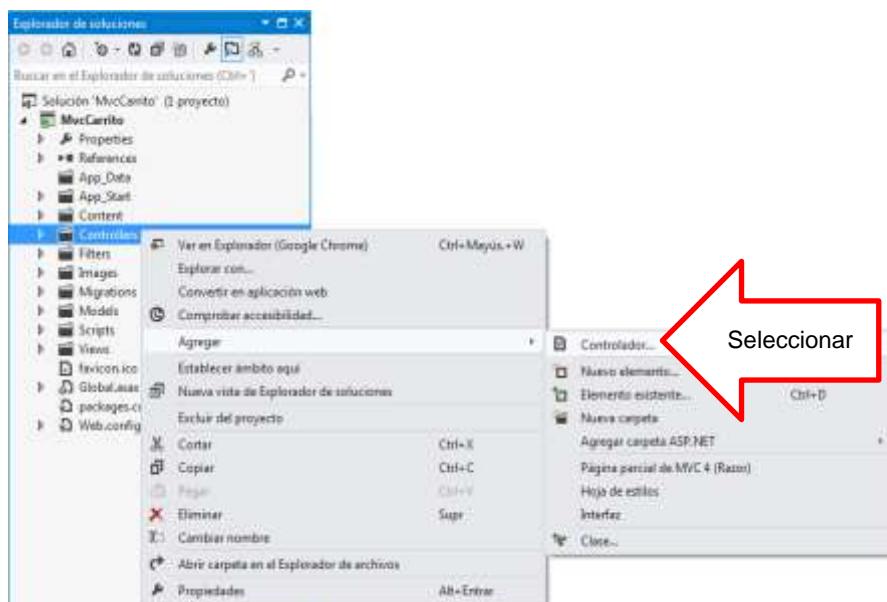
_PartialRegistro.cshtml    Seleccionar.cshtml    styleCarrito.css    CarritoController.cs    Index.cshtml
@model MvcCarrito.Models.tb_productos
<link href "~/Content/styleCarrito.css" type="text/css" rel="stylesheet" />
<div class="cuerpo">
<div class="agrega">
    <div>
        Código:@Model.idproducto<br />
        Producto:@Model.nombreproducto<br />
        Medida:@Model.umedida<br />
        Precio:@Model.preciounidad<br />
    </div>

    <div>
        <img class="imagen" src=@Url.Content(string.Format("~/imagenes/{0}.jpg",Model.idproducto)) />
    </div>
</div>
</div>

```

## Agregando un Controlador

A continuación agregamos el controlador llamado CarritoController, tal como se muestra



En el controlador Carrito, defina el ActionResult Index, que retorna los registros de tb\_productos, evalúo si la Session carrito exists, de no ser así, lo defino

```

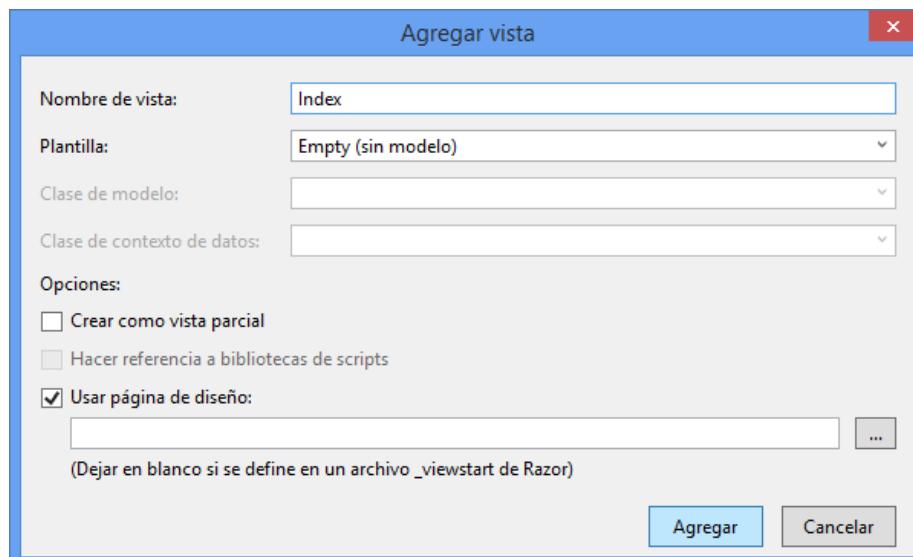
Index.cshtml  CarritoController.cs*  ✘
MvcCarrito.Controllers.CarritoController      + | @ Index()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcCarrito.Models;
namespace MvcCarrito.Controllers
{
    public class CarritoController : Controller
    {
        Negocios2014DB db=new Negocios2014DB();

        public ActionResult Index()
        {
            //evaluo si la sesion esta vacia
            if (Session["carrito"] == null)
            {
                List<registro> detalle = new List<registro>();
                Session["carrito"] = detalle;
            }
            return View(db.Productos.ToList());
        }
    }
}

```

### Agregando la Vista

Agregar la vista llamada Index, de plantilla Empty; presiona el botón Agregar



A continuación defina la Vista del Action Index, diseña la vista tal como se muestra.

```
Index.cshtml 1 |  PartialCompracarrito.cshtml  CompraController.cs  CuentaController.cs
@model IEnumerable<MvcCarrito.Models.tb_productos>
@{
    ViewBag.Title = "Index";
}



## Index



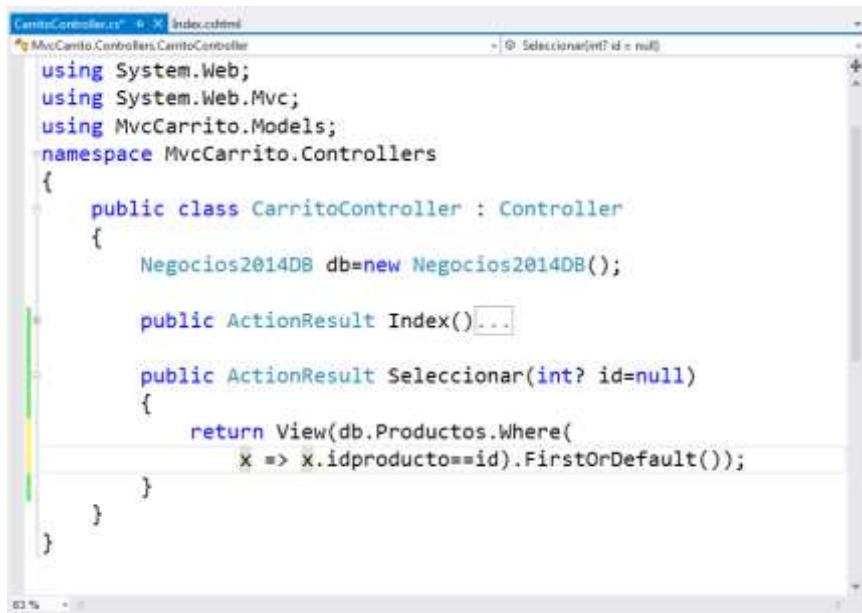
@Html.ActionLink("Realizar compra", "Comprar")


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

Ejecuta la aplicación, donde se visualiza la lista de productos

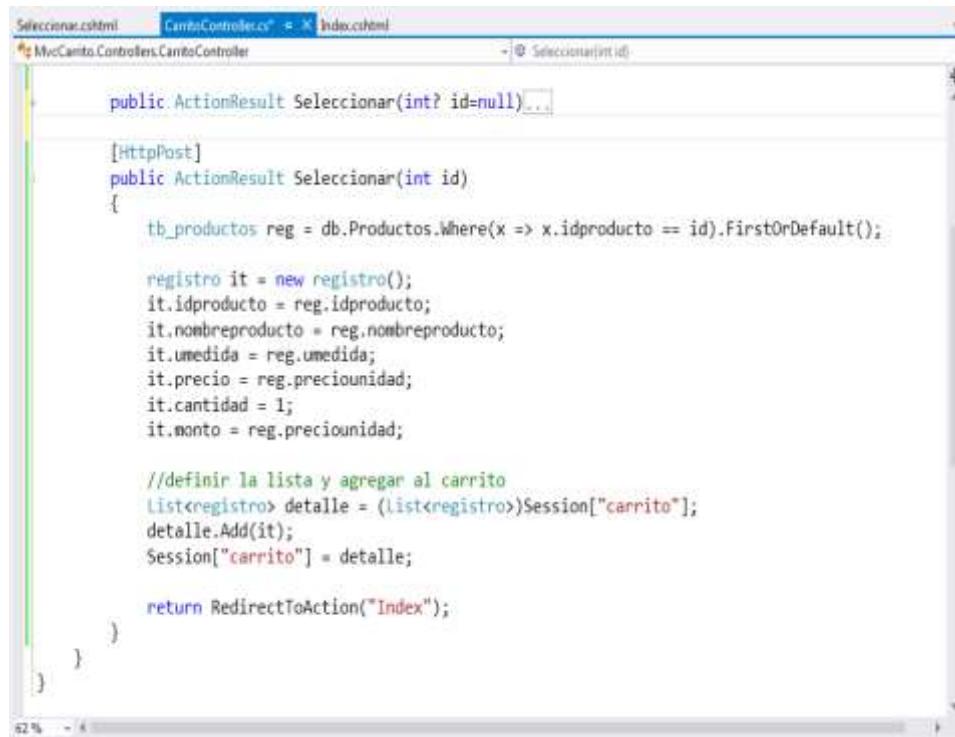
Nombre del Producto	Precio	Opción
Ta Chinesca	18	Seleccionar
Cerdo churrasco Barbacoa	19	Seleccionar
Arroz de regalo	18	Seleccionar
Sándwich Cajón del chef Anton	22	Seleccionar
Mezcla Qumita del chef Anton	21	Seleccionar
Mermelada de granadilla de la abuela	21	Seleccionar
Queso seco urgencias del chef Beto	20	Seleccionar
Queso de queso Marañoncito	40	Seleccionar
Baby Vino Verde	...	...
Peri Peri	...	...
Queso Calabazas	...	...
Queso Manchego La Patata	...	...

En el controlador Carrito, defina el método Seleccionar, donde recibe el id, busca el registro en la tabla tb\_productos, enviando el registro.



```
CarritoController.cs  Índice.cshtml
MvcCarrito.Controllers.CarritoController
using System.Web;
using System.Web.Mvc;
using MvcCarrito.Models;
namespace MvcCarrito.Controllers
{
    public class CarritoController : Controller
    {
        Negocios2014DB db=new Negocios2014DB();

        public ActionResult Index()...
        public ActionResult Seleccionar(int? id=null)
        {
            return View(db.Productos.Where(
                x => x.idproducto==id).FirstOrDefault());
        }
    }
}
```



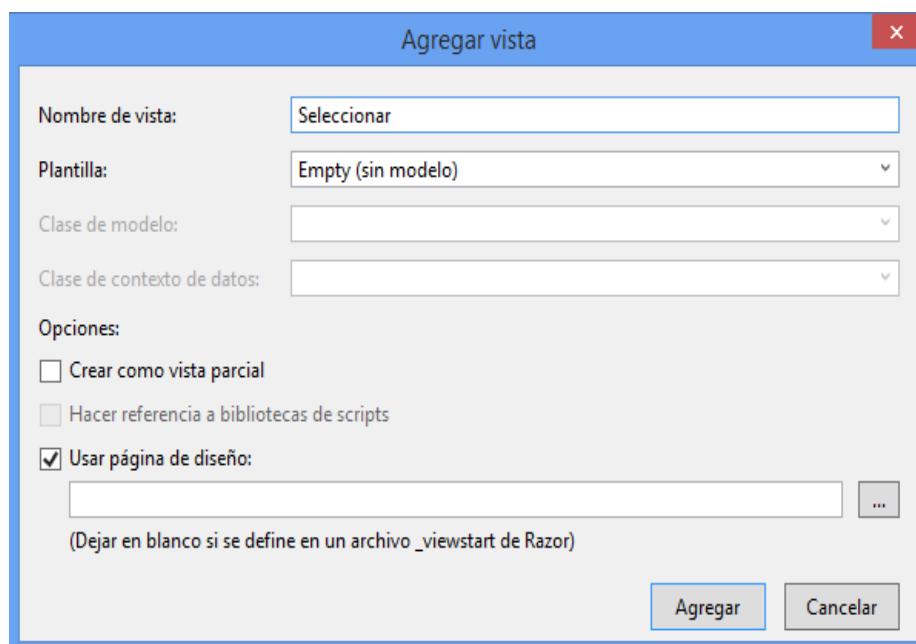
```
Seleccionar.cshtml  CarritoController.cs  Índice.cshtml
MvcCarrito.Controllers.CarritoController
public ActionResult Seleccionar(int? id=null)...
[HttpPost]
public ActionResult Seleccionar(int id)
{
    tb_productos reg = db.Productos.Where(x => x.idproducto == id).FirstOrDefault();

    registro it = new registro();
    it.idproducto = reg.idproducto;
    it.nombreproducto = reg.nombreproducto;
    it.umedida = reg.umedida;
    it.precio = reg.preciounidad;
    it.cantidad = 1;
    it.monto = reg.preciounidad;

    //definir la lista y agregar al carrito
    List<registro> detalle = (List<registro>)Session["carrito"];
    detalle.Add(it);
    Session["carrito"] = detalle;

    return RedirectToAction("Index");
}
}
```

A continuación agregar la Vista Seleccionar, tal como se muestra



Defina la vista a continuación utilizando la clase parcial `_PartialRegistro`

```
Seleccionar.cshtml # X CarritoController.cs Index.cshtml
@model MvcCarrito.Models.tb_productos
 @{
    ViewBag.Title = "Seleccionar";
}



## Producto Seleccionado



@Html.ActionLink("Ir al Principal", "Index")


@Html.Partial("_PartialRegistro")

@using (Html.BeginForm())
{
    <div>
        <input type="submit" value="Agregar" />
    </div>
}
```

A continuación ejecuta el proyecto, al seleccionar el producto, mostraremos sus datos, tal como se muestra



## Realizar la Compra

En el controlador Carrito, defina el ActionResult Comprar(); el cual retorna los registros de Session["carrito"] y almacena el total del monto en el ViewBag.mt, tal como se muestra.

```

    Seleccionar.cshtml   CarritoController.cs*  Index.cshtml
    MvcCarrito.Controllers.CarritoController
    - | @ Comprar()

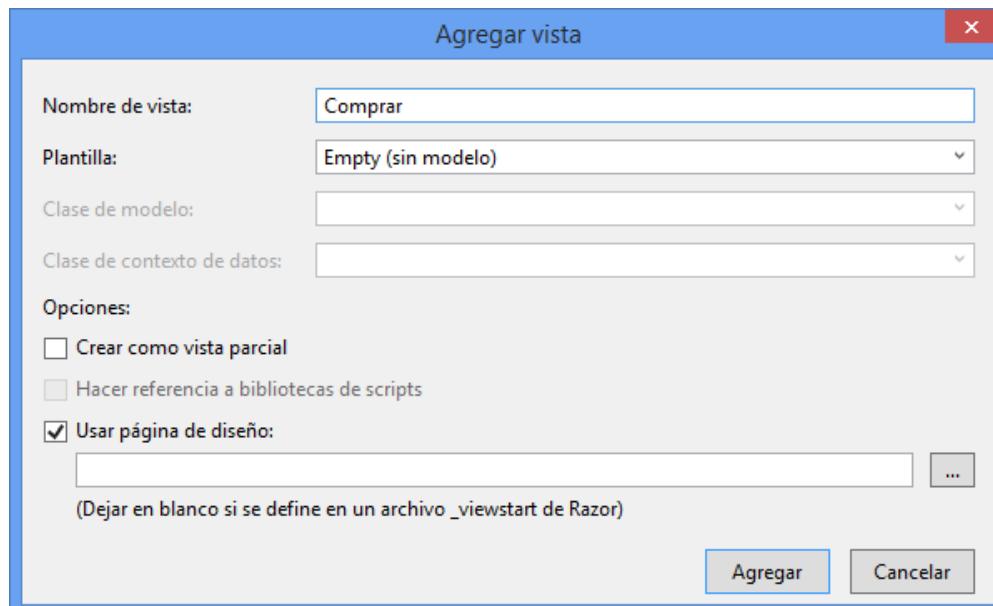
    [HttpPost]
    public ActionResult Seleccionar(int id)...

    public ActionResult Comprar()
    {
        List<registro> detalle = (List<registro>)Session["carrito"];
        decimal mt = 0;
        foreach (registro it in detalle) { mt += it.monto; }

        ViewBag.mt = mt;
        return View(detalle);
    }
}

```

Defina la vista Comprar, tal como se muestra



Defina la vista parcial: \_PartialComprar la cual se agregará en la Vista Comprar

```
PartialComprar.cshtml  Comprar.cshtml  CarritoController.cs
@model IEnumerable<MvcCarrito.Models.registro>
<link href="~/Content/styleCarrito.css" type="text/css" rel="stylesheet" />

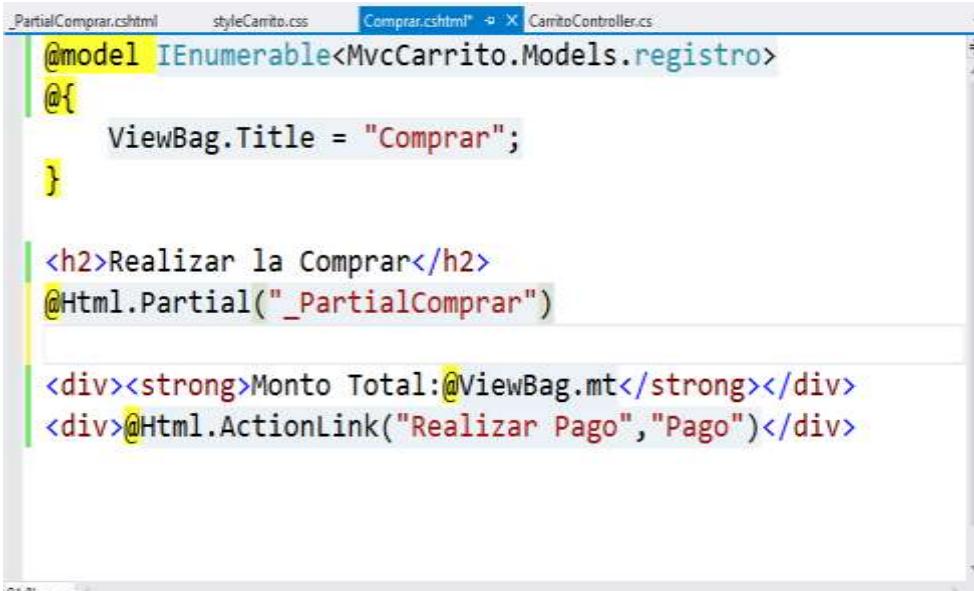

|        |             |        |          |       |  |
|--------|-------------|--------|----------|-------|--|
| Código | Descripción | Precio | Cantidad | Monto |  |
|        |             |        |          |       |  |


@foreach (var reg in Model){
| @reg.idproducto | @reg.nombreproducto | @reg.precio | @reg.cantidad | @reg.monto | @Html.ActionLink("Elimina", "Elimina", new {id=@reg.idproducto}) |

}


```

A continuación defina la vista Comprar, tal como se muestra



```
PartialComprar.cshtml    styleCarrito.css    Comprar.cshtml*  X CarritoController.cs
@model IEnumerable<MvcCarrito.Models.registro>
 @{
     ViewBag.Title = "Comprar";
 }

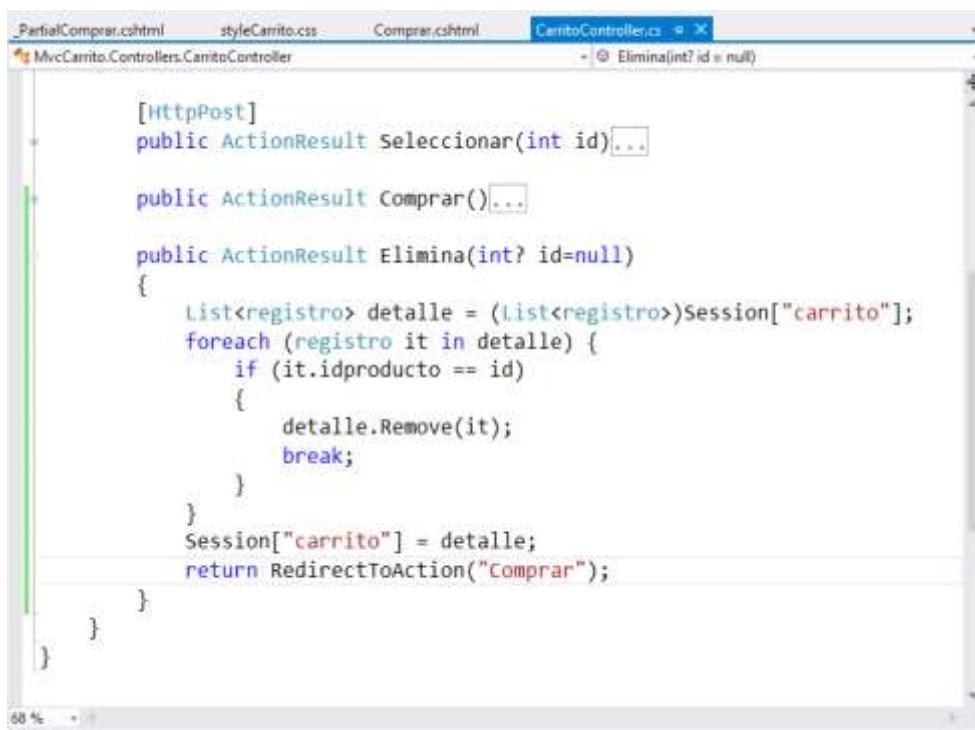
<h2>Realizar la Comprar</h2>
@Html.Partial("_PartialComprar")

<div><strong>Monto Total:@ViewBag.mt</strong></div>
<div>@Html.ActionLink("Realizar Pago", "Pago")</div>
```

Al ejecutar el proyecto, agrega algunos productos. Al presionar la opción realizar Compra, se visualiza tal como se muestra



ActionResult para eliminar el registro del producto seleccionado



```

    ParcialComprar.cshtml      styleCarrito.css      Comprar.cshtml      CarritoController.cs
   MvcCarrito.Controllers.CarritoController
    Elimina(int? id = null)

    [HttpPost]
    public ActionResult Seleccionar(int id)...

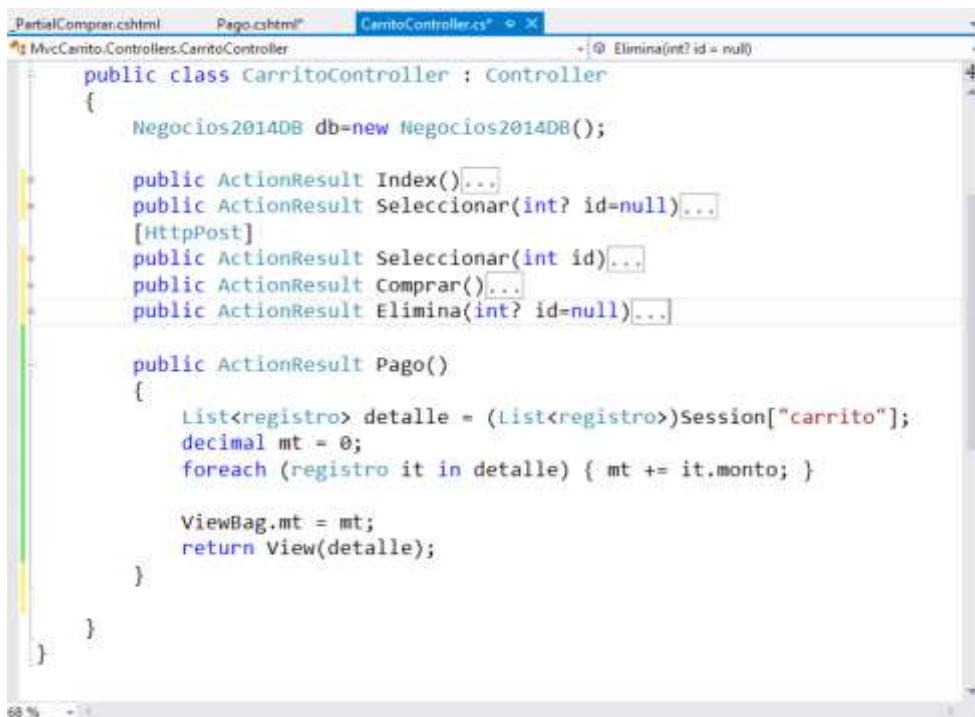
    public ActionResult Comprar()...

    public ActionResult Elimina(int? id=null)
    {
        List<registro> detalle = (List<registro>)Session["carrito"];
        foreach (registro it in detalle) {
            if (it.idproducto == id)
            {
                detalle.Remove(it);
                break;
            }
        }
        Session["carrito"] = detalle;
        return RedirectToAction("Comprar");
    }
}
}

```

## Realizar el Pago

Defina el ActionResult Pago, el cual envía los datos de la compra, tal como se muestra. A continuación defina la vista del método.



```

    ParcialComprar.cshtml      Pago.cshtml      CarritoController.cs
   MvcCarrito.Controllers.CarritoController
    Elimina(int? id = null)

    public class CarritoController : Controller
    {
        Negocios2014DB db=new Negocios2014DB();

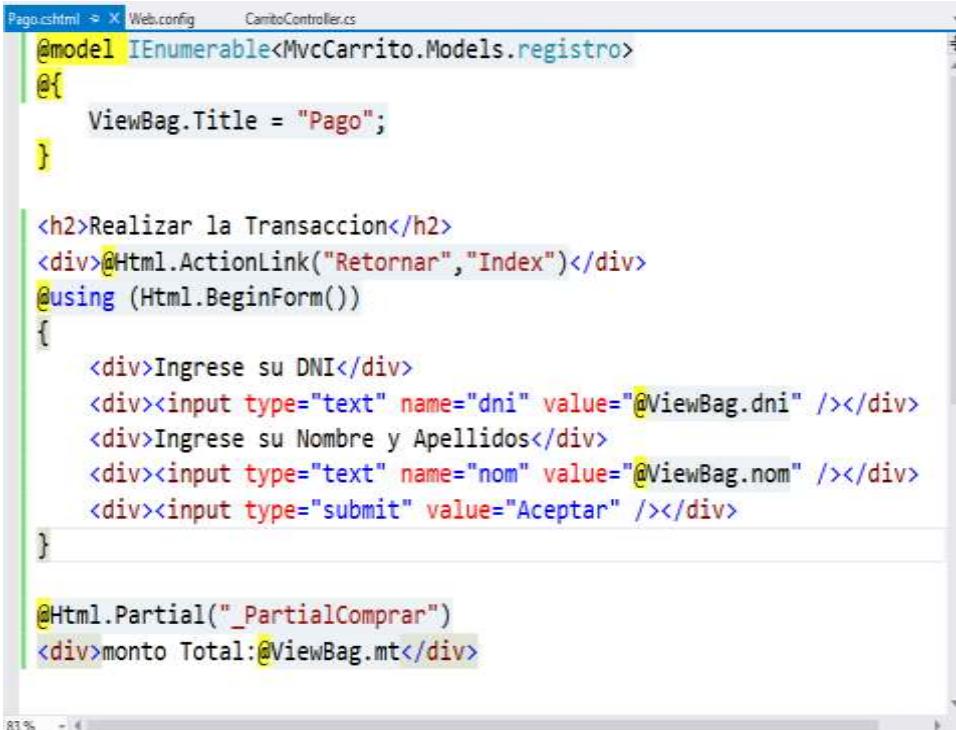
        public ActionResult Index()...
        public ActionResult Seleccionar(int? id=null)... 
        [HttpPost]
        public ActionResult Seleccionar(int id)...
        public ActionResult Comprar()...
        public ActionResult Elimina(int? id=null)...

        public ActionResult Pago()
        {
            List<registro> detalle = (List<registro>)Session["carrito"];
            decimal mt = 0;
            foreach (registro it in detalle) { mt += it.monto; }

            ViewBag.mt = mt;
            return View(detalle);
        }
    }
}

```

### Vista del ActionResult Pago



```

@model IEnumerable<MvcCarrito.Models.registro>
 @{
     ViewBag.Title = "Pago";
 }

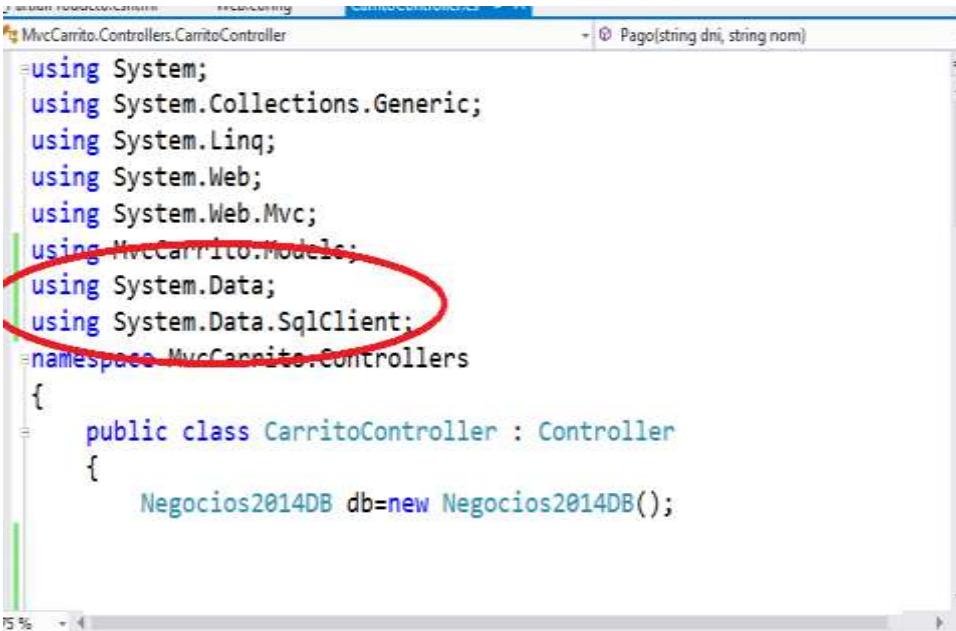
<h2>Realizar la Transaccion</h2>
<div>@Html.ActionLink("Retornar", "Index")</div>
@using (Html.BeginForm())
{
    <div>Ingrese su DNI</div>
    <div><input type="text" name="dni" value="@ViewBag.dni" /></div>
    <div>Ingrese su Nombre y Apellidos</div>
    <div><input type="text" name="nom" value="@ViewBag.nom" /></div>
    <div><input type="submit" value="Aceptar" /></div>
}

@Html.Partial("_PartialComprar")
<div>monto Total:@ViewBag.mt</div>

```

### Almacenar la Transacción

Defina las librerías de trabajo dentro del controlador

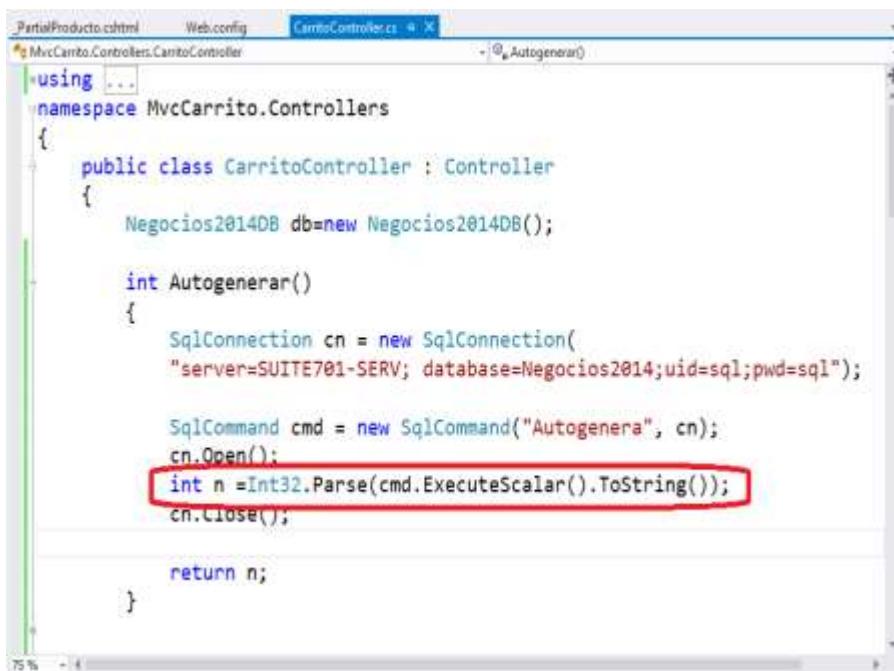


```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcCarrito.Models;
using System.Data;
using System.Data.SqlClient;
namespace MvcCarrito.Controllers
{
    public class CarritoController : Controller
    {
        Negocios2014DB db=new Negocios2014DB();
    }
}

```

Función autogenera(), retorna el siguiente valor del número del pedido. Utilice la función ExecuteScalar en el proceso



```

PartialProducto.cshtml    Web.config    CarritoController.cs  ✘
MvcCarrito.Controllers.CarritoController - [④ Autogenerar()]
using ...
namespace MvcCarrito.Controllers
{
    public class CarritoController : Controller
    {
        Negocios2014DB db=new Negocios2014DB();

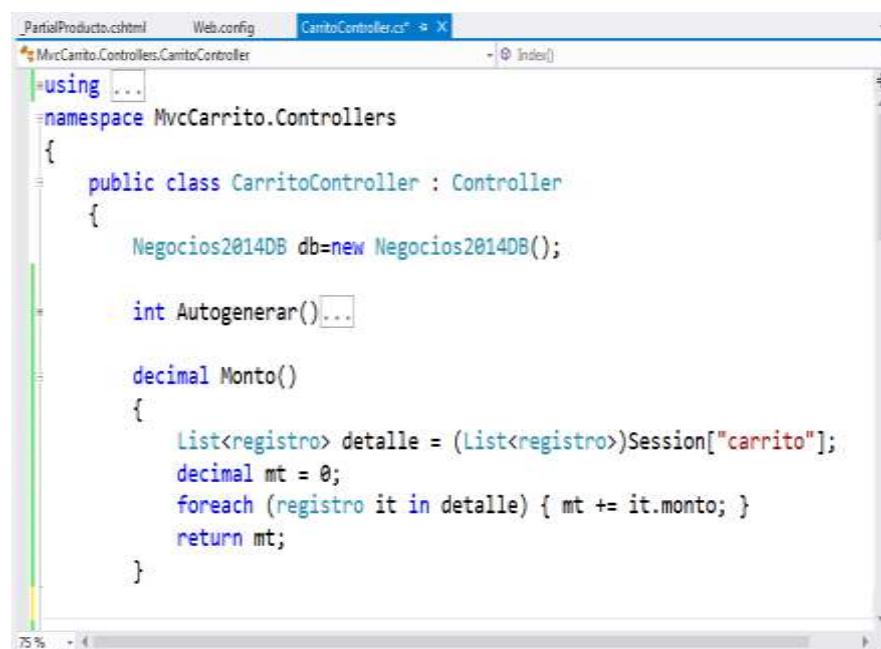
        int Autogenerar()
        {
            SqlConnection cn = new SqlConnection(
                "server=SUITE701-SERV; database=Negocios2014;uid=sq1;pwd=sq1");

            SqlCommand cmd = new SqlCommand("Autogenera", cn);
            cn.Open();
            int n =Int32.Parse(cmd.ExecuteScalar().ToString());
            cn.Close();

            return n;
        }
    }
}

```

Función Monto(), retorna el total acumulados del campo monto de detalle.



```

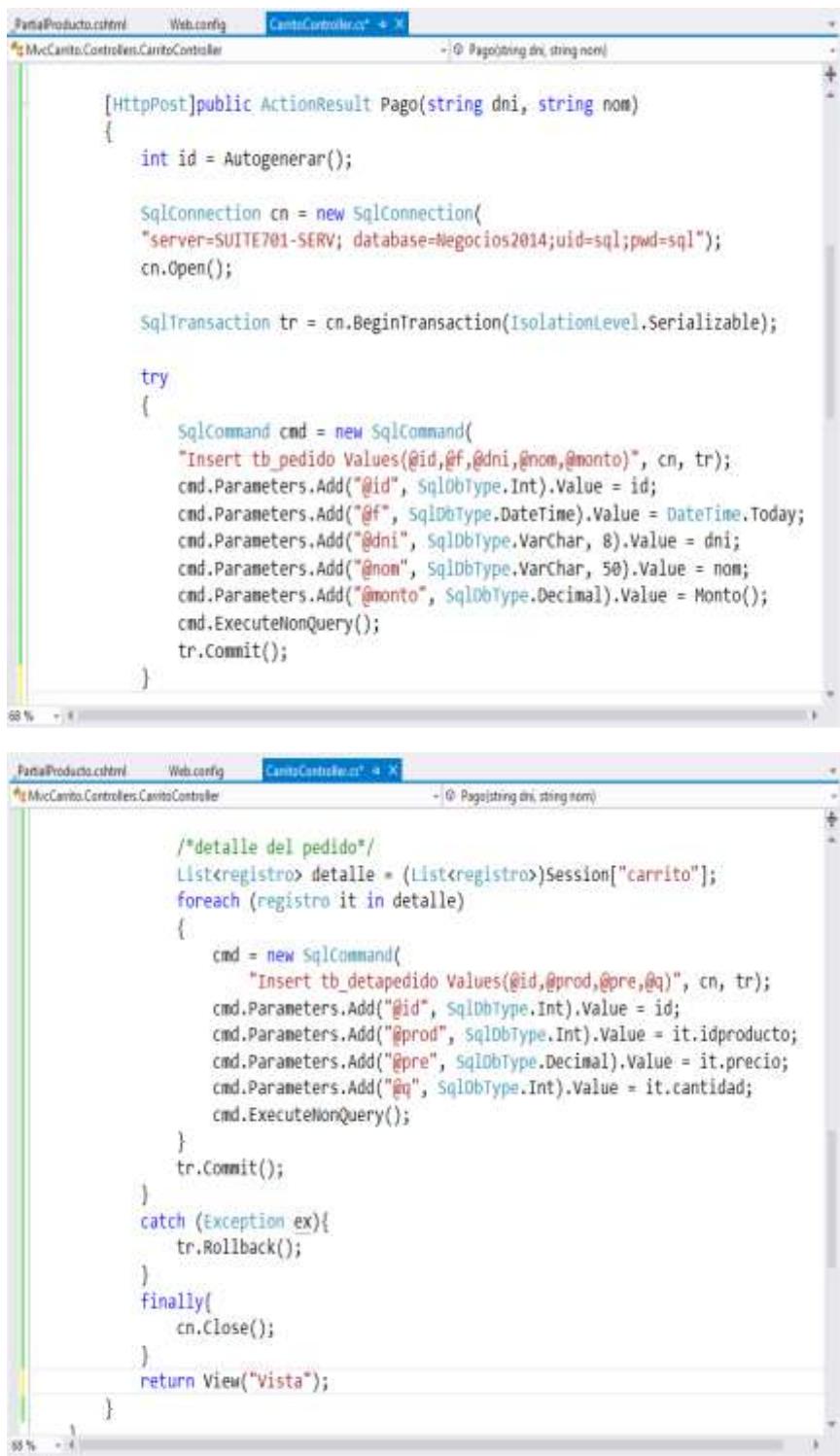
PartialProducto.cshtml    Web.config    CarritoController.cs  ✘
MvcCarrito.Controllers.CarritoController - [④ Index()]
using ...
namespace MvcCarrito.Controllers
{
    public class CarritoController : Controller
    {
        Negocios2014DB db=new Negocios2014DB();

        int Autogenerar()...

        decimal Monto()
        {
            List<registro> detalle = (List<registro>)Session["carrito"];
            decimal mt = 0;
            foreach (registro it in detalle) { mt += it.monto; }
            return mt;
        }
    }
}

```

A continuación defina el Actionresult Pago, el cual ejecuta el proceso para agrega registros a la base de datos.



```

    // CarritoController.cs
    [HttpPost]
    public ActionResult Pago(string dni, string nom)
    {
        int id = Autogenerar();
        SqlConnection cn = new SqlConnection(
            "server=SUITE701-SERV; database=Negocios2014;uid=sql;pwd=sql");
        cn.Open();
        SqlTransaction tr = cn.BeginTransaction(IsolationLevel.Serializable);
        try
        {
            SqlCommand cmd = new SqlCommand(
                "Insert tb_pedido Values(@id,@f,@dni,@nom,@monto)", cn, tr);
            cmd.Parameters.Add("@id", SqlDbType.Int).Value = id;
            cmd.Parameters.Add("@f", SqlDbType.DateTime).Value = DateTime.Today;
            cmd.Parameters.Add("@dni", SqlDbType.VarChar, 8).Value = dni;
            cmd.Parameters.Add("@nom", SqlDbType.VarChar, 50).Value = nom;
            cmd.Parameters.Add("@monto", SqlDbType.Decimal).Value = Monto();
            cmd.ExecuteNonQuery();
            tr.Commit();
        }
        catch (Exception ex)
        {
            tr.Rollback();
        }
        finally
        {
            cn.Close();
        }
        return View("Vista");
    }

```

## Archivos del proceso

The screenshot shows a SQL script window titled "SQLCarrito.sql - SUI...ocios2014 (sql (53))". The script contains the following SQL code:

```
use Negocios2014
go
Create table tb_pedido(
    idpedido int primary key,
    fecpedido datetime default(getdate()),
    dnicli varchar(8),
    nomcli varchar(60),
    monto decimal default(0)
)

Create table tb_detapedido(
    idpedido int references tb_pedido,
    idproducto int references tb_productos,
    precio decimal,
    cantidad int
)
go

create proc Autogenera
As
    Select count(*)+1 from tb_pedido
go
```

The status bar at the bottom indicates "62 %", a green checkmark icon, "Consulta ejecutada correctamente.", and "00:00:00 | 1 filas".

# Resumen

- El comercio electrónico, o E-commerce, como es conocido en gran cantidad de portales existentes en la web, es definido por el Centro Global de Mercado Electrónico como "cualquier forma de transacción o intercambio de información con fines comerciales en la que las partes interactúan utilizando Tecnologías de la Información y Comunicaciones (TIC), en lugar de hacerlo por intercambio o contacto físico directo".
- La actividad comercial en Internet o comercio electrónico, no difiere mucho de la actividad comercial en otros medios, el comercio real, y se basa en los mismos principios: una oferta, una logística y unos sistemas de pago
- La capa de acceso a datos realiza las operaciones CRUD (Crear, Obtener, Actualizar y Borrar), el modelo de objetos que .NET Framework proporciona para estas operaciones es ADO.NET. Los objetivos básicos de cualquier sitio web comercial son tres: atraer visitantes, fidelizarlos y, en consecuencia, venderles nuestros productos o servicios
- Para poder obtener un beneficio con el que rentabilizar las inversiones realizadas que son las que permiten la realización de los dos primeros objetivos. El equilibrio en la aplicación de los recursos necesarios para el cumplimiento de estos objetivos permitirá traspasar el umbral de rentabilidad y convertir la presencia en Internet en un auténtico negocio.
- La aparición y consolidación de las plataformas de e-commerce ha provocado que, en el tramo final del ciclo de compra (en el momento de la transacción), el comprador potencial no interactúe con ninguna persona, sino con un canal web habilitado por la empresa, con el llamado **carrito de compra**
- Los carritos de compra son aplicaciones dinámicas que están destinadas a la venta por internet y que si están confeccionadas a medida pueden integrarse fácilmente dentro de websites o portales existentes, donde el cliente busca comodidad para elegir productos (libros, música, videos, comestibles, indumentaria, artículos para el hogar, electrodomésticos, muebles, juguetes, productos industriales, software, hardware, y un largo etc.) -o servicios- de acuerdo a sus características y precios, y simplicidad para comprar.
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
  - <http://albeverry.blogspot.com/>
  - [http://rogerarandavega.blogspot.com/2014/04/arquitectura-n-capas-estado-del-arte\\_18.html](http://rogerarandavega.blogspot.com/2014/04/arquitectura-n-capas-estado-del-arte_18.html)
  - <http://icomparable.blogspot.com/2011/03/aspnet-mvc-el-mvc-no-es-una-forma-de.html>
  - <http://lgjluis.blogspot.com/2013/11/aplicaciones-en-n-capas-con-net.html>
  - <http://anexsoft.com/p/36/asp-net-mvc-usando-ado-net-y-las-3-capas>



# CONSUMO DE SERVICIOS

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla aplicaciones Web para consumir servicios WCF.

## TEMARIO

### Tema 7: Implementación y consumo de servicios WCF (4 horas)

- 7.1 Introducción a los servicios WCF
- 7.2 Implementación de un servicio simple WCF con acceso a datos.
- 7.3 Consumo de un servicio WCF desde una aplicación Web.
- 7.4 CRUD con WCF

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web consumiendo servicios para el manejo de datos utilizando el patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



## 7. Implementando y consumo de servicios WCF

### 7.1 Introducción a los servicios WCF

Windows Communication Foundation (WCF) es un marco de trabajo para la creación de aplicaciones orientadas a servicios. Con WCF, es posible enviar datos como mensajes asíncronos de un extremo de servicio a otro. Un extremo de servicio puede formar parte de un servicio disponible continuamente hospedado por IIS, o puede ser un servicio hospedado en una aplicación. Un extremo puede ser un cliente de un servicio que solicita datos de un extremo de servicio. Los mensajes pueden ser tan simples como un carácter o una palabra enviados como XML, o tan complejos como un flujo de datos binarios.



**Figura 2: servicio WCF**  
<http://blog.ccaldera.com/creacion-y-consumo-de-servicio-wcf-ef/>

#### Ventajas de utilizar WCF

- Código Centralizado. La principal ventaja del uso de servicios públicos es el código centralizado, por ejemplo, si por algún motivo la estructura de la base de datos de una aplicación cambia, solo se cambiara en el servicio y no en todas las aplicaciones que dependan de ella.
- Seguridad. Gracias a que todo el acceso a la información se encuentra en una sola capa de acceso, es posible tener un mejor control de la misma.
- Escalabilidad. En caso de que se creen nuevas aplicaciones con funcionalidades similares, ya no será necesario crear todas las funciones desde cero, sino que bastara con vincular la aplicación al WCF y tendrá tanta funcionalidad como sus similares.

A continuación se indican unos cuantos escenarios de ejemplo:

- Un servicio seguro para procesar transacciones comerciales.
- Un servicio que proporciona datos actualizados a otras personas, como un informe sobre tráfico u otro servicio de supervisión.
- Un servicio de chat que permite a dos personas comunicarse o intercambiar datos en tiempo real.
- Una aplicación de panel que sondea los datos de uno o varios servicios y los muestra en una presentación lógica.
- Exponer un flujo de trabajo implementado utilizando Windows Workflow Foundation como un servicio WCF.
- Una aplicación de Silverlight para sondear un servicio en busca de las fuentes de datos más recientes.

Los mensajes son enviados entre endpoints. Un endpoint es un lugar donde un mensaje es enviado, o recibido, o ambos.

Un servicio expone uno o más application endpoints, y un cliente genera un endpoint compatible con uno de los endpoints de un servicio dado. La combinación de un servicio y un cliente compatibles conforman un communication stack.

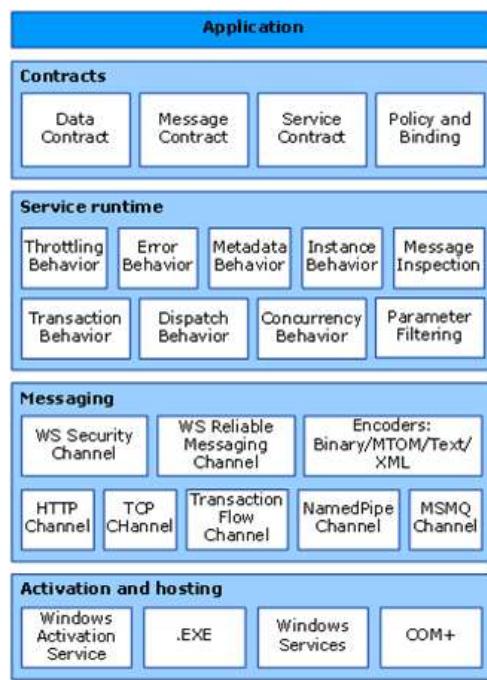


Figura 2: arquitectura  
<http://www.esasp.net/2009/09/wcf-introduccion-y-conceptos-basicos.html>

## 7.2 Implementación de un servicio simple WCF con acceso a datos

Pasos para desarrollar un servicio.

- Definir el Contrato (ServiceContract): Se escribe la interfaz en un lenguaje de programación de .NET, agregando los distintos métodos que serán incluidos en el contrato.

```

Web.config App_Code/CodigosWS.cs App_Code/ICodigosWS.cs Página de inicio
CodigosWS

// NOTA: si cambia el nombre de clase "CodigosWS", también debe act.
// la referencia "CodigosWS" en Web.config.
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single,
    ConcurrencyMode=ConcurrencyMode.Multiple)]
public class CodigosWS : ICodigosWS
{
    private List<CodigoWS> lstPaises;

    #region Miembros de ICodigosWS
    public List<CodigoWS> ObtenerPaises() { ... }

    public void RefrescarPaises() { ... }

    public List<CodigoWS> AdicionarPais(string codigo, string descripcion) { ... }
    #endregion
}

```

- Implementar el Contrato (ServiceContract): Se escribe una clase mediante la cual se implemente la interfaz. Es posible establecer comportamientos a la definición del servicio usando el atributo ServiceBehavior.

```
#region Miembros de ICodigosWS
public List<CodigoWS> ObtenerPaises()
{
    if (lstPaises == null)
        RefrescarPaises();

    return lstPaises;
}
```

Configurar el Servicio: Especificar los endpoints y metadata del servicio, estos son definidos en un archivo de configuración de .NET (Web.config o App.config).

En el Web.Config, configuraremos 2 cosas:

- El Servicio (<services>):
- El nombre del servicio que coincidirá con el nombre de la clase.
- El behaviorConfiguration que es definido más abajo en el mismo fichero.

Dentro del servicio debemos identificar un endpoint del mismo:

- Definiremos el contrato del endpoint.
- El tipo de binding.
- El Comportamiento (<behaviors>):
- Exponer el metadata para que los clientes puedan ver y consumir el servicio.

Llegados a este punto, podemos compilar el servicio y comprobar si todo ha ido bien, para ello abriremos el fichero CodigosWS.svc en el explorador, y debemos obtener una imagen como esta:

```
<system.serviceModel>
    <services>
        <service behaviorConfiguration="returnsFalse"
            name="CodigosWS">
            <endpoint address="" binding="basicHttpBinding"
                contract="ICodigosWS">
                <identity>
                    <dns value="localhost" />
                </identity>
            </endpoint>
        </service>
    </services>

    <behaviors>
        <serviceBehaviors>
            <behavior name="returnsFalse">
                <serviceMetadata httpGetEnabled="true" />
                <serviceDebug
                    includeExceptionDetailInFaults="false" />
            </behavior>
        </serviceBehaviors>
    </behaviors>
</system.serviceModel>
```

- Diseñar una aplicación Hosting del servicio: Web Host dentro del IIS - Self-Host dentro de cualquier proceso .NET - Managed Windows Services - Windows Process Activation Service.



- Diseñar una aplicación cliente del servicio: Acá definiremos las aplicaciones clientes que consumirán el servicio.

```

protected void btnObtener_Click(object sender, EventArgs e)
{
    try
    {
        svcCódigosWS.CódigoWSClient svc =
            new ClienteWeb.svcCódigosWS.CódigoWSClient();
        grdPaises.DataSource = svc.ObtenerPaises();
        grdPaises.DataBind();
    }
    catch (Exception ex)
    {
        lblError.Text = ex.Message;
    }
}

protected void btnAdicionar_Click(object sender, EventArgs e)
{
    try
    {
        svcCódigosWS.CódigoWSClient svc =
            new ClienteWeb.svcCódigosWS.CódigoWSClient();
        svc.ObtenerPaises();
        grdPaises.DataSource = svc.AdicionarPais("ES", "España ");
        grdPaises.DataBind();
    }
    catch (Exception ex)
    {
        lblError.Text = ex.Message;
    }
}

```

### 7.3 Consumo de un servicio WCF desde una aplicación WEB

El escenario de alojamiento elegido puede influir en el lado del consumidor. Puede consumir los servicios WCF de varias maneras. Si usa WCF en el lado del cliente será muy productivo, ya que WCF incluye herramientas que pueden generar clases de proxy para llamar a los servicios WCF. WCF proporciona compatibilidad con los estándares y las herramientas principalmente a través de SvcUtil.exe. Lo usará como herramienta principal de interpretación de metadatos. Eso, en combinación con la capacidad de Framework de WCF para aprovechar la reflexión para interrogar tipos adornados con los atributos apropiados, hace la generación y el uso de Framework de WCF menos complicado que con los marcos existentes. Además, Visual Studio 2015 incluye sencillas características para agregar referencias al servicio a sus proyectos y generar continuamente clases de proxy.

### 7.4 CRUD con WCF

Windows Communication Foundation o WCF es un marco para la creación de aplicaciones orientadas a servicios. Mediante el uso de WCF podemos transferir datos como mensajes asíncronos.

WCF Data Services proporciona un medio fácil de permitir a los clientes utilizar las funcionalidades de back-end en las bases de datos que residen en el lado del servidor, donde crea una capa de servicios de interfaz basado en la nube, la cual es expuesta entre el cliente y el Entity Data Model (EDM), que, por otra parte interactúa con el database.

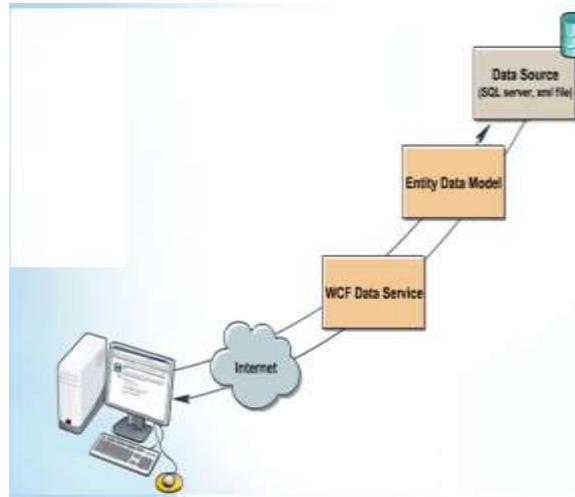


Figura 3: CRUD en WCF

<http://www.dotnetfunda.com/articles/show/1893/a-basic-introduction-to-creation-of-crud-operation-using-odata-web-ser>

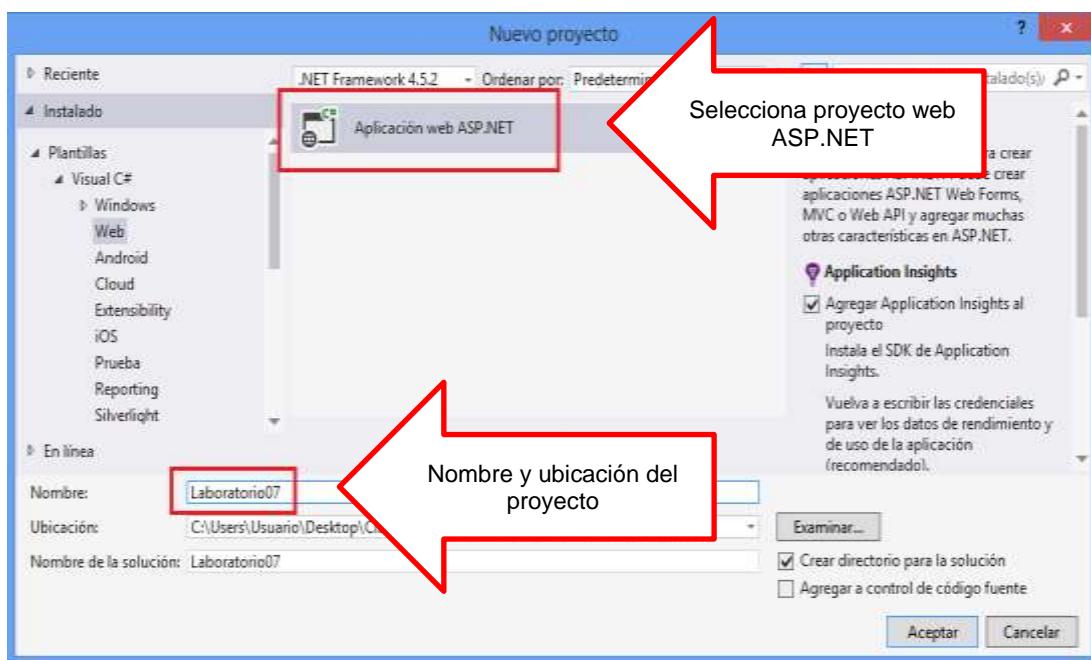
## Laboratorio 7.1

### Implementando consulta en ASP.NET MVC consumiendo un servicio WCF

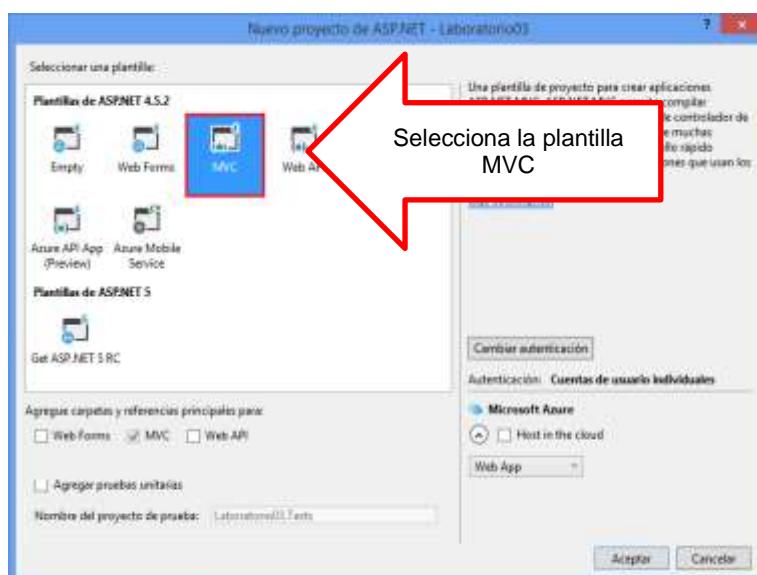
Implemente un proyecto ASP.NET MVC donde permita realizar las operaciones de consulta de datos consumiendo un servicio WCF.

#### SOLUCION

Crear un nuevo proyecto tipo Aplicación web ASP.NET, tal como se muestra.

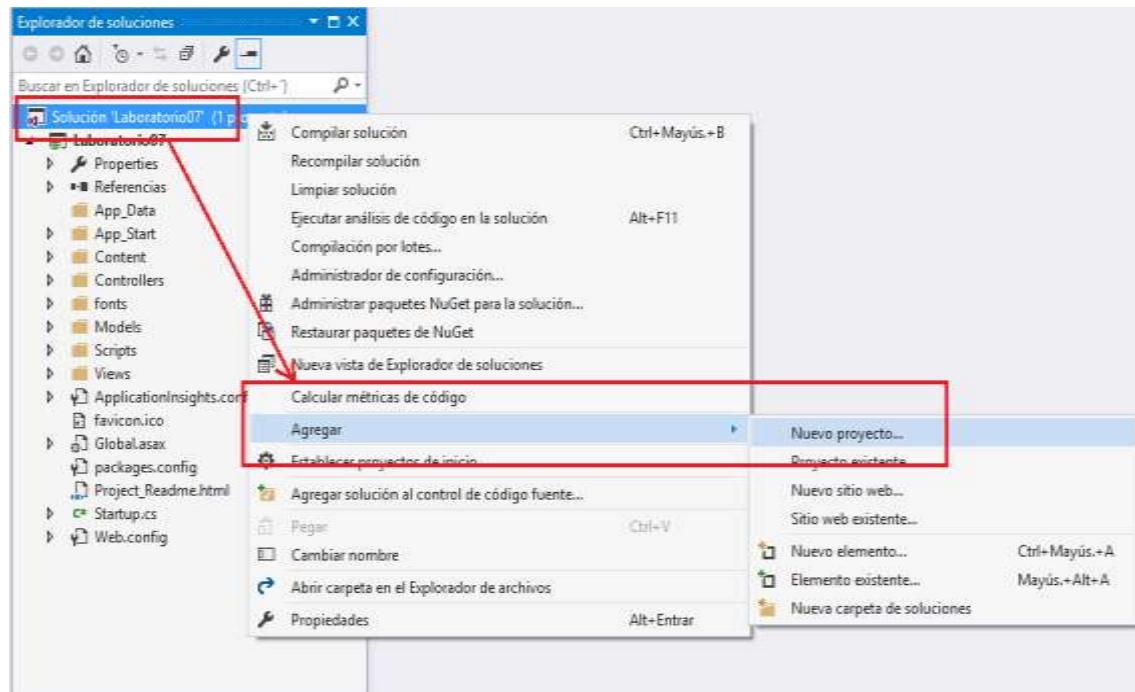


Selecciona la plantilla del proyecto web: MVC, tal como se muestra. Para continuar presionar el botón AGREGAR

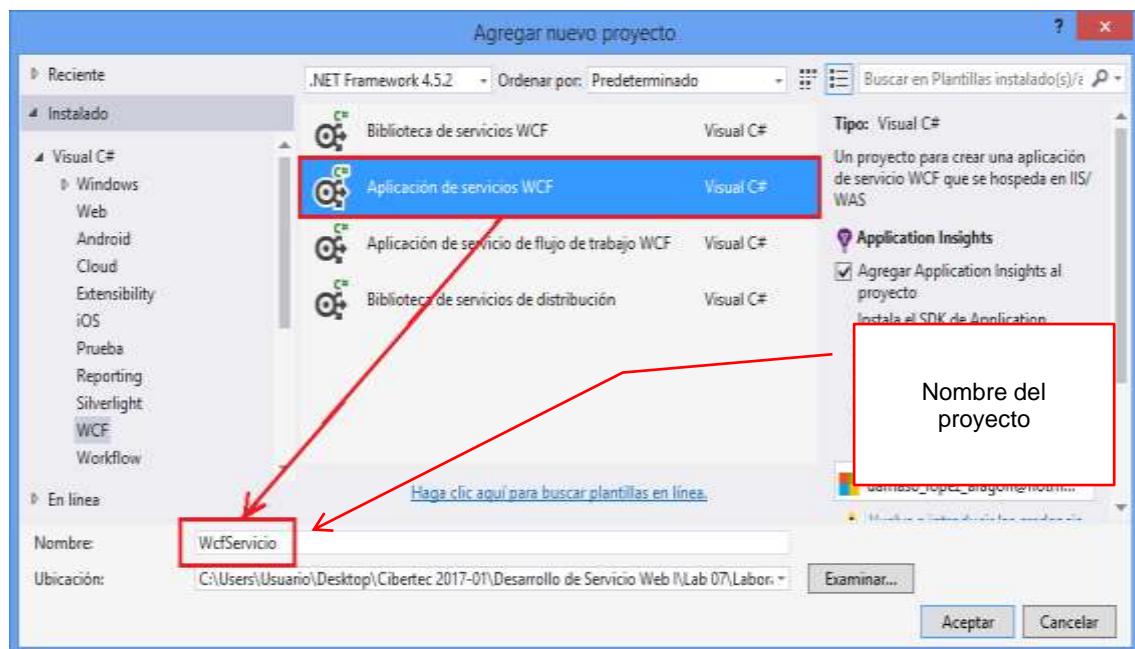


## Trabajando con el Servicio Web

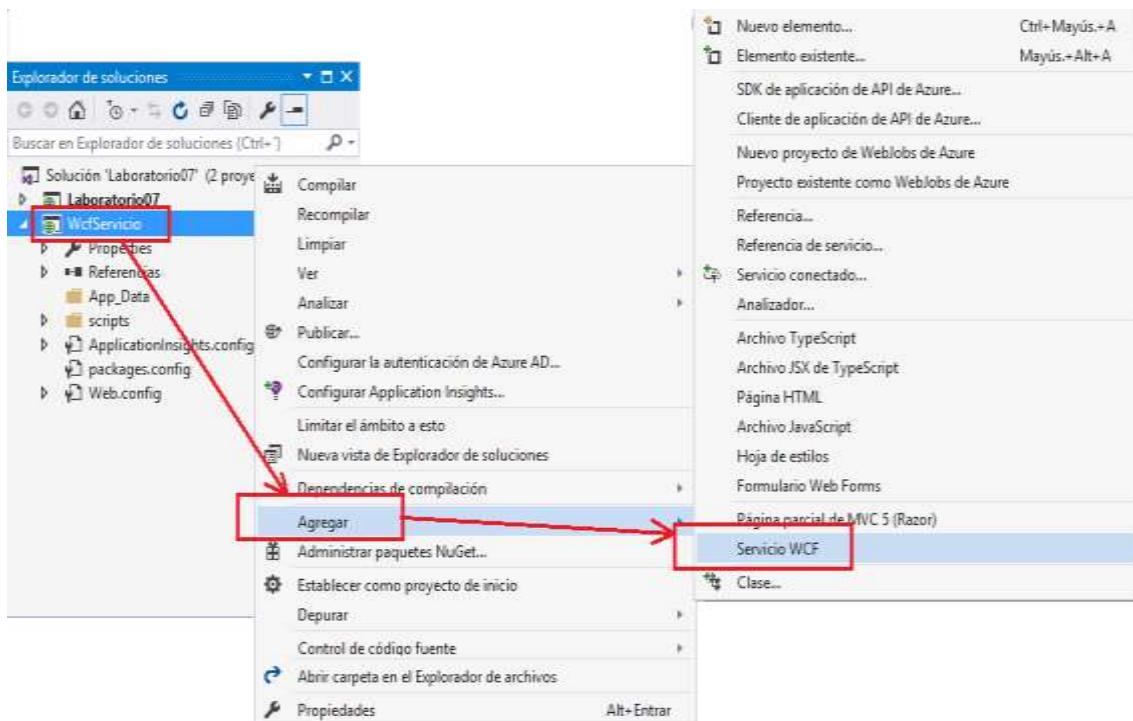
Dentro de la solución, agrega un nuevo proyecto, tal como se muestra



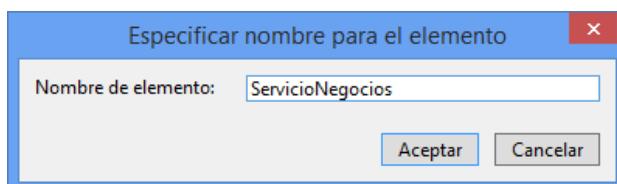
Selecciona el proyecto WCF de tipo Aplicación de servicios WCF, tal como se muestra, asigne un nombre y presiona el botón Aceptar



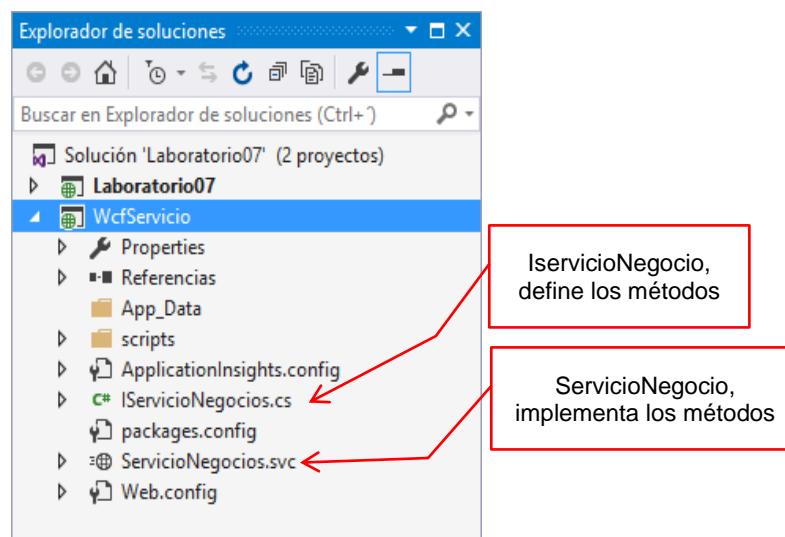
A continuación agregamos un servicio de tipo WCF al proyecto, tal como se muestra



Asignar el nombre del servicio: ServicioNegocios, presiona el botón Aceptar



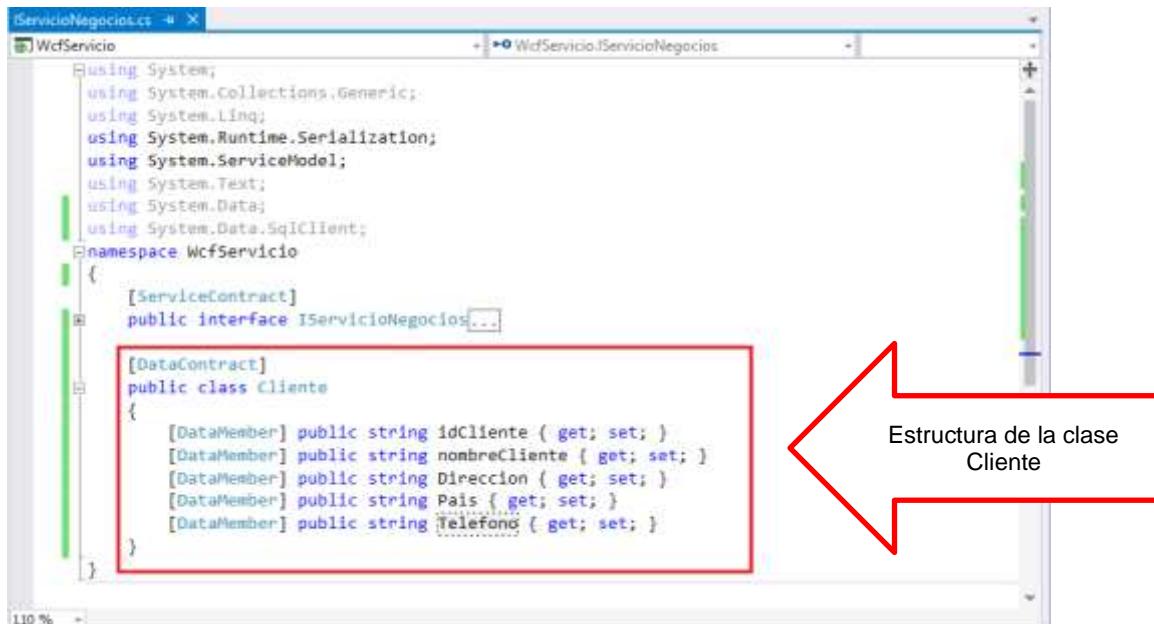
El servicio agregado esta conformado por dos archivos: IservicioNegocio.cs, ServicioNegocios.svc



### Trabajando con la Interface: Definiendo las clases

Como primer paso definimos las clases del modelo de datos: en el archivo IServicioNegocios.cs defina la clase Cliente y la clase Pedido, tal como se muestra.

En el proceso de la definición asigne a cada clase la etiqueta [DataContract] y a sus atributos [DataMember]



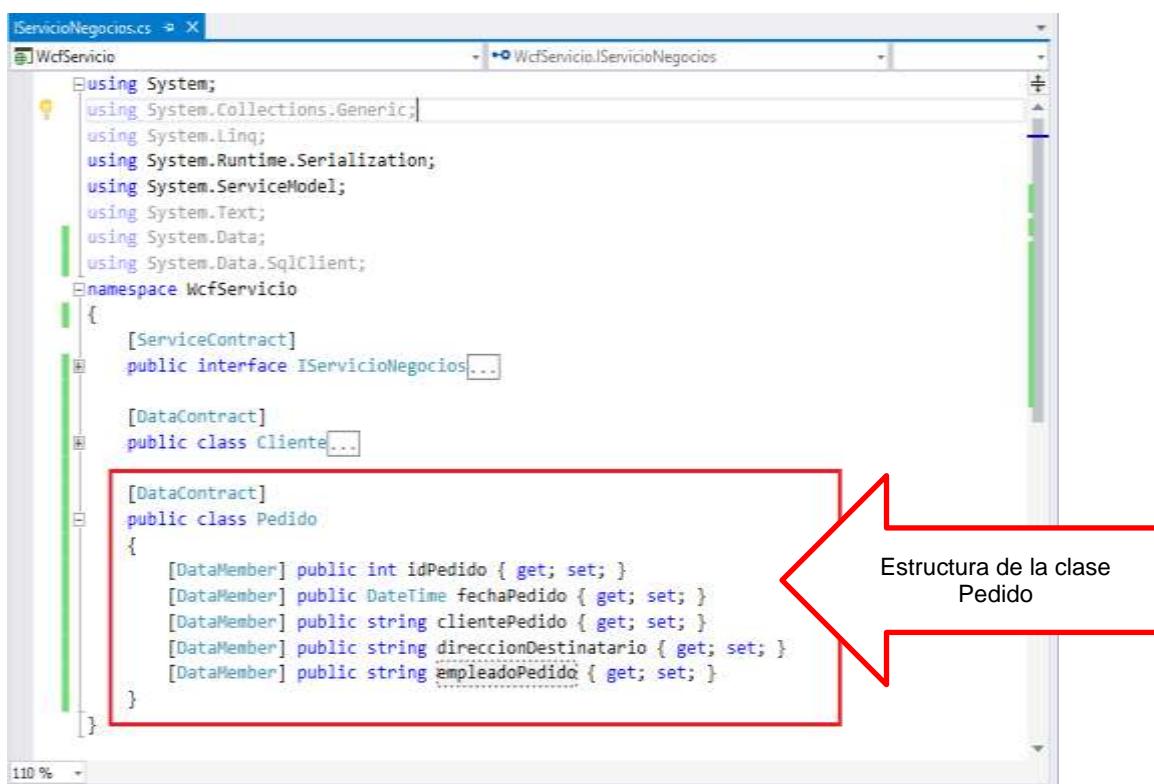
```

namespace WcfServicio
{
    [ServiceContract]
    public interface IServicioNegocios...
}

[DataContract]
public class Cliente
{
    [DataMember] public string idCliente { get; set; }
    [DataMember] public string nombreCliente { get; set; }
    [DataMember] public string Direccion { get; set; }
    [DataMember] public string Pais { get; set; }
    [DataMember] public string Telefono { get; set; }
}

```

Estructura de la clase Cliente



```

namespace WcfServicio
{
    [ServiceContract]
    public interface IServicioNegocios...
}

[DataContract]
public class Cliente...

[DataContract]
public class Pedido
{
    [DataMember] public int idPedido { get; set; }
    [DataMember] public DateTime fechaPedido { get; set; }
    [DataMember] public string clientePedido { get; set; }
    [DataMember] public string direccionDestinatario { get; set; }
    [DataMember] public string empleadoPedido { get; set; }
}

```

Estructura de la clase Pedido

En la Interface IServicioNegocios, defina los métodos a ser consumidos por el Servicio. Recuerda que cada método debe llevar la etiqueta [OperationContract]

```

IServicioNegocios.cs  + X
WcfServicio
  ↳ WcfServicio.ServicioNegocios
    ↳ Clientes()
[ServiceContract]
public interface IServicioNegocios
{
    [OperationContract] List<Cliente> Clientes();
    [OperationContract] List<Cliente> ClientexNombre(string nombre);
    [OperationContract] List<Pedido> PedidoxCiente(string cliente);
    [OperationContract] List<Pedido> PedidoxYear(int y);
    [OperationContract] List<Pedido> PedidoxFechas(DateTime f1, DateTime f2);
}

[DataContract]
public class Cliente{...}

[DataContract]
public class Pedido{...}

```

Definición de los métodos

### Trabajando con ServicioNegocios: Implementando los métodos

A continuación nos situamos en el archivo ServicioNegocios.svc e implementamos los métodos definidos en la interfaz.

```

ServicioNegocios.svc.cs*  + X
WcfServicio
  ↳ WcfServicio.ServicioNegocios
    ↳ DoWork()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfServicio
{
    public class ServicioNegocios : IServicioNegocios
    {
        public void DoWork()
        {
        }
    }
}

```

Hacer click derecho, seleccionar la opción Implementar interfaz

Implementar interfaz

Implementar interfaz de forma explícita

Seleccionando la opción se agrega la lista de los métodos a implementar, tal como se muestra

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfServicio
{
    public class ServicioNegocios : IServicioNegocios
    {
        public List<Cliente> Clientes()...
        public List<Cliente> ClientexNombre(string nombre)...
        public List<Pedido> PedidoxCliente(string cliente)...
        public List<Pedido> PedidoxFechas(DateTime f1, DateTime f2)...
        public List<Pedido> PedidoxYear(int y)...
    }
}
```

Métodos a implementar en ServicioNegocios

Para iniciar el desarrollo, primero agregamos las referencias de las librerías de trabajo, tal como se muestra

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using System.Data;
using System.Data.SqlClient;

namespace WcfServicio
{
    public class ServicioNegocios : IServicioNegocios
    {
        public List<Cliente> Clientes()...
        public List<Cliente> ClientexNombre(string nombre)...
        public List<Pedido> PedidoxCliente(string cliente)...
        public List<Pedido> PedidoxFechas(DateTime f1, DateTime f2)...
        public List<Pedido> PedidoxYear(int y)...
    }
}
```

Referencias a las librerías de datos

Luego definimos la conexión a la base de datos Negocios2017, tal como se muestra.



```

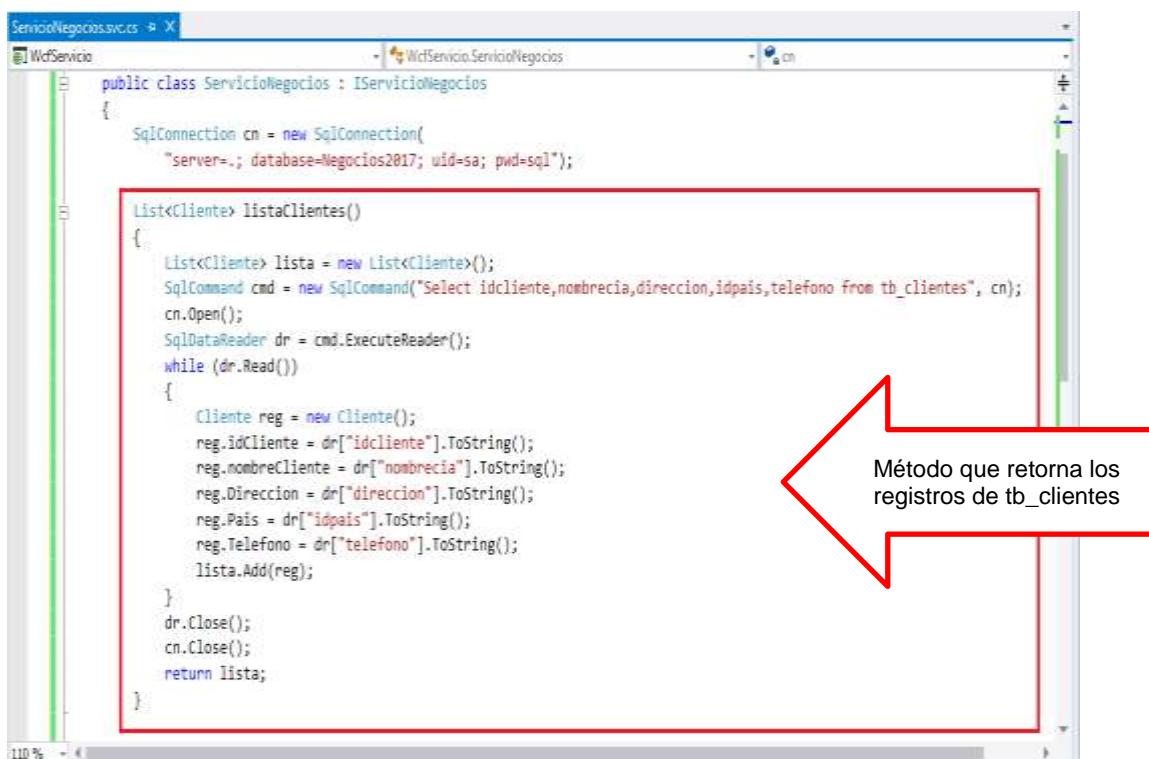
ServicioNegocios.svc.cs  X
WcfService1              WcfService.ServicioNegocios      cn
using ...
namespace WcfService
{
    public class ServicioNegocios : IServicioNegocios
    {
        SqlConnection cn = new SqlConnection(
            "server=.; database=Negocios2017; uid=sa; pwd=sq1");

        public List<Cliente> Clientes()...
        public List<Cliente> ClientexNombre(string nombre)...
        public List<Pedido> PedidoxCliente(string cliente)...
        public List<Pedido> PedidoxFechas(DateTime f1, DateTime f2)...
        public List<Pedido> PedidoxYear(int y)...
    }
}

```

A red box highlights the line of code `SqlConnection cn = new SqlConnection("server=.; database=Negocios2017; uid=sa; pwd=sq1");` which defines the database connection. A red callout points to it with the text "Definición de la conexión de datos".

En el archivo defina un método listaClientes(), el cual retorna, como lista, los registros de la tabla tb\_clientes.



```

ServicioNegocios.svc.cs  X
WcfService1              WcfService.ServicioNegocios      cn
public class ServicioNegocios : IServicioNegocios
{
    SqlConnection cn = new SqlConnection(
        "server=.; database=Negocios2017; uid=sa; pwd=sq1");

    List<Cliente> listaClientes()
    {
        List<Cliente> lista = new List<Cliente>();
        SqlCommand cmd = new SqlCommand("Select idcliente,nombreclie,direccion,idpais,telefono from tb_clientes", cn);
        cn.Open();
        SqlDataReader dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            Cliente reg = new Cliente();
            reg.idCliente = dr["idcliente"].ToString();
            reg.nombreCliente = dr["nombreclie"].ToString();
            reg.Direccion = dr["direccion"].ToString();
            reg.Pais = dr["idpais"].ToString();
            reg.Telefono = dr["telefono"].ToString();
            lista.Add(reg);
        }
        dr.Close();
        cn.Close();
        return lista;
    }
}

```

A large red box highlights the entire implementation of the `listaClientes()` method, which reads data from the `tb\_clientes` table and returns it as a list of `Cliente` objects. A red callout points to it with the text "Método que retorna los registros de tb\_clientes".

Implementa los métodos Clientes() y ClientexNombre(), tal como se muestra

```

namespace WcfServicio
{
    public class ServicioNegocios : IServicioNegocios
    {
        SqlConnection cn = new SqlConnection("server=.; database=Negocios2017; uid=sa; pwd=sql");

        List<Cliente> listaClientes()...
        public List<Cliente> Clientes()
        {
            return listaClientes().ToList();
        }

        public List<Cliente> ClientexNombre(string nombre)
        {
            return listaClientes().Where(c => c.nombreCliente.StartsWith(nombre)).ToList();
        }

        public List<Pedido> PedidoxCliente(string cliente)...
        public List<Pedido> PedidoxFechas(DateTime f1, DateTime f2)...
        public List<Pedido> PedidoxYear(int y)...
    }
}

```

Método que retorna la lista de clientes

Método que retorna la lista de clientes filtrando por las iniciales de su nombre

En el archivo defina un método listaPedido(), el cual retorna, como lista, los registros de la tabla tb\_pedidoscabe

```

List<Pedido> listaPedidos()
{
    List<Pedido> lista = new List<Pedido>();
    SqlCommand cmd = new SqlCommand(
        "Select idpedido, FechaPedido, NombreCia, DireccionDestinatario, apeEmpleado from tb_clientes c " +
        "join tb_pedidoscabe p on c.idcliente=p.idcliente join tb_empleados e on e.idempleado=p.idempleado", cn);
    cn.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
    {
        Pedido reg = new Pedido();
        reg.idPedido =(int)dr["idpedido"];
        reg.fechaPedido =(DateTime)dr["FechaPedido"];
        reg.direccionDestinatario = dr["direccionDestinatario"].ToString();
        reg.empleadoPedido = dr["apeEmpleado"].ToString();
        lista.Add(reg);
    }
    dr.Close();
    cn.Close();
    return lista;
}

public List<Pedido> PedidoxCliente(string cliente)...

public List<Pedido> PedidoxFechas(DateTime f1, DateTime f2)...

public List<Pedido> PedidoxYear(int y)...
}

```

Definir el método que retorna los registros

Implementa los métodos PedidoxCiente(), PedidoxFechas() y PedidoxYear(), tal como se muestra

```

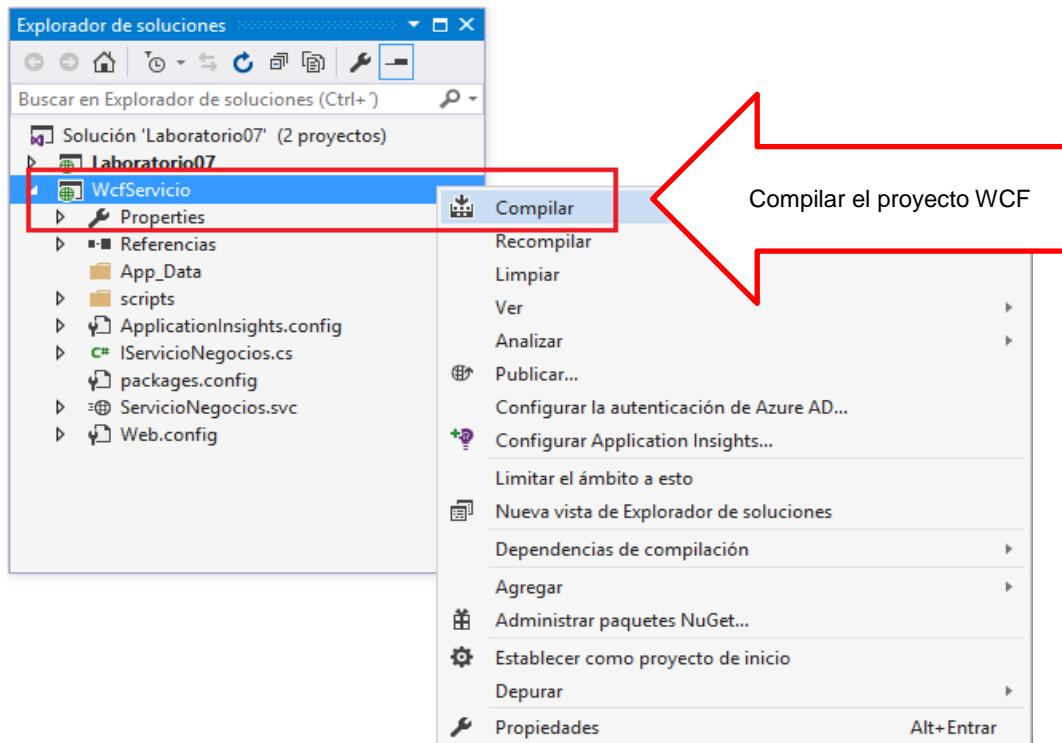
ServicioNegocios.svc.cs  WcfService  WcfService.ServicioNegocios  PedidoxYear(int y)

public class ServicioNegocios : IServicioNegocios
{
    SqlConnection cn = new SqlConnection("server=.; database=Negocios2017; uid=sa; pwd=sq1");

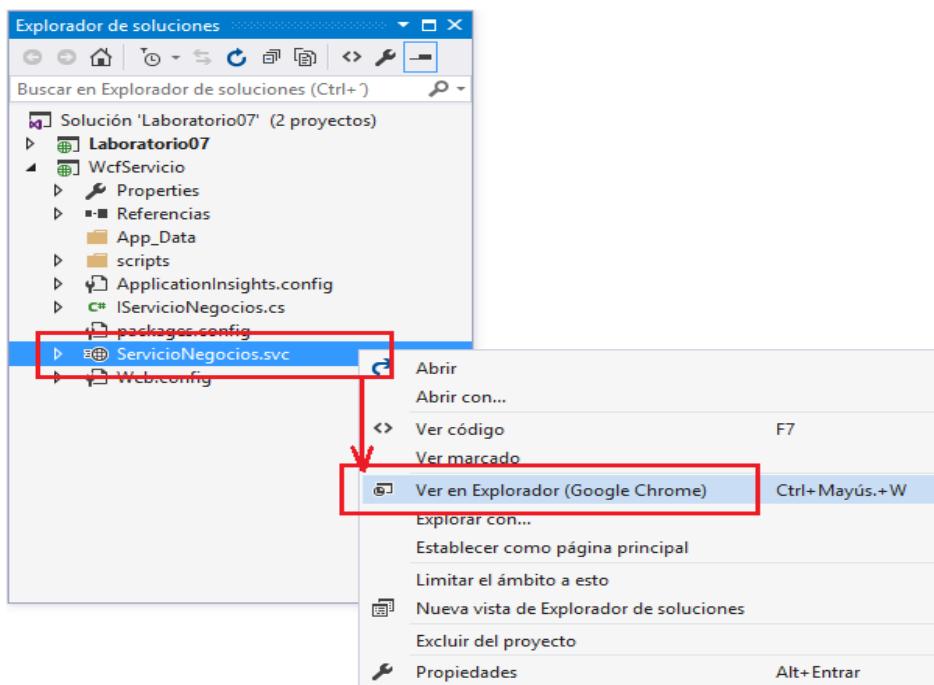
    List<Cliente> listaClientes()...
    public List<Cliente> Clientes()...
    public List<Cliente> ClienteNombre(string nombre)...
    List<Pedido> listaPedidos()...
    public List<Pedido> PedidoxCiente(string cliente)
    {
        return listaPedidos().Where(p => p.clientePedido == cliente).ToList();
    }
    public List<Pedido> PedidoxFechas(DateTime f1, DateTime f2)
    {
        return listaPedidos().Where(p => p.fechaPedido >= f1 && p.fechaPedido <= f2).ToList();
    }
    public List<Pedido> PedidoxYear(int y)
    {
        return listaPedidos().Where(p => p.fechaPedido.Year == y).ToList();
    }
}

```

Para finalizar, compilamos el proyecto WCF, tal como se muestra



Para consumir el servicio, debemos ejecutarlo en un Explorador, el cual nos dará una dirección URL donde se ejecuta el servicio.

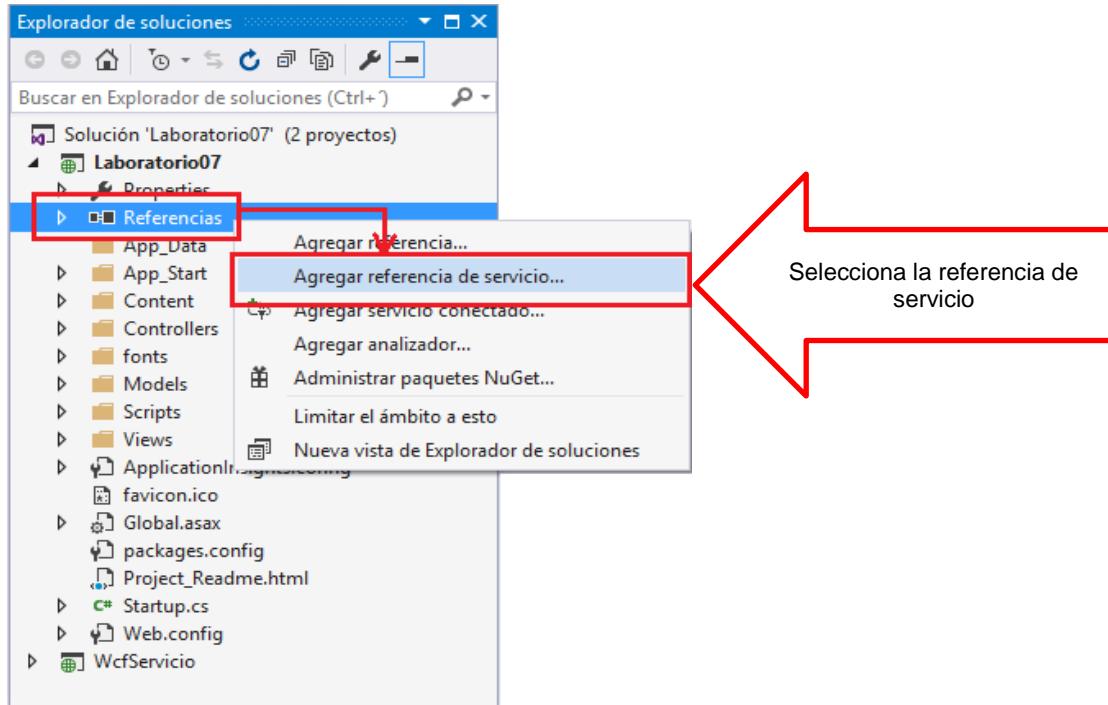


Copiar la dirección URL, para consumir el servicio

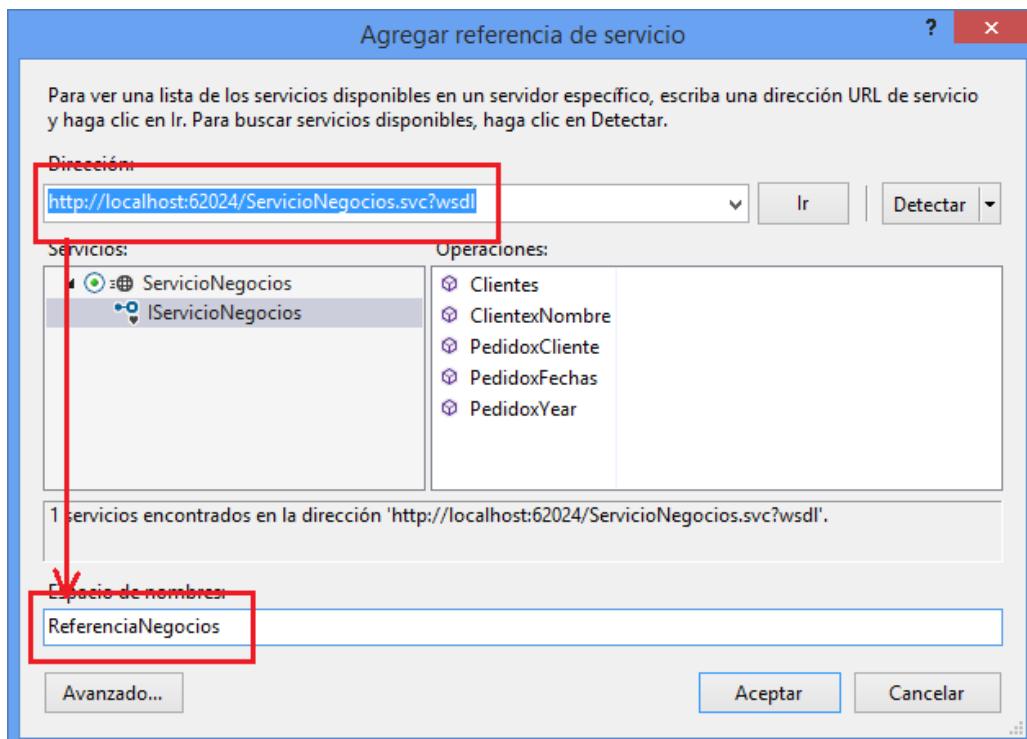


## Trabajando con el proyecto ASP.NET MVC

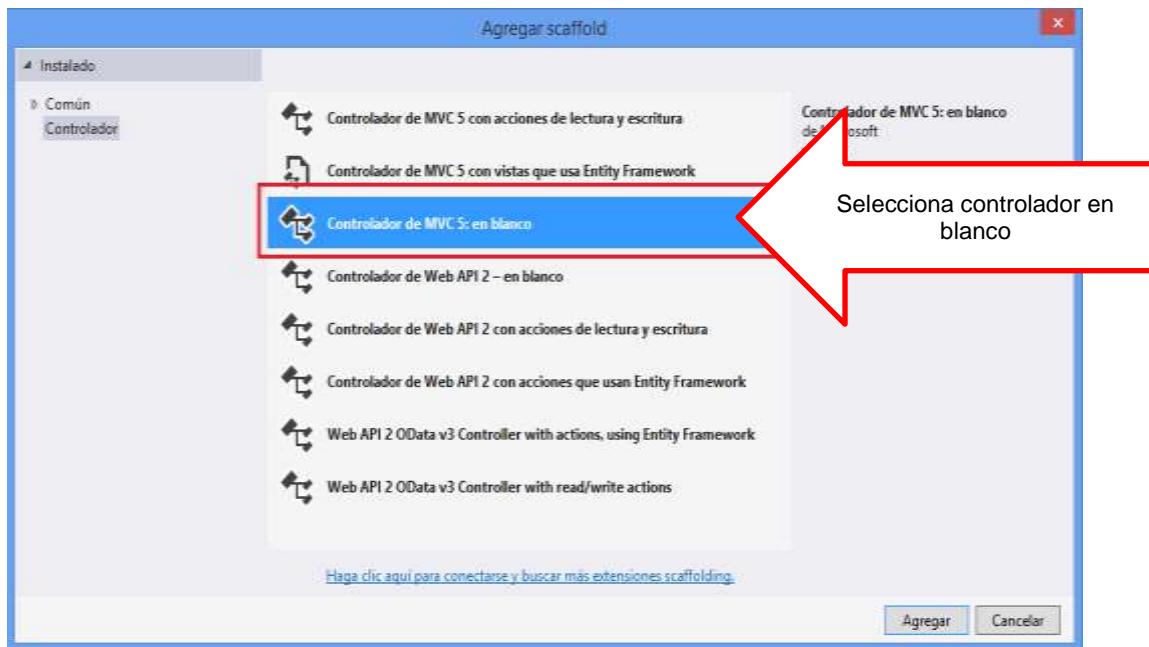
En el proyecto MVC, agregar una referencia de servicio, para consumir el servicio del proyecto WCF, tal como se muestra



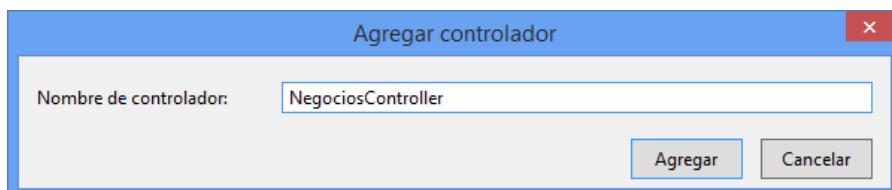
En la ventana, pegar la dirección del servicio y presiona el botón Ir, donde se visualiza los métodos definidos. Asigne un nombre a la referencia, tal como se muestra.



A continuación agregamos un controlador al proyecto



Asigne el nombre del Controlador, tal como se muestra.

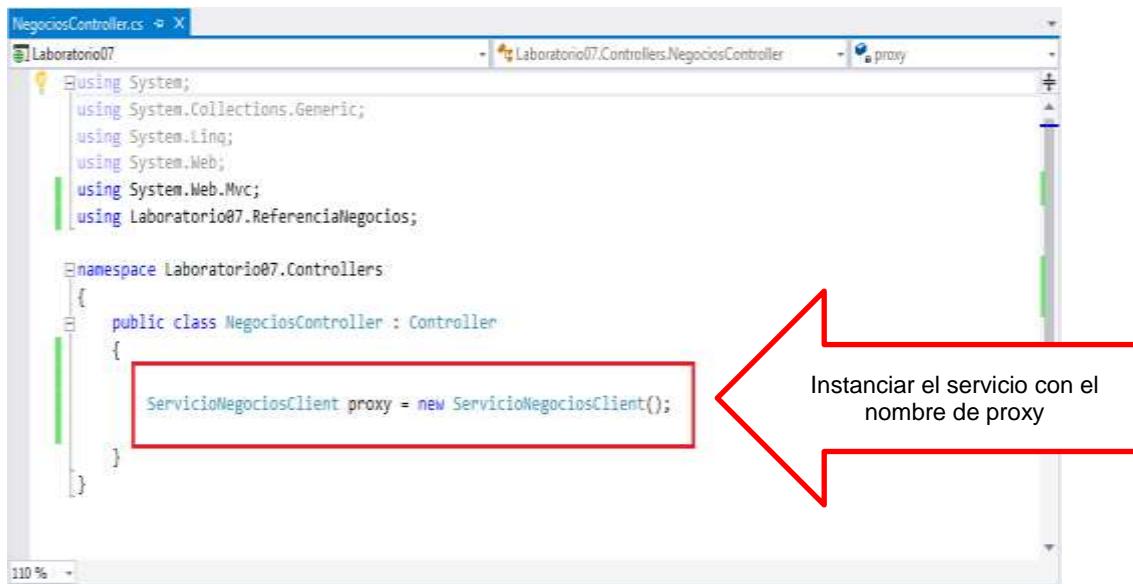


Como primer paso, importar la referencia de Negocio, tal como se muestra

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Laboratorio07.ReferenciaNegocios;

namespace Laboratorio07.Controllers
{
    public class NegociosController : Controller
    {
    }
}
```

Instanciar la referencia con el nombre de proxy.



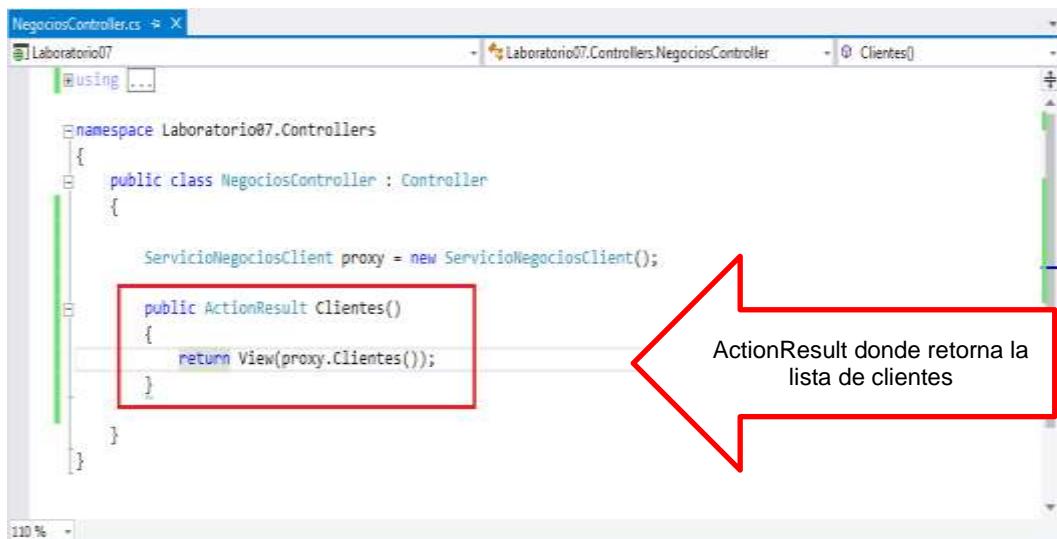
```
NegociosController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Laboratorio07.ReferenciaNegocios;

namespace Laboratorio07.Controllers
{
    public class NegociosController : Controller
    {
        ServicioNegociosClient proxy = new ServicioNegociosClient();
    }
}
```

A red box highlights the line of code: `ServicioNegociosClient proxy = new ServicioNegociosClient();`. A red arrow points from this box to the text "Instanciar el servicio con el nombre de proxy".

### Trabajando con el ActionResult Clientes()

Defina el ActionResult Cliente(), el cual retorna la lista de los clientes.

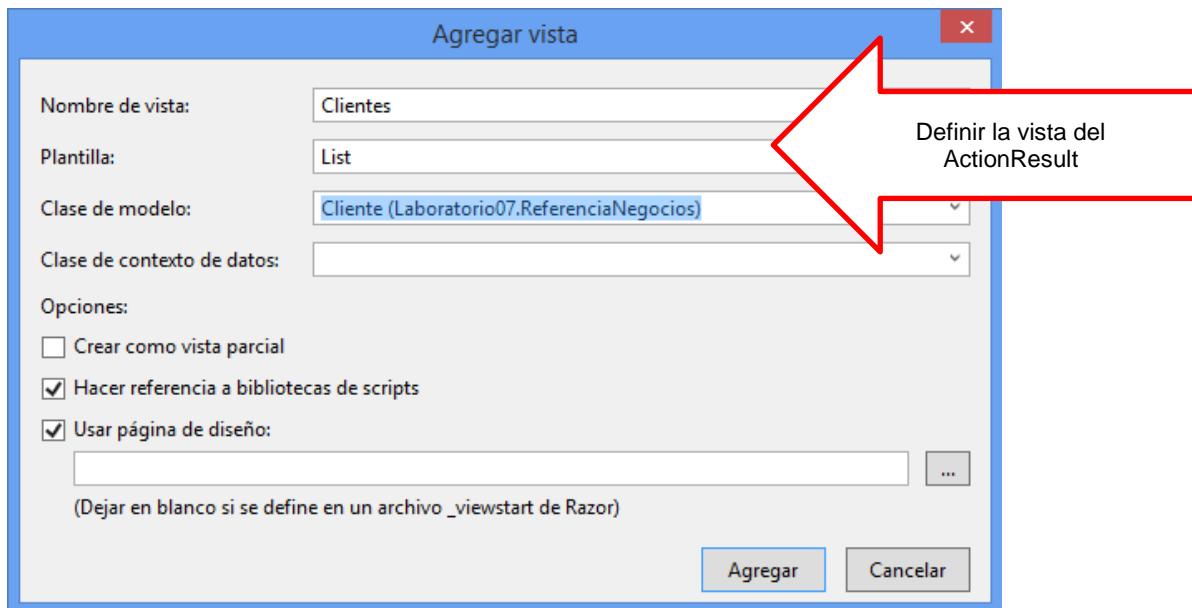


```
NegociosController.cs
using ...
namespace Laboratorio07.Controllers
{
    public class NegociosController : Controller
    {
        ServicioNegociosClient proxy = new ServicioNegociosClient();

        public ActionResult Clientes()
        {
            return View(proxy.Clientes());
        }
    }
}
```

A red box highlights the line of code: `return View(proxy.Clientes());`. A red arrow points from this box to the text "ActionResult donde retorna la lista de clientes".

Definido el ActionResult, agregar una vista, cuya plantilla será de tipo List y su clase Cliente (WCF), tal como se muestra



Ejecuta la Vista, visualizando los resultados de la consulta

Dirección	País	Teléfono	idCliente	nombreCliente	
Lima	005	5662321	A6003	Juan Garcia Valderrama	Edit   Details   Delete
Av. Lima, 46533	015	9556655	AA456	Aaron Alvarez Garcia	Edit   Details   Delete
Jr. Cusco 1222	001	5663232	AB123	Abelardo Garcia	Edit   Details   Delete
Obere Str. 57	002	030-0174321	ALFKI	Alfredo Fiterkista	Edit   Details   Delete
Avda. de la Constitucion 2222	005	(5) 555-4729	ANATR	Ana Trujillo Emparedados y helados	Edit   Details   Delete
Mataderos 2312	007	(5) 555-3932	ANTON	Antonio Moreno Taqueria	Edit   Details   Delete
120 Hanover Sq.	004	(71) 555-7788	AROUT	Around the Horn	Edit   Details   Delete
Berguvägen 8	005	0521-12 34 65	BERGS	Berglunds snabbköp	Edit   Details   Delete

### Trabajando con el ActionResult ClientesxNombre()

Defina el ActionResult ClientesxNombre(), el cual retorna la lista de los clientes por las iniciales del nombre, tal como se muestra.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Laboratorio07;
using Laboratorio07.Controllers;
using Laboratorio07.ServicioNegociosClient;
using System.ServiceModel;

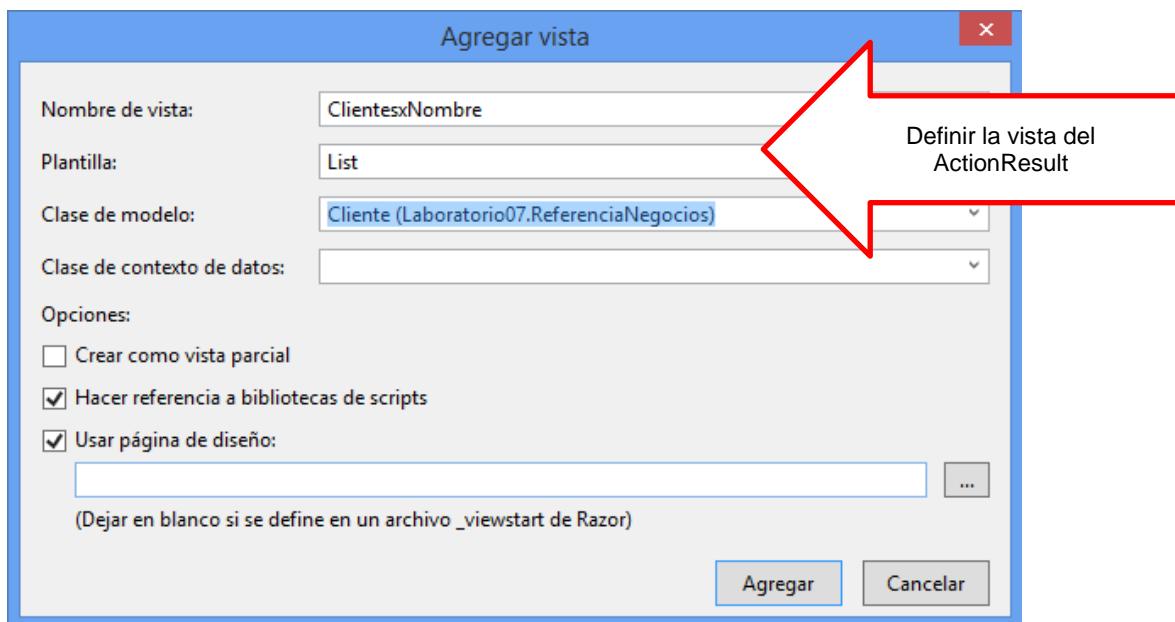
namespace Laboratorio07.Controllers
{
    public class NegociosController : Controller
    {
        ServicioNegociosClient proxy = new ServicioNegociosClient();

        public ActionResult Clientes()
        {
            return View();
        }

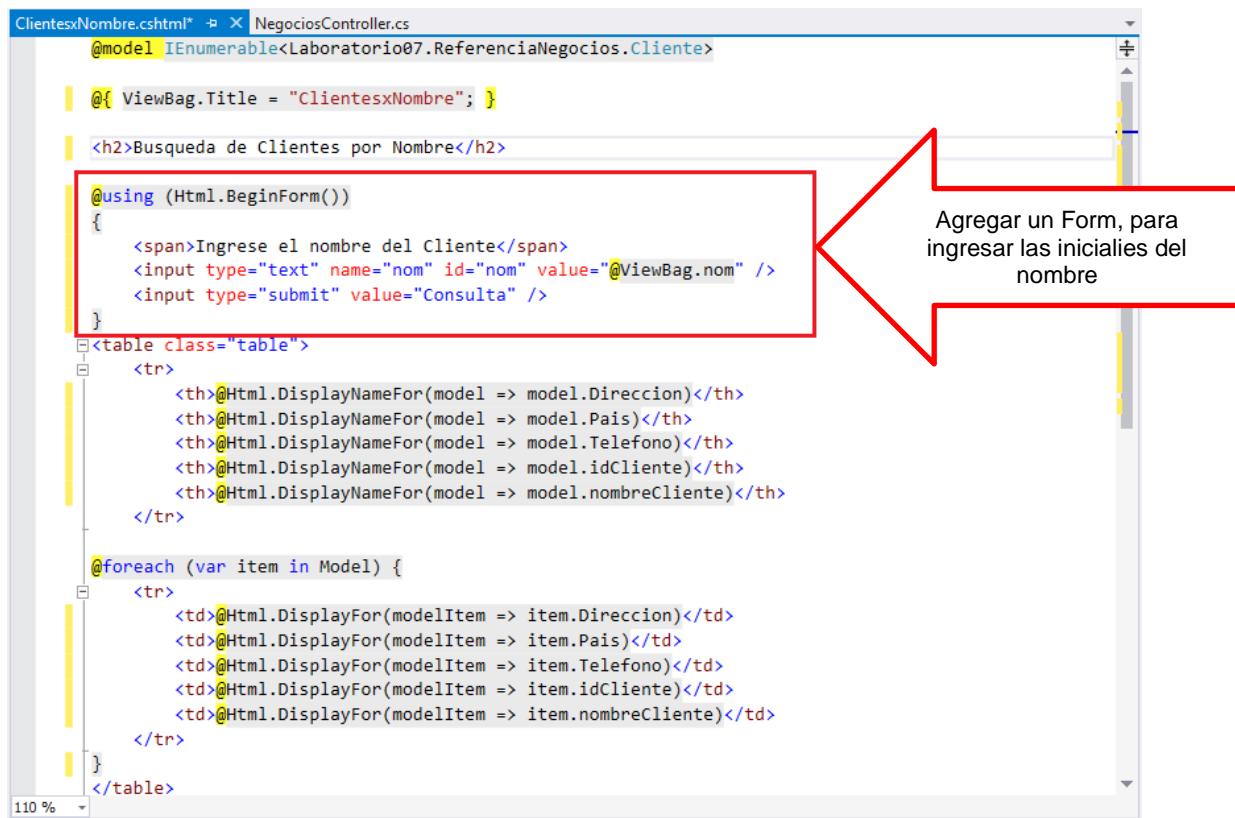
        public ActionResult ClientesxNombre(string nom)
        {
            ViewBag.nom = nom;
            return View(proxy.ClientexNombre(nom));
        }
    }
}

```

Definido el ActionResult, agregar una vista, cuya plantilla será de tipo List y su clase Cliente (WCF), tal como se muestra



Modifica la estructura de la Vista, para ingresar el nombre del cliente desde un control tipo text, donde al ejecutar la Vista, visualizamos los clientes del filtro.



```

ClientesxNombre.cshtml"  ➔ × NegociosController.cs
@model IEnumerable<Laboratorio07.ReferenciaNegocios.Cliente>

@{ ViewBag.Title = "ClientesxNombre"; }



## Busqueda de Clientes por Nombre



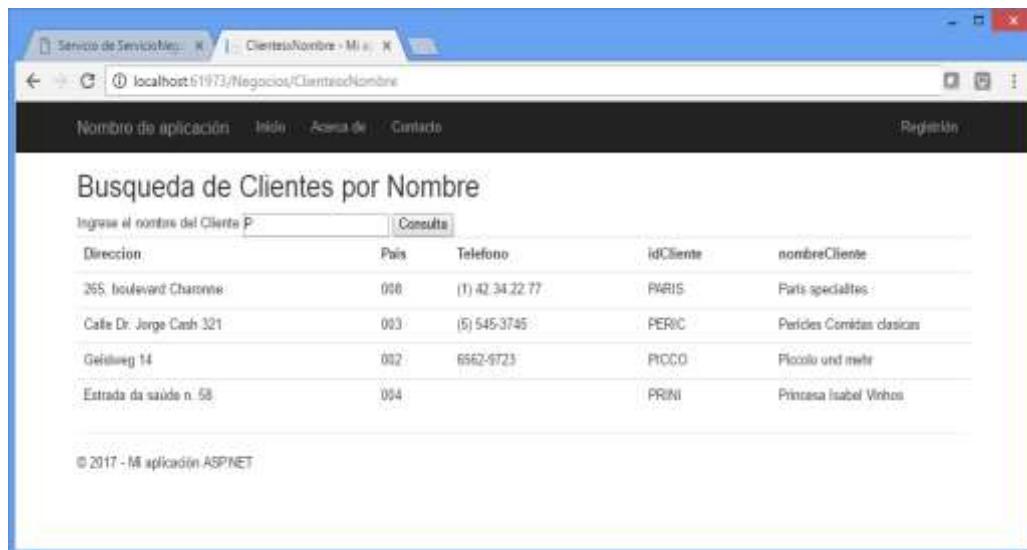
@using (Html.BeginForm())
{
    <span>Ingrese el nombre del Cliente</span>
    <input type="text" name="nom" id="nom" value="@ViewBag.nom" />
    <input type="submit" value="Consulta" />
}



| @Html.DisplayNameFor(model => model.Direccion) | @Html.DisplayNameFor(model => model.Pais) | @Html.DisplayNameFor(model => model.Telefono) | @Html.DisplayNameFor(model => model.idCliente) | @Html.DisplayNameFor(model => model.nombreCliente) |
|------------------------------------------------|-------------------------------------------|-----------------------------------------------|------------------------------------------------|----------------------------------------------------|
|------------------------------------------------|-------------------------------------------|-----------------------------------------------|------------------------------------------------|----------------------------------------------------|


```

Ejecuta la Vista, ingrese las iniciales del nombre, al presionar el botón Consulta, se visualiza los registros filtrados.



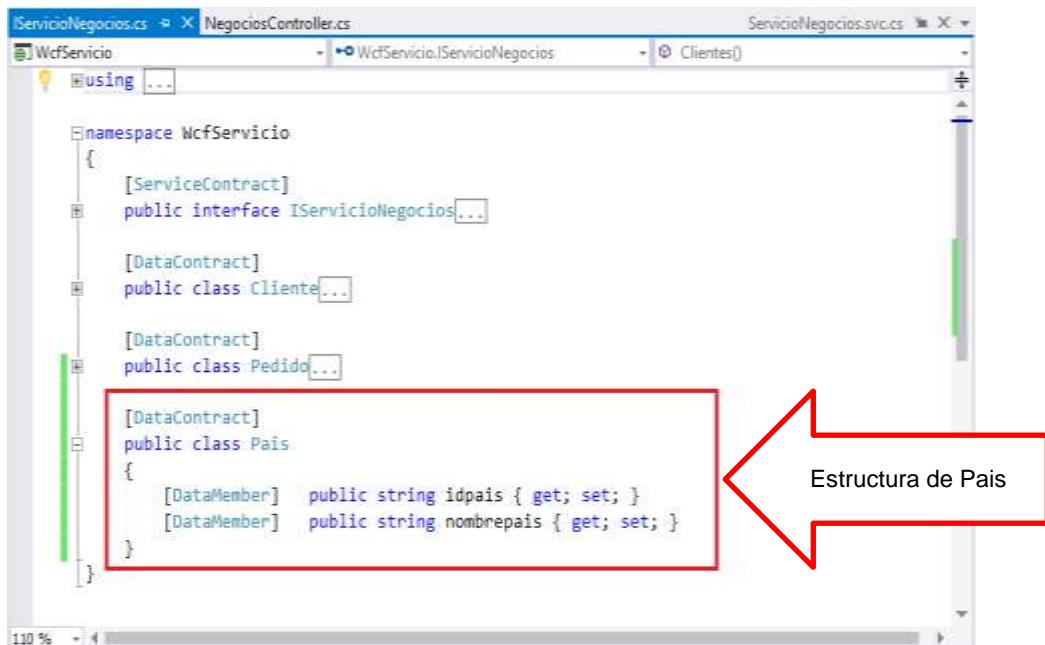
## Laboratorio 7.2

### Implementando CRUD en ASP.NET MVC consumiendo un servicio WCF

Implemente un proyecto ASP.NET MVC donde permita realizar las operaciones de actualización de datos consumiendo un servicio WCF.

#### SOLUCION

Para el desarrollo del proceso de actualización a la tabla tb\_clientes, agregamos en Interface del proyecto WCF, un DataContract llamado País, tal como se muestra



```

namespace WcfServicio
{
    [ServiceContract]
    public interface IServicioNegocios
    {
        [OperationContract]
        List<Cliente> Clientes();

        [OperationContract]
        List<Cliente> ClientexNombre(string nombre);

        [OperationContract]
        List<Pedido> PedidoxCiente(string cliente);

        [OperationContract]
        List<Pedido> PedidoxYear(int y);

        [OperationContract]
        List<Pedido> PedidoxFechas(DateTime f1, DateTime f2);

        [OperationContract]
        List<Pais> Paises();

        [OperationContract]
        string AgregarCliente(Cliente reg);

        [OperationContract]
        string ActualizarCliente(Cliente reg);

        Cliente DetalleCliente(string id);
    }

    [DataContract]
    public class Cliente
    {
        [DataMember]
        public string nombre { get; set; }

        [DataMember]
        public string apellido { get; set; }

        [DataMember]
        public string dni { get; set; }

        [DataMember]
        public string email { get; set; }

        [DataMember]
        public string telefono { get; set; }

        [DataMember]
        public string direccion { get; set; }
    }

    [DataContract]
    public class Pedido
    {
        [DataMember]
        public string idpedido { get; set; }

        [DataMember]
        public string idcliente { get; set; }

        [DataMember]
        public string idproducto { get; set; }

        [DataMember]
        public decimal cantidad { get; set; }

        [DataMember]
        public decimal precio { get; set; }
    }

    [DataContract]
    public class Pais
    {
        [DataMember]
        public string idpais { get; set; }

        [DataMember]
        public string nombrepais { get; set; }
    }
}

```

Agregar a la Interface los nuevos métodos, tal como se muestra



```

namespace WcfServicio
{
    [ServiceContract]
    public interface IServicioNegocios
    {
        [OperationContract]
        List<Cliente> Clientes();

        [OperationContract]
        List<Cliente> ClientexNombre(string nombre);

        [OperationContract]
        List<Pedido> PedidoxCiente(string cliente);

        [OperationContract]
        List<Pedido> PedidoxYear(int y);

        [OperationContract]
        List<Pedido> PedidoxFechas(DateTime f1, DateTime f2);

        [OperationContract]
        List<Pais> Paises();

        [OperationContract]
        string AgregarCliente(Cliente reg);

        [OperationContract]
        string ActualizarCliente(Cliente reg);

        Cliente DetalleCliente(string id);

        [OperationContract]
        void EliminarCliente(string id);
    }

    [DataContract]
    public class Cliente
    {
        [DataMember]
        public string nombre { get; set; }

        [DataMember]
        public string apellido { get; set; }

        [DataMember]
        public string dni { get; set; }

        [DataMember]
        public string email { get; set; }

        [DataMember]
        public string telefono { get; set; }

        [DataMember]
        public string direccion { get; set; }
    }

    [DataContract]
    public class Pedido
    {
        [DataMember]
        public string idpedido { get; set; }

        [DataMember]
        public string idcliente { get; set; }

        [DataMember]
        public string idproducto { get; set; }

        [DataMember]
        public decimal cantidad { get; set; }

        [DataMember]
        public decimal precio { get; set; }
    }

    [DataContract]
    public class Pais
    {
        [DataMember]
        public string idpais { get; set; }

        [DataMember]
        public string nombrepais { get; set; }
    }
}

```

A continuación regresamos al archivo ServicioNegocios.svc e implementamos la interface, tal como se muestra

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.SqlClient;

namespace WcfServicio
{
    public class ServicioNegocios : IServicioNegocios
    {
        SqlConnection cn = new SqlConnection("...");

        List<Cliente> listaClientes()...
        public List<Cliente> Clientes()...
        public List<Cliente> ClienteNombre(string nombre)...
        List<Pedido> listaPedidos()...
        public List<Pedido> PedidoxCliente(string cliente)...
        public List<Pedido> PedidoxFechas(DateTime f1, DateTime f2)...
        public List<Pedido> PedidoxYear(int y)
        {
            return listaPedidos().Where(p => p.fechaPedido.Year == y).ToList();
        }
    }
}

```

Implementar la interfaz

C52035 'ServicioNegocios' no implementa el miembro de interfaz 'IServicioNegocios.Paises()'

Vista previa de cambios

Conseguir todas las repeticiones de Documento | Proyecto | Solución

A continuación implementamos el método Paises, el cual retorna la lista de los países

```

public List<Pais> Paises()
{
    List<Pais> lista = new List<Pais>();
    SqlCommand cmd = new SqlCommand("Select idpais,nombrePais from tb_paises", cn);
    cn.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
    {
        Pais reg = new Pais();
        reg.idpais = dr["idpais"].ToString();
        reg.nombrepais = dr["nombrepais"].ToString();
        lista.Add(reg);
    }
    dr.Close();
    cn.Close();
    return lista;
}

public string AgregarCliente(Cliente reg)...
public string ActualizarCliente(Cliente reg)...
public Cliente DetalleCliente(string id)...
}

```

Método que retorna la lista de países

Implementa el método AgregarCliente, el cual ejecuta el comando SQL, donde insertar un registro a la tabla tb\_clientes, tal como se muestra

```

public class NegociosController : WcfService
{
    public List<Pais> Paises()
    {
        ...
    }

    public string AgregarCliente(Cliente reg)
    {
        string msg = "";
        cn.Open();
        try
        {
            SqlCommand cmd = new SqlCommand(
                "Insert tb_clientes Values(@cod,@nom,@dir,@idpais,@fono)", cn);
            cmd.Parameters.AddWithValue("@cod", reg.idCliente);
            cmd.Parameters.AddWithValue("@nom", reg.nombreCliente);
            cmd.Parameters.AddWithValue("@dir", reg.Direccion);
            cmd.Parameters.AddWithValue("@idpais", reg.Pais);
            cmd.Parameters.AddWithValue("@fono", reg.Telefono);

            int q = cmd.ExecuteNonQuery();
            msg = q.ToString() + " registro agregado";
        }
        catch (SqlException ex) { msg = ex.Message; }
        finally { cn.Close(); }
        return msg;
    }

    public string ActualizarCliente(Cliente reg)
    {
        ...
    }

    public Cliente DetalleCliente(string id)
    {
        ...
    }
}

```

Implementa el método ActualizarCliente, el cual ejecuta el comando SQL, donde actualiza un registro a la tabla tb\_clientes por su campo idcliente, tal como se muestra

```

public class NegociosController : WcfService
{
    public List<Pais> Paises()
    {
        ...
    }

    public string AgregarCliente(Cliente reg)
    {
        ...
    }

    public string ActualizarCliente(Cliente reg)
    {
        string msg = "";
        cn.Open();
        try
        {
            SqlCommand cmd = new SqlCommand(
                "Update tb_clientes Set nombre=@nom,Direccion=@dir,idpais=@idpais," +
                "Telefono=@fono Where idcliente=@cod", cn);
            cmd.Parameters.AddWithValue("@cod", reg.idCliente);
            cmd.Parameters.AddWithValue("@nom", reg.nombreCliente);
            cmd.Parameters.AddWithValue("@dir", reg.Direccion);
            cmd.Parameters.AddWithValue("@idpais", reg.Pais);
            cmd.Parameters.AddWithValue("@fono", reg.Telefono);

            int q = cmd.ExecuteNonQuery();
            msg = q.ToString() + " registro actualizado";
        }
        catch (SqlException ex) { msg = ex.Message; }
        finally { cn.Close(); }
        return msg;
    }

    public Cliente DetalleCliente(string id)
    {
        ...
    }
}

```

Implementa el método DetalleCliente, el cual ejecuta el comando SQL, donde busca un registro a la tabla tb\_clientes por su campo idcliente, tal como se muestra

```

ServicioNegocios.svc.cs  NegociosController.cs
WcfService1                               WcfService1.ServicioNegocios
public class WcfService1 : WcfService1.ServicioNegocios
{
    public List<Pais> Paises()
    {
        return null;
    }

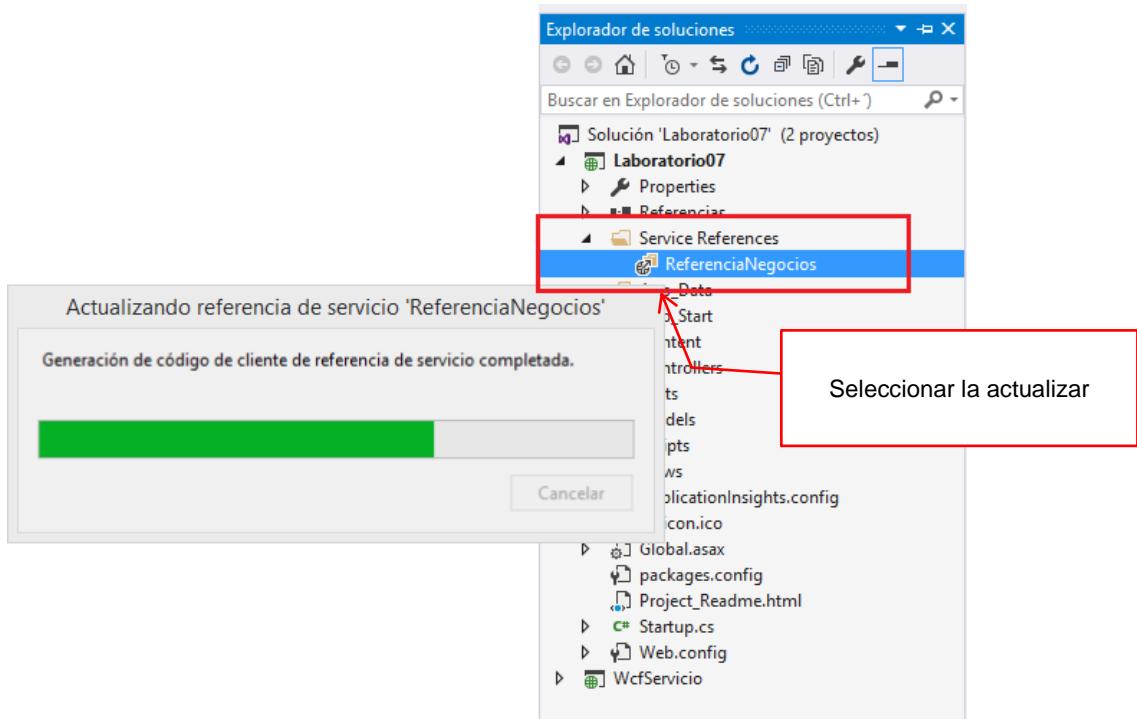
    public string AgregarCliente(Cliente reg)
    {
        return null;
    }

    public string ActualizarCliente(Cliente reg)
    {
        return null;
    }

    public Cliente DetalleCliente(string id)
    {
        SqlCommand cmd = new SqlCommand("Select top 1 * from tb_clientes Where idcliente=@cod", cn);
        cmd.Parameters.AddWithValue("@cod", id);
        cn.Open();
        SqlDataReader dr = cmd.ExecuteReader();
        Cliente reg = new Cliente();
        if(dr.Read())
        {
            reg.idCliente = dr["idcliente"].ToString();
            reg.nombreCliente = dr["nombrecia"].ToString();
            reg.Direccion = dr["direccion"].ToString();
            reg.Pais = dr["idpais"].ToString();
            reg.Telefono = dr["telefono"].ToString();
        }
        dr.Close();
        cn.Close();
        return reg;
    }
}

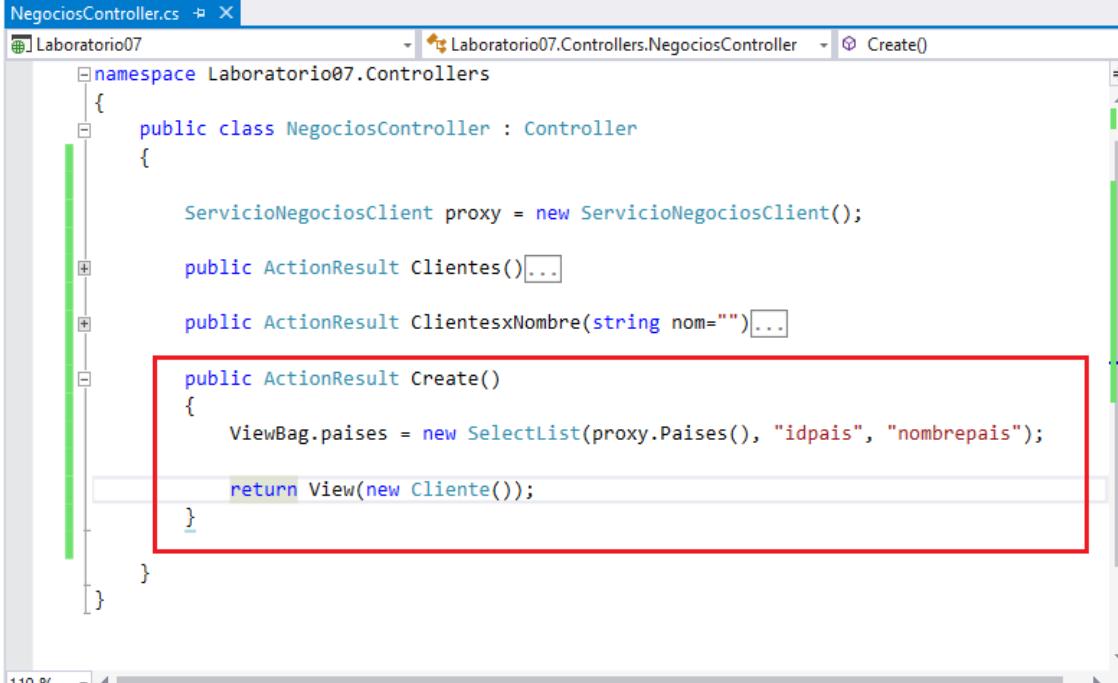
```

Después de compilar la solución, actualizamos la referenciaNegocios, tal como se muestra.



## Trabajando con el Action Create

En el controlador defina el método Create (Get) el cual envía los datos de un nuevo Cliente, y la lista de los registros de países en el ViewBag.paises, tal como se muestra



```

NegociosController.cs  X
Laboratorio07          Laboratorio07.Controllers.NegociosController  Create()
namespace Laboratorio07.Controllers
{
    public class NegociosController : Controller
    {
        ServicioNegociosClient proxy = new ServicioNegociosClient();

        public ActionResult Clientes()...
        public ActionResult ClientesxNombre(string nom="")
        {
            ViewBag.paises = new SelectList(proxy.Paises(), "idpais", "nombrepais");
            return View(new Cliente());
        }
    }
}

```

Defina el método Post, donde ejecutamos el método para agregar un registro a la tabla tb\_clientes desde el método AgregarCliente()



```

NegociosController.cs  X
Laboratorio07          Laboratorio07.Controllers.NegociosController  proxy
public class NegociosController : Controller
{
    ServicioNegociosClient proxy = new ServicioNegociosClient();

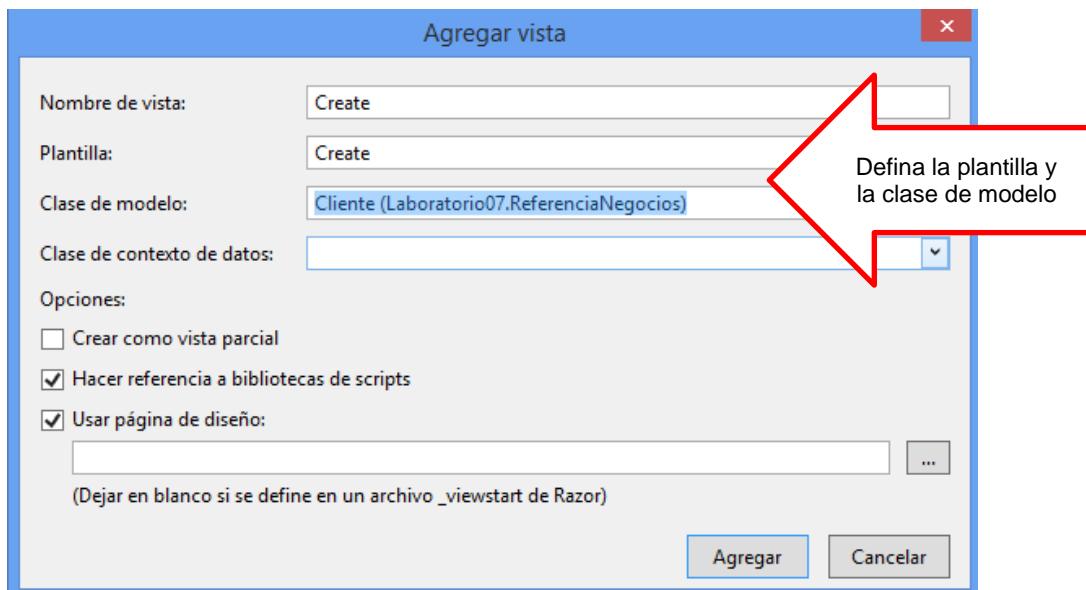
    public ActionResult Clientes()...
    public ActionResult ClientesxNombre(string nom="")...
    public ActionResult Create()...

    [HttpPost]
    public ActionResult Create(Cliente reg)
    {
        string msg = proxy.AgregarCliente(reg);
        ViewBag.msg = msg;

        return RedirectToAction("Clientes");
    }
}

```

Defina la Vista del ActionResult: plantilla de tipo Create y la clase de modelo Cliente, tal como se muestra



En la vista modificar el contenido del campo País, agregar un DropDownList para listar los registros de países, tal como se muestra

```

Create.cshtml  NegociosController.cs

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Cliente</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">...</div>

        <div class="form-group">
            @Html.Label("Pais", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownList("Pais", ViewBag.paises, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Pais, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">...</div>
        <div class="form-group">...</div>
        <div class="form-group">...</div>
        <div class="form-group">...</div>
    </div>

    <div>@Html.ActionLink("Back to List", "Index")</div>

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

```

Ejecuta la Vista, ingresa los datos, al presionar el botón Create, se ejecuta el proceso y se visualiza en la lista los clientes y el nuevo cliente agregado.

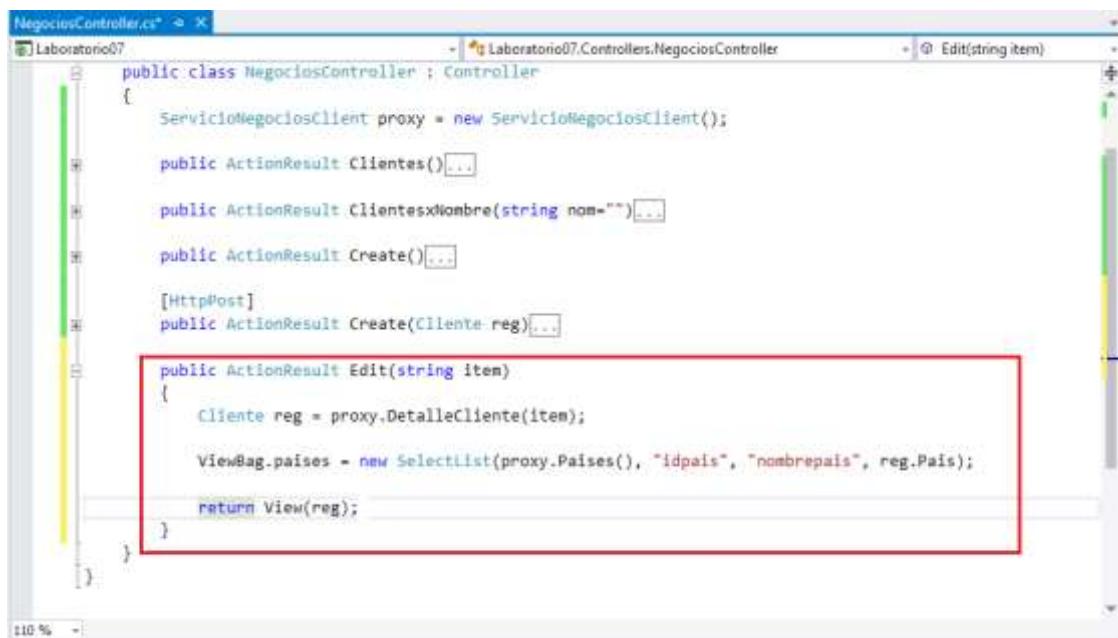
The screenshot shows a web browser window titled 'Create - Mi aplicación'. The URL in the address bar is 'localhost:61973/Negocios/Create'. The page has a header with links for 'Nombre de aplicación', 'Inicio', 'Acerca de', 'Contacto', 'Registrarse', and 'Iniciar sesión'. Below the header, the word 'Create' is displayed in bold. Underneath it, the word 'Cliente' is shown. There are five input fields: 'Dirección' (Lince), 'País' (Peru), 'Teléfono' (456123), 'idCliente' (A0123), and 'nombreCliente' (Luis Alberto). A 'Create' button is located below the input fields. At the bottom of the form, there are links for 'Back to List' and '© 2017 - Mi aplicación ASP.NET'.

The screenshot shows a web browser window titled 'Cuentas - Mi aplicación'. The URL in the address bar is 'localhost:61973/Negocios/Cuentas'. The page has a header with links for 'Nombre de aplicación', 'Inicio', 'Acerca de', 'Contacto', 'Registrarse', and 'Iniciar sesión'. Below the header, the word 'Cuentas' is displayed in bold. Underneath it, the word 'Crear Nuevo' is shown. A table lists eight clients with the following data:

Dirección	País	Teléfono	IdCliente	nombreCliente	Actions
Lima	005	5662321	A0003	Juan Garcia Valdivia	Edit   Details   Delete
Lince	001	456123	A0123	Luis Alberto	Edit   Details   Delete
Lima	003	5789999	A1234	Juan	Edit   Details   Delete
Av. Lima 46633	005	9556655	AA455	Aaron Alvarez Garcia	Edit   Details   Delete
Jr. Cusco 1222	001	5663232	AB123	Abelardo Garcia	Edit   Details   Delete
Obere Str. 57.	002	030-0074321	ALFKI	Alfreds Futterkiste	Edit   Details   Delete
Avenida de la Constitución 2222	005	(5) 555-4729	ANATR	Ana Trujillo Emparedados y helados	Edit   Details   Delete

### Trabajando con el Action Edit

En el controlador defina el método Edit (Get) el cual envía los datos de un Cliente seleccionado por su campo idcliente, y la lista de los registros de países en el ViewBag.paises, tal como se muestra



```

NegociosController.cs  ✘ X
Laboratorio07 Controllers.NegociosController Edit(string item)
public class NegociosController : Controller
{
    ServicioNegociosClient proxy = new ServicioNegociosClient();

    public ActionResult Clientes()...
    public ActionResult ClientesxNombre(string nom="")...
    public ActionResult Create()...
    [HttpPost]
    public ActionResult Create(Cliente reg)...

    public ActionResult Edit(string item)
    {
        Cliente reg = proxy.DetalleCliente(item);

        ViewBag.paises = new SelectList(proxy.Paises(), "idpais", "nombrepais", reg.Pais);
        return View(reg);
    }
}

```

Defina el método Post, donde ejecutamos el método para actualizar un registro a la tabla tb\_clientes desde el método ActualizarCliente()



```

NegociosController.cs  ✘ X
Laboratorio07 Controllers.NegociosController proxy
public class NegociosController : Controller
{
    ServicioNegociosClient proxy = new ServicioNegociosClient();

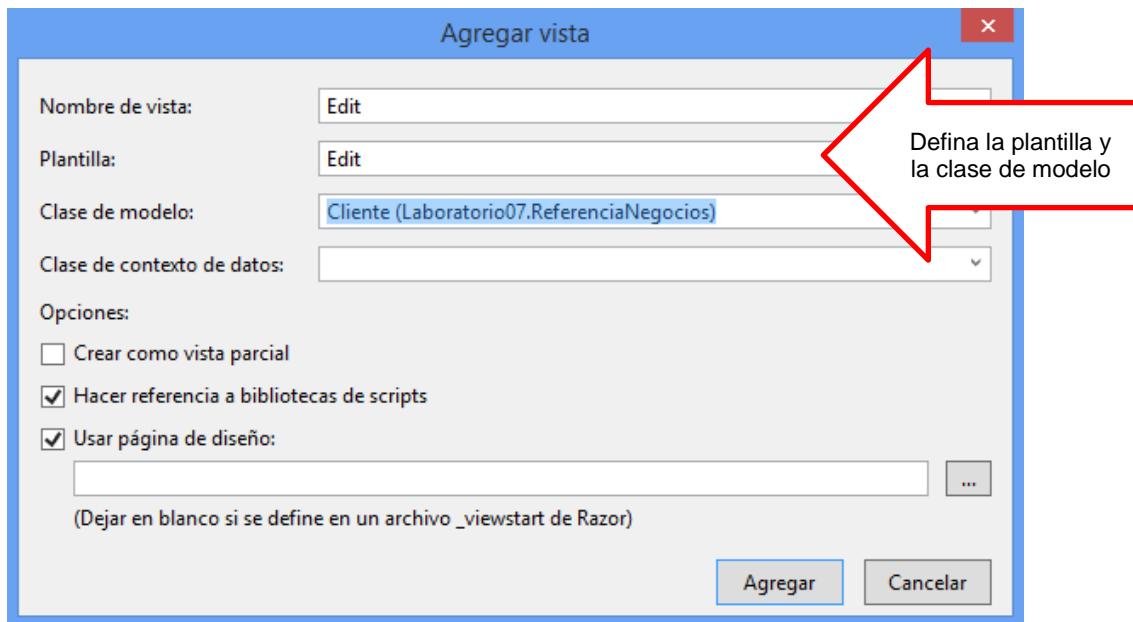
    public ActionResult Clientes()...
    public ActionResult ClientesxNombre(string nom="")...
    public ActionResult Create()...
    [HttpPost]
    public ActionResult Create(Cliente reg)...

    public ActionResult Edit(string item)...
    [HttpPost]
    public ActionResult Edit(Cliente reg)
    {
        string msg = proxy.ActualizarCliente(reg);
        ViewBag.msg = msg;

        return RedirectToAction("Clientes");
    }
}

```

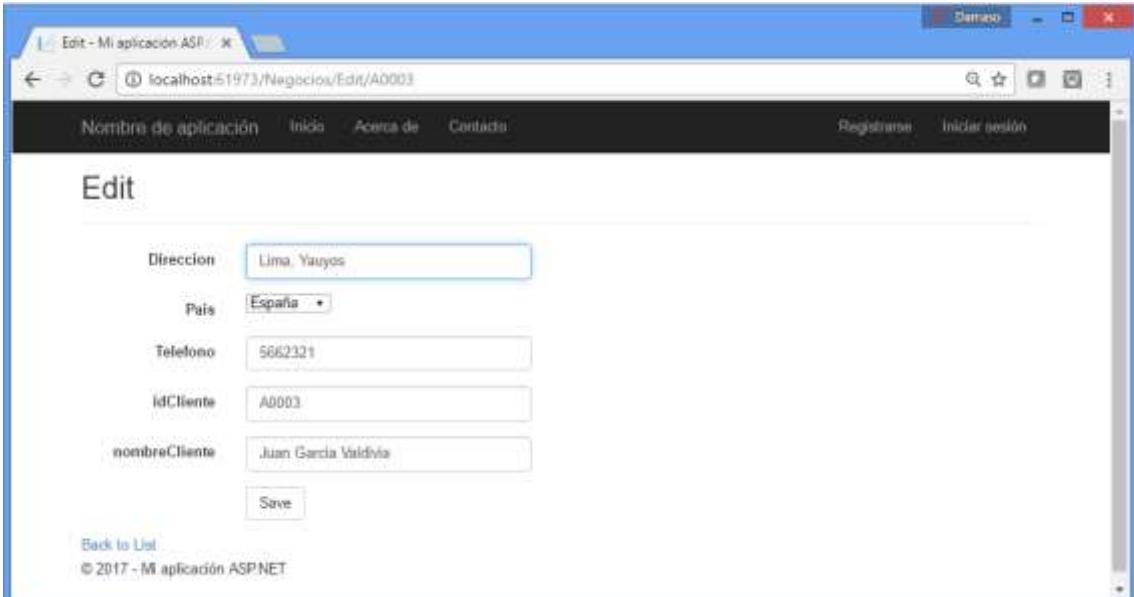
Defina la Vista del ActionResult: plantilla de tipo Edit y la clase de modelo Cliente, tal como se muestra



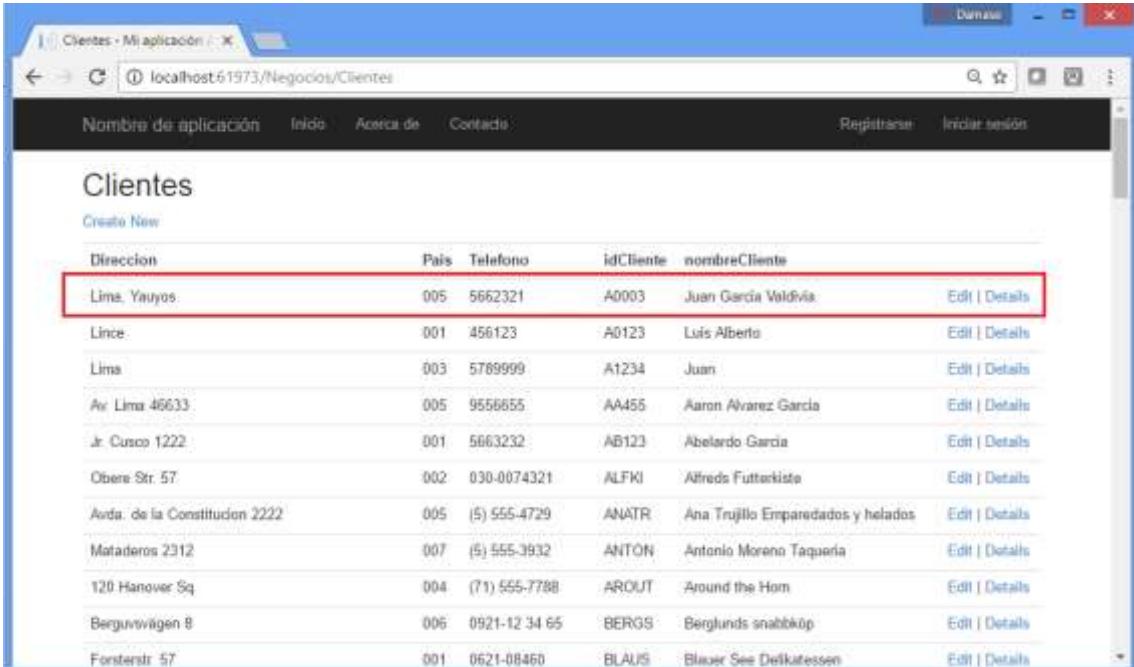
En la vista modificar el contenido del campo País, agregar un DropDownList para listar los registros de países, tal como se muestra



Ejecuta la Vista Clientes(), selecciona un cliente, donde se visualiza los datos del cliente, edite o modifique sus valores, al presionar el botón Save, se ejecuta el proceso y se visualiza en la lista los clientes y del cliente modificado.



The screenshot shows a web browser window titled 'Edit - Mi aplicación ASP.NET'. The URL is 'localhost:61973/Negocios/Edit/A0003'. The page has a header with 'Nombre de aplicación', 'Inicio', 'Acerca de', 'Contacto', 'Registrarse', and 'Iniciar sesión'. Below the header, the word 'Edit' is displayed. The main content area contains five input fields: 'Dirección' (Lima, Yauyos), 'País' (España), 'Teléfono' (5662321), 'idCliente' (A0003), and 'nombreCliente' (Juan García Valdivia). A 'Save' button is located below these fields. At the bottom left is a link 'Back to List' and at the bottom center is the copyright notice '© 2017 - Mi aplicación ASP.NET'.



The screenshot shows a web browser window titled 'Cuentas - Mi aplicación'. The URL is 'localhost:61973/Negocios/Cuentas'. The page has a header with 'Nombre de aplicación', 'Inicio', 'Acerca de', 'Contacto', 'Registrarse', and 'Iniciar sesión'. Below the header, the word 'Cuentas' is displayed. The main content area shows a table with the following data:

Dirección	País	Teléfono	idCliente	nombreCliente	
Lima, Yauyos.	005	5662321	A0003	Juan García Valdivia.	<a href="#">Edit</a>   <a href="#">Details</a>
Lince	001	456123	A0123	Luis Alberto	<a href="#">Edit</a>   <a href="#">Details</a>
Lima	003	5789999	A1234	Juan	<a href="#">Edit</a>   <a href="#">Details</a>
Av. Lima 46633.	005	95566555	AA455	Aaron Alvarez Garcia	<a href="#">Edit</a>   <a href="#">Details</a>
Jr. Cusco 1222	001	5663232	AB123	Abelardo García	<a href="#">Edit</a>   <a href="#">Details</a>
Obera Str. 57	002	030-0074321	ALFKI	Alfreds Futterkiste	<a href="#">Edit</a>   <a href="#">Details</a>
Avda. de la Constitución 2222	005	(5) 555-4729	ANATR	Ana Trujillo Emparedados y helados	<a href="#">Edit</a>   <a href="#">Details</a>
Mataderos 2312	007	(5) 555-3932	ANTÓN	Antonio Moreno Taquería	<a href="#">Edit</a>   <a href="#">Details</a>
120 Hanover Sq	004	(71) 555-7788	AROUT	Around the Horn	<a href="#">Edit</a>   <a href="#">Details</a>
Berguvsvägen 8	006	0921-12 34 65	BERGS	Berglunds snabbköp	<a href="#">Edit</a>   <a href="#">Details</a>
Forsterstr. 57	001	0621-08460	BLAUS	Blauer See Delikatessen	<a href="#">Edit</a>   <a href="#">Details</a>

# Resumen

- Windows Communication Foundation (WCF) es un marco de trabajo para la creación de aplicaciones orientadas a servicios. Con WCF, es posible enviar datos como mensajes asincrónicos de un extremo de servicio a otro. Un extremo de servicio puede formar parte de un servicio disponible continuamente hospedado por IIS, o puede ser un servicio hospedado en una aplicación. Un extremo puede ser un cliente de un servicio que solicita datos de un extremo de servicio. Los mensajes pueden ser tan simples como un carácter o una palabra enviados como XML, o tan complejos como un flujo de datos binarios.
- Los mensajes son enviados entre endpoints. Un endpoint es un lugar donde un mensaje es enviado, o recibido, o ambos. Un servicio expone uno o más application endpoints, y un cliente genera un endpoint compatible con uno de los endpoints de un servicio dado. La combinación de un servicio y un cliente compatibles conforman un communication stack.
- Una de las Ventajas de utilizar WCF
  - Código Centralizado. La principal ventaja del uso de servicios públicos es el código centralizado.
  - Seguridad. Donde se accede a la información que se encuentra en una sola capa de acceso, es posible tener un mejor control de la misma.
  - Escalabilidad
- El consumo de servicios WCF le habilita para agregar servicios WCF existentes al proceso empresarial. Es posible agregar varios servicios WCF a una única orquestación.
- Entre los adaptadores de envío WCF de BizTalk se incluyen cinco adaptadores de envío físicos que representan los enlaces predefinidos de WCF BasicHttpBinding, WsHttpBinding, NetTcpBinding, NetNamedPipeBinding y NetMsmqBinding. Asimismo, los adaptadores de WCF de BizTalk incorporan los adaptadores personalizados que permiten configurar libremente la información de comportamiento y enlace de WCF en un puerto de envío.
- Si desea ver mas
  - [https://msdn.microsoft.com/es-es/library/ms731082\(v=vs.110\).aspx](https://msdn.microsoft.com/es-es/library/ms731082(v=vs.110).aspx)
  - <http://www.esasp.net/2009/09/wcf-introduccion-y-conceptos-basicos.html>
  - <https://msdn.microsoft.com/es-es/library/bb226529.aspx>
  - <http://www.dotnetfunda.com/articles/show/1893/a-basic-introduction-to-creation-of-crud-operation-using-odata-web-ser>
  - <http://pabletoreto.blogspot.com/2014/12/crear-hospedar-y-consumir-un-wcf4.html>
  - <http://www.esasp.net/2009/09/wcf-desarrollando-un-servicio-y-cliente.html>



# CONSUMO DE SERVICIOS

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno desarrolla aplicaciones Web con Formularios para consumir servicios WCF.

## TEMARIO

### Tema 8: Implementación y consumo de servicios Web API (4 horas)

- 8.1 El web API y el modelo MVC
- 8.2 Verbos HTTP y convención para la implementación de un servicio Web
- 8.3 Implementación de un simple Web API
- 8.4 Consumo de servicios Web API desde una aplicación Web
- 8.5 Implementación de un proceso CRUD con Web API

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web consumiendo servicios para el manejo de datos utilizando el patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



## 8. Implementando y consumo de servicios Web API

### 8.1 El web API y el modelo MVC

La API de Web ASP.NET es un marco que hace que sea fácil de construir servicios HTTP que llegan a una amplia gama de clientes, incluyendo los navegadores y dispositivos móviles.



*Figura 1*  
<http://maninformatic.blogspot.pe/2013/12/diferencia-entre-mvc-y-web-api.html>

La API de Web ASP.NET es una plataforma ideal para la creación de aplicaciones cuyo propósito es exponer servicios bajo el protocolo HTTP utilizando el estilo de la arquitectura REST (Representational State Transfer).

#### ¿Por qué utilizar el API Web?

- Es un framework que usa los servicios HTTP y hace que esto sea fácil para proporcionar una respuesta a la solicitud del cliente. La respuesta depende de la petición de los clientes. Web API construye los servicios HTTP y gestiona la petición usando el protocolo HTTP. Web API es código abierto y este puede ser hospedado en la aplicación o en el servidor IIS. La petición puede ser GET, POST, DELETE o PUT.
- Permite mostrar los datos en varios formatos, como XML y JSON.
- Está diseñado para llegar a un amplio rango de clientes en aplicaciones móviles, tablets, Web.

### 8.2 Verbos HTTP y convención para la implementación de un servicio web

Como se mencionó anteriormente, en este artículo vamos a consumir un URI API WEB ASP.NET existentes (alojado en el servidor), por debajo de la mesa es de mencionar todos los URI:

Acción	Método HTTP	URI relativa
Obtenga una lista de ServerData	GET	/Api / ServerData

Acción	Método HTTP	URI relativa
Obtener una ServerData por ID	GET	/Api / ServerData / Identificación
Obtén ServerData por tipo de datos	GET	/Api / ServerData / tipo / tipo de datos
Obtén ServerData por la máquina IP	GET	/Api / ServerData / ip / ip
Crear un ServerData fresca	POST	/Api / ServerData
Actualizar una ServerData existente	PUT	/Api / ServerData / Identificación
Eliminar una ServerData existente	DELETE	/Api / ServerData / Identificación

### 8.3 Implementacion de un simple Web API

Si desea exponer los datos y/o información de una solicitud a sus clientes, pueden utilizar dichos datos e interactuar con los mismos para exponerlos a sus clientes en la Web.

Por ejemplo

- Una aplicación móvil requiere un servicio.
- HTML 5 requiere un servicio.
- PC de escritorio y tabletas requieren servicios.

Actualmente la mayoría de las aplicaciones de dispositivos requieren servicios API Web.

El Marco ASP.Net aprovecha tanto los estándares web como HTTP, JSON y XML y proporciona una forma sencilla de construir y exponer a los servicios de datos basados en REST.

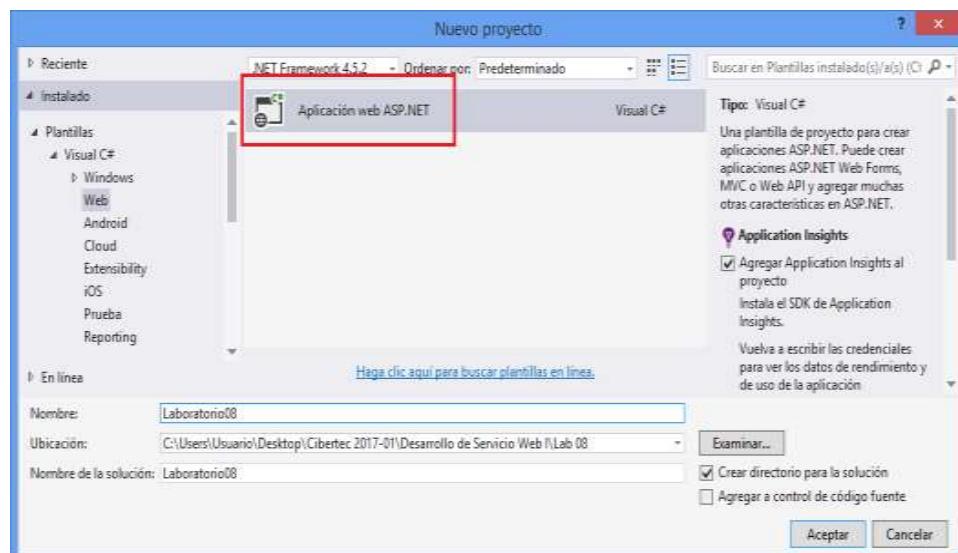
Algunos conceptos básicos de ASP.Net MVC son similares a la API de Web ASP.Net, tales como enrutamiento y los controladores.

#### Vamos a empezar con la creación de un proyecto Web API.

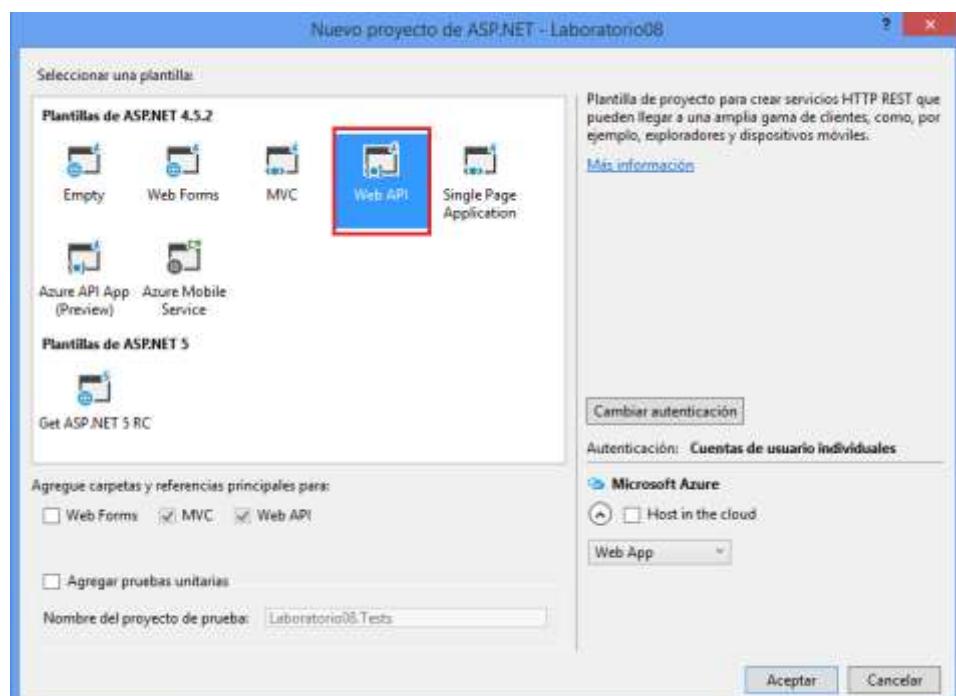
Inicie Visual Studio y seleccione Nuevo proyecto desde la página de inicio o en el menú Archivo, seleccione "Archivo" -> "Nuevo" -> "Proyecto ...".

En el panel de la plantilla seleccionar plantillas instaladas y ampliar el menú de Visual C#. Seleccione Web.

En la lista de proyectos seleccione Aplicación Web ASP.Net. Asigne el nombre del proyecto Web



Selecciona la plantilla Web API



## 8.4 Consumo de servicio Web API desde una aplicación web

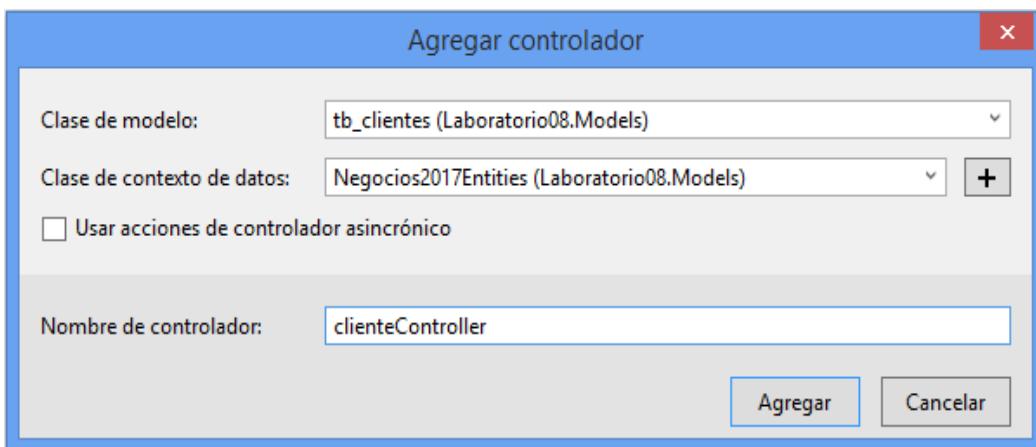
Para crear los controladores para la API. WebAPI es un framework MVC-como que podemos utilizar para crear fácilmente un servicio RESTful, y se puede ejecutar dentro de una aplicación MVC4, en su propio proyecto, o puede ser autoalojados fuera del IIS.

Pero eso no es todo; tiene muchas otras características, tales como: la negociación de contenido (para serializar automáticamente los datos en el formato que se solicita).



Primero tenemos que crear un punto final con WebAPI, y lo hacemos mediante la creación de una clase que hereda ApiController.

Crear ApiController



## 8.5 Implementación de un proceso CRUD con Web API

CRUD significa "Crear, leer, actualizar y eliminar", que son las cuatro operaciones básicas de base de datos. Muchos servicios HTTP también modelan las operaciones CRUD a través de REST o APIs similares a REST.

Para implementar un proceso CRUD, Web API nos provee de cuatro métodos HTTP principales (GET, PUT, POST y DELETE) se pueden asignar a operaciones CRUD de la siguiente manera.

- GET recupera la representación del recurso en un URI especificado. GET no debería tener efectos secundarios en el servidor.

- PUT actualiza un recurso en un URI especificado. PUT también se puede utilizar para crear un nuevo recurso en un URI especificado, si el servidor permite a los clientes especificar nuevos URI. Para este tutorial, la API no admite la creación mediante PUT.
- POST crea un nuevo recurso. El servidor asigna el URI para el nuevo objeto y devuelve este URI como parte del mensaje de respuesta.
- DELETE elimina un recurso en un URI especificado.

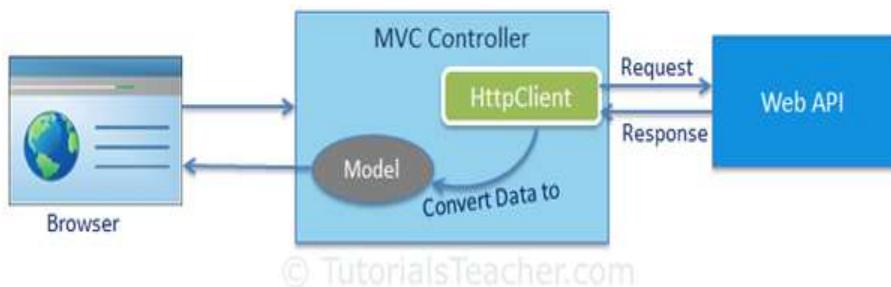
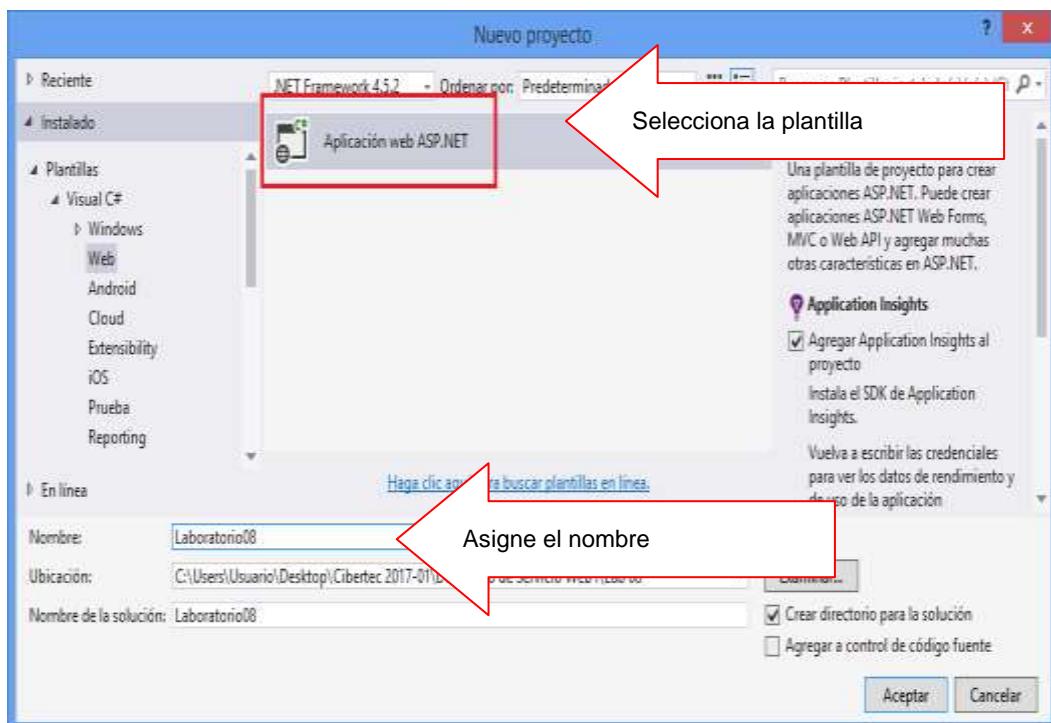


Figura 2  
<http://www.tutorialsteacher.com/webapi/consume-web-api-for-crud-operation>

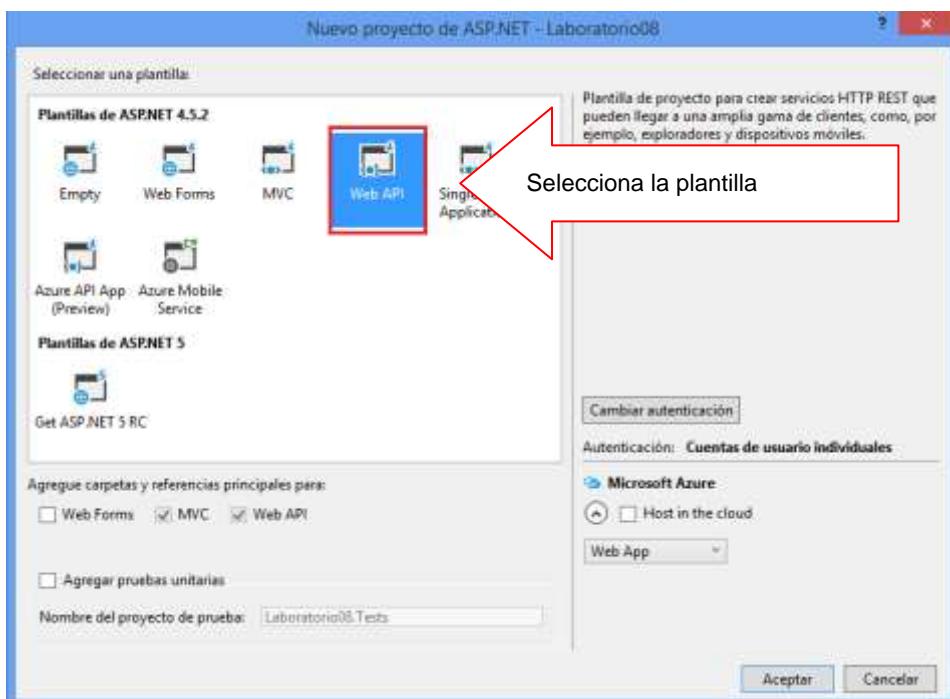
## Laboratorio 8.1

### Implementando consulta en ASP.NET MVC consumiendo un WEB API

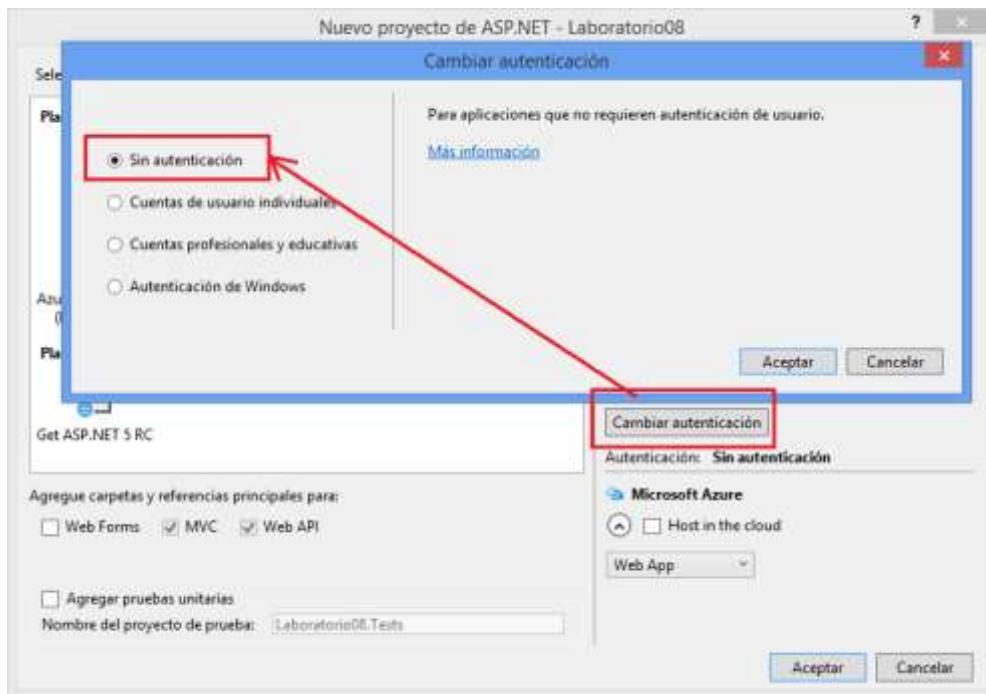
Implemente un proyecto ASP.NET MVC donde permita realizar las operaciones de consulta y actualización de datos consumiendo un servicio Web API.



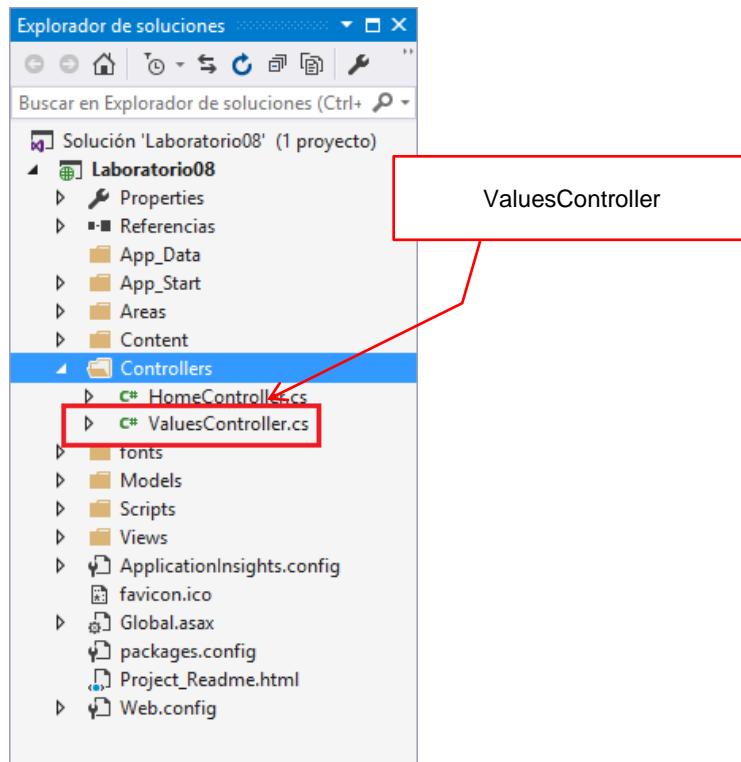
Selecciona la plantilla Web API



Cambiar la autenticación: Sin autenticación, tal como se muestra, presionar el botón Aceptar

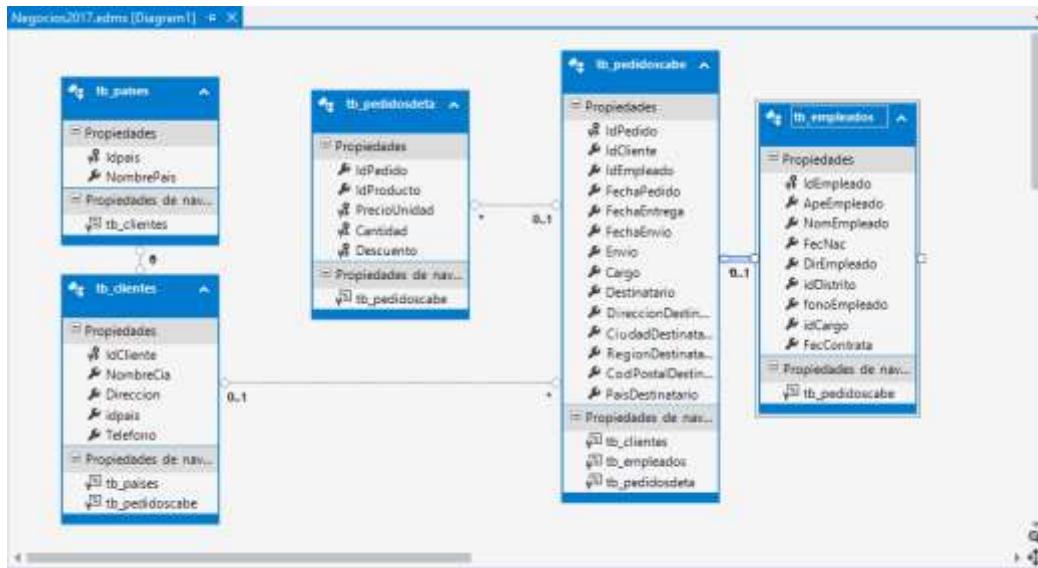


Al momento de crear el proyecto, en el controlador (Carpeta Controllers) se agrega el controlador ValuesController para definir los métodos del Web API



## Modelo de Datos

Se tiene el siguiente modelo de datos llamado Negocios2017Entities, el cual se encuentra almacenado en la carpeta Models



## Trabajando con ValuesController

En el controlador debemos referenciar la carpeta Models, la cual se almacena el modelo de datos. Dentro del controlador instanciamos el modelo de datos Negocios2017Entities, tal como se muestra.

Código de ValuesController.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using Laboratorio08.Models;
namespace Laboratorio08.Controllers
{
    public class ValuesController : ApiController
    {
        Negocios2017Entities bd = new Negocios2017Entities();
    }
}

```

Explicación de los pasos:

- Referenciar la carpeta Models**: Se muestra la declaración `using Laboratorio08.Models;` resaltada en rojo.
- Instanciar el modelo de datos de entidades**: Se muestra la línea de código `Negocios2017Entities bd = new Negocios2017Entities();` resaltada en rojo.

Dentro del ApiController, defina los métodos Get(), que permita listar todos los registros de tb\_clientes, y Get(parámetro) para buscar un cliente por su código. Además creamos GetPaises() donde retorna los registros de tb\_paises

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Web.Http;
using Negocios2017Entities;
using System.Data.Entity;

namespace Laboratorio08.Controllers
{
    public class ValuesController : ApiController
    {
        Negocios2017Entities bd = new Negocios2017Entities();

        // GET api/values
        public IEnumerable<tb_clientes> Get()
        {
            return bd.tb_clientes.ToList();
        }

        // GET api/values/A0001
        public tb_clientes Get(string id)
        {
            return bd.tb_clientes.ToList().Where(c => c.IdCliente == id).FirstOrDefault();
        }

        public IEnumerable<tb_paises> GetPaises()
        {
            return bd.tb_paises.ToList();
        }
    }
}

```

Metodo que retorna todos los registros

Metodo que retorna un registro por un idcliente

Metodo que retorna todos los registros de tb\_paises

A continuación definimos el método Post, el cual agrega un registro a la tabla tb\_clientes, tal como se muestra

```

public class ValuesController : ApiController
{
    Negocios2017Entities bd = new Negocios2017Entities();

    public IEnumerable<tb_clientes> Get()...
    public tb_clientes Get(string id)...
    public IEnumerable<tb_paises> GetPaises()...

    public void Post(tb_clientes reg)
    {
        try
        {
            tb_clientes obj = new tb_clientes();
            obj.IdCliente = reg.IdCliente;
            obj.NombreCia = reg.NombreCia;
            obj.Direccion = reg.Direccion;
            obj.idpais = reg.idpais;
            obj.Telefono = reg.Telefono;
            bd.tb_clientes.Add(obj);
            bd.SaveChanges();
        }
        catch (Exception) { }
    }

    public void Put(int id, [FromBody]string value)...

    public void Delete(int id)...
}

```

Método que permite agregar un registro a tb\_clientes

Luego, implementa el método Put, el cual permite actualizar el registro de tb\_clientes, tal como se muestra



```

ValuesController.cs  X
Laboratorio08          Laboratorio08.Controllers.ValuesController  Put(tb_clientes reg)
public class ValuesController : ApiController
{
    Negocios2017Entities bd = new Negocios2017Entities();

    public IEnumerable<tb_clientes> Get()...
    public tb_clientes Get(string id)...
    public IEnumerable<tb_paises> GetPaises()...

    public void Post(tb_clientes reg)...

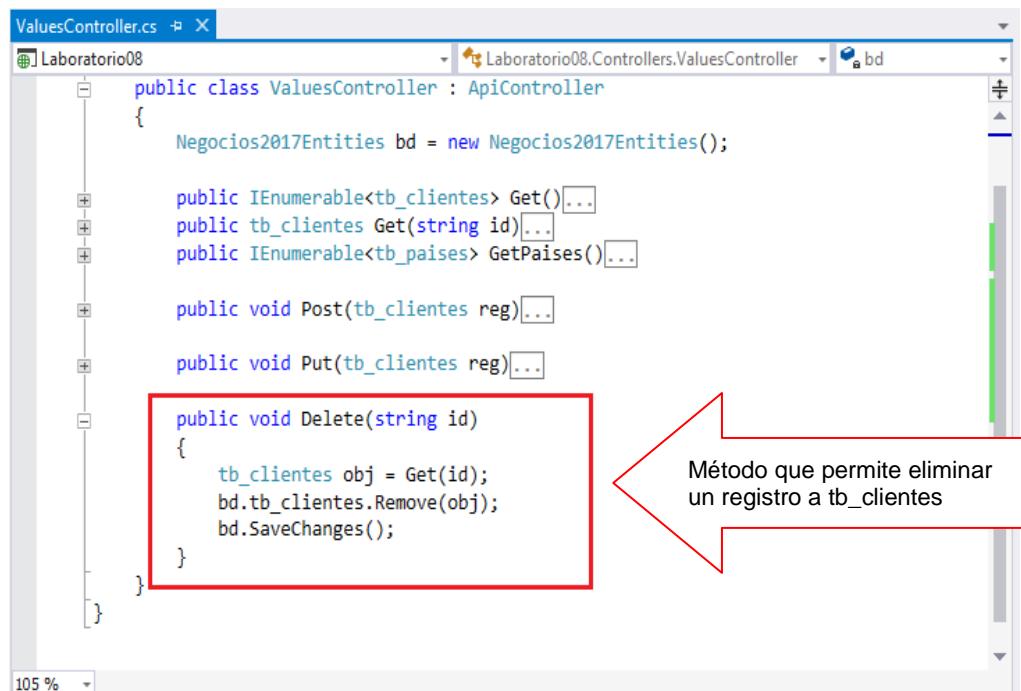
    public void Put(tb_clientes reg)
    {
        bd.Entry(reg).State = System.Data.EntityState.Modified;
        bd.SaveChanges();
    }

    public void Delete(int id)...
}

```

A red box highlights the Put(tb\_clientes reg) method. A red arrow points from this box to the text "Método que permite actualizar un registro a tb\_clientes".

Finalizando, implementa el método Delete, el cual elimina un registro de la tabla tb\_clientes, tal como se muestra



```

ValuesController.cs  X
Laboratorio08          Laboratorio08.Controllers.ValuesController  bd
public class ValuesController : ApiController
{
    Negocios2017Entities bd = new Negocios2017Entities();

    public IEnumerable<tb_clientes> Get()...
    public tb_clientes Get(string id)...
    public IEnumerable<tb_paises> GetPaises()...

    public void Post(tb_clientes reg)...

    public void Put(tb_clientes reg)...

    public void Delete(string id)
    {
        tb_clientes obj = Get(id);
        bd.tb_clientes.Remove(obj);
        bd.SaveChanges();
    }
}

```

A red box highlights the Delete(string id) method. A red arrow points from this box to the text "Método que permite eliminar un registro a tb\_clientes".

## Trabajando con el Controlador: Action Index

En HomeController, instanciar el controlador ValuesController, tal como se muestra. Defina el ActionResult Index, el cual envía a la vista el método Get().

```

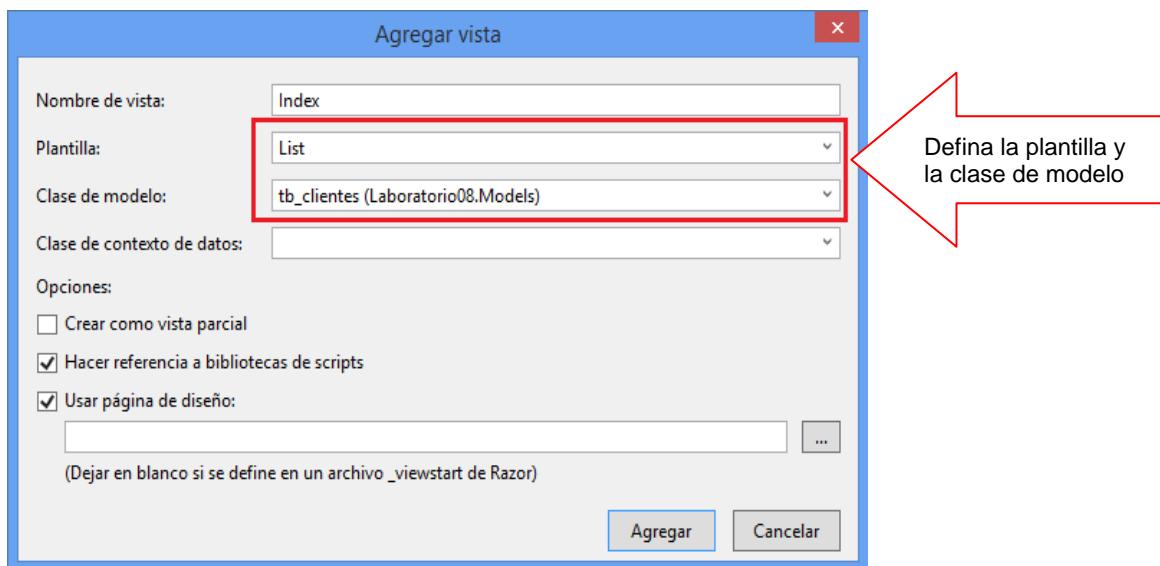
HomeController.cs*  X  ValuesController.cs*
Laboratorio08          Laboratorio08.Controllers.HomeController      bd
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Laboratorio08.Controllers
{
    public class HomeController : Controller
    {
        ValuesController bd = new ValuesController();

        public ActionResult Index()
        {
            return View(bd.Get());
        }
    }
}

```

Agrega la Vista del método, de plantilla List y de clase de modelo tb\_clientes, tal como se muestra



Modifica la estructura de la vista sobre los ActionLink, tal como se muestra

```

Index.cshtml" # × HomeController.cs      ValuesController.cs
@model IEnumerable<Laboratorio08.Models.tb_clientes>

    @{
        ViewBag.Title = "Index";
    }

    <h2>Index</h2>

    <p>@Html.ActionLink("Create New", "Create")</p>
    <table class="table">
        <tr>
            <th>@Html.DisplayNameFor(model => model.IdCliente)</th>
            <th>@Html.DisplayNameFor(model => model.NombreCia)</th>
            <th>@Html.DisplayNameFor(model => model.Direccion)</th>
            <th>@Html.DisplayNameFor(model => model.idpais)</th>
            <th>@Html.DisplayNameFor(model => model.Telefono)</th>
            <th></th>
        </tr>

        @foreach (var item in Model) {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.IdCliente)</td>
                <td>@Html.DisplayFor(modelItem => item.NombreCia)</td>
                <td>@Html.DisplayFor(modelItem => item.Direccion)</td>
                <td>@Html.DisplayFor(modelItem => item.idpais)</td>
                <td>@Html.DisplayFor(modelItem => item.Telefono)</td>
                <td>
                    @Html.ActionLink("Edit", "Edit", new { id=item.IdCliente }) |
                    @Html.ActionLink("Details", "Details", new { id=item.IdCliente }) |
                    @Html.ActionLink("Delete", "Delete", new { id=item.IdCliente })
                </td>
            </tr>
        }
    </table>

```

Ejecuta la vista, donde se lista los registros de tb\_clientes

Index					
Create New					
IdCliente	NombreCia	Direccion	idpais	Telefono	
A0003	Juan Garcia Velarde	Lima, Yanayos	005	6662125	Edit   Details   Delete
A0123	Luis Alberto	Lince	001	456123	Edit   Details   Delete
A1234	Juan	Lima	003	5789999	Edit   Details   Delete
AA456	Aaron Alvarez Garcia	Av Lima 40633	005	9656665	Edit   Details   Delete
AB123	Abelardo Garcia	Jr Cuervo 1222	001	6663232	Edit   Details   Delete
ALF10	Alteds Futurista	Obras St 57	002	030-0074321	Edit   Details   Delete
ANATR	Ana Trujillo Empredados y helados	Avda. de la Constitucion 2222	005	(5) 555-4729	Edit   Details   Delete
ANTON	Antonio Moreno Taqueria	Mataderos 2312	007	(5) 555-3832	Edit   Details   Delete
ARCUT	Around the Horn	129 Hanover Sq.	004	(71) 555-7788	Edit   Details   Delete
BERGS	Berglunds snabbköp	Berguvsvägen 8	006	0921-12 34 65	Edit   Details   Delete
BLAUS	Blauer See Delikatessen	Festivit 57	001	0621-08460	Edit   Details   Delete
BLONF	Blondel pâtes et fils	24, place Kleber Estrasburgo	008	88 60 15 31	Edit   Details   Delete
BOLID	Bolido Comidas preparadas	C/ Araceli, 67	009	(91) 555 91 99	Edit   Details   Delete

## Trabajando con el Controlador: Action Create

En el HomeController, defina el ActionResult Create, el cual envía la vista con un registro en blanco y recibe el registro con los datos para ejecutar el método Post del ValueController

```

HomeController.cs  X
Laboratorio08    Laboratorio08.Controllers.HomeController
public class HomeController : Controller
{
    ValuesController bd = new ValuesController();

    public ActionResult Index()...
    public ActionResult Create()
    {
        ViewBag.paises = new SelectList(bd.GetPaises(), "idpais", "nombrepais");
        return View(new tb_clientes());
    }
}

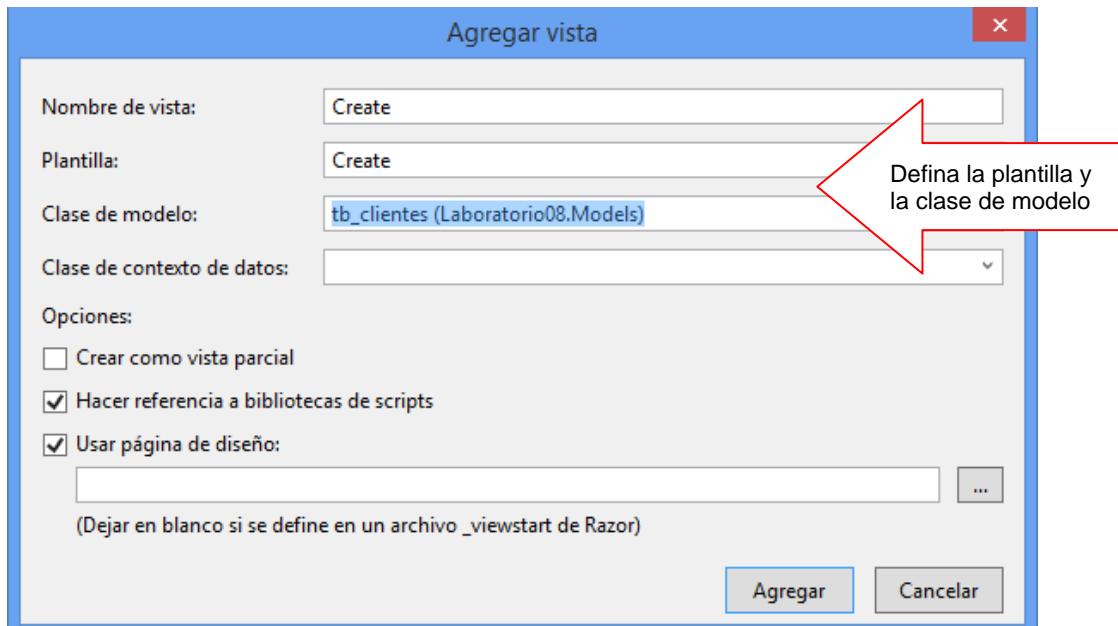
[HttpPost]
public ActionResult Create(tb_clientes reg)
{
    if (!ModelState.IsValid)
    {
        return View(reg);
    }
    bd.Post(reg);
    return RedirectToAction("Index");
}
}

```

Método donde envía un registro en blanco

Método que recibe el registro y ejecuta el método Post()

Crear la vista del Action Create, tal como se muestra



En la Vista, modifica el scaffold del campo idpais por un DropDownList, tal como se muestra en la figura.

The screenshot shows the 'Create.cshtml' file in a code editor. A red box highlights the following code block:

```
        <div class="form-group">
            @Html.Label("Pais", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownList("idpais", (SelectList)ViewBag.paises, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>
```

A red arrow points from a callout box containing the text "Modifica el Scafold de idpais por un DropDownList" to the highlighted code block.

Ejecuta la Vista, ingresa los datos a los controles, al presionar el botón Create, el registro se agrega a la tabla y se visualiza en la página Index, tal como se muestra

The screenshot shows a web browser window with the title 'Create'. The address bar shows 'localhost:59487/Home/Create'. The page content is titled 'Create' and contains the following form fields:

IdCliente	A0005
NombreCia	Jorge Perez
Direccion	Lima
Pais	Peru
Telefono	12345698

Below the form is a 'Create' button. At the bottom of the page is a 'Back to List' link.

IdCliente	NombreCis	Dirección	idpais	Telefono	
A0093	Juan Garcia Valdivia	Lima, Yaquia	001	5662321	Edit   Details   Delete
A0094	Juan Carlos	Lince	007	4667891	Edit   Details   Delete
A0095	Jorge Perez	Lima	001	12345668	Edit   Details   Delete
AB123	Luis Alberto	Lince	001	466123	Edit   Details   Delete
A1234	Juan	Lima	003	5789999	Edit   Details   Delete
AA155	Aaron Alvarez Garcia	Av. Lima 46633	005	9556655	Edit   Details   Delete
AB123	Abelardo Garcia	Jr. Casco 1222	001	5663232	Edit   Details   Delete
ALFRI	Alfredo Fuentes	Obrero St. 57	002	(5) 999-4729	Edit   Details   Delete
ANATR	Ana Trujillo Emparedados y helados	Avenida de la Constitucion 2222	006	(5) 999-4729	Edit   Details   Delete
ANTON	Antonio Munoz Tocino	Miraflores 7743	007	466-555-3032	Edit   Details   Delete

## Trabajando con el Controlador: Action Edit

En el HomeController, defina el ActionResult Edit, el cual envía la vista un registro de la tabla tb\_clientes y recibe el registro con los datos para ejecutar el método Put del ValueController

```

HomeController.cs + X
Liberatorio08
public class HomeController : Controller
{
    ValuesController bd = new ValuesController();
    public ActionResult Index()...
    public ActionResult Create()...
    [HttpPost]
    public ActionResult Create(tb_clientes reg)...

    public ActionResult Edit(string id)
    {
        tb_clientes reg = bd.Get(id);
        ViewBag.paises = new SelectList(bd.GetPaises(), "idpais", "nombrepais", reg.idpais);
        return View(reg);
    }

    [HttpPost]
    public ActionResult Edit(tb_clientes reg)
    {
        if (!ModelState.IsValid)
        {
            ViewBag.paises = new SelectList(bd.GetPaises(), "idpais", "nombrepais", reg.idpais);
            return View(reg);
        }

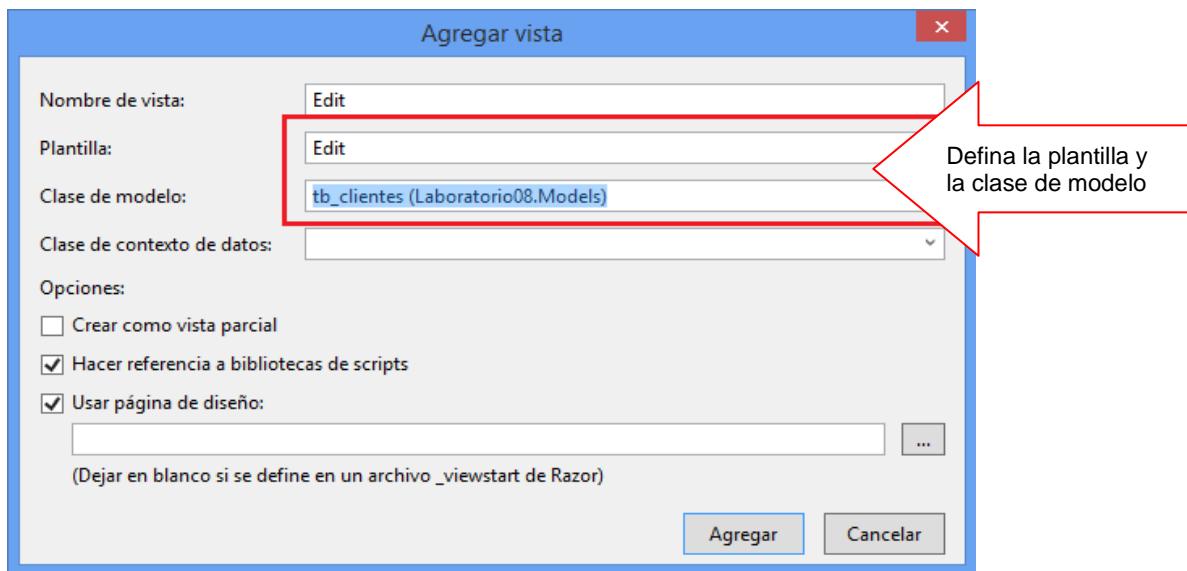
        bd.Put(reg);
        return RedirectToAction("Index");
    }
}

```

Método donde envía un registro de tb\_clientes

Método que recibe el registro y ejecuta el método Put()

Crear la Vista Edit, tal como se muestra



En la Vista, modifica el scaffold del campo idpais por un DropDownList, tal como se muestra en la figura.

```

using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>tb_clientes</h4>
        <br />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">...</div>

        <div class="form-group">...</div>

        <div class="form-group">...</div>
        <div class="form-group">
            @Html.Label("País", htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.DropDownList("idpais", (SelectList)ViewBag.paises, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>
    </div>
    <div class="form-group">...</div>
    <div class="form-group">...</div>
</div>

<div><a href="#">Back to List</a></div>
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}

```

Ejecuta la Vista Index, al hacer click en el link Edit de un registro, visualizamos en la página Edit, los datos del cliente seleccionado. Modificar su contenido. Al hacer click en el botón Save, se redirecciona a la página index(), con los datos actualizados

The screenshot shows a web browser window with the title bar "Index" and "Edit". The address bar displays "localhost:59487/Home/Edit/A0003". The main content area has a header "Edit". Below it are five input fields: "IdCliente" (A0003), "NombreCia" (Juan García Valdivia), "Direccion" (Jr Trujillo 123, Yauyos), "País" (Peru), and "Teléfono" (98856456). A "Save" button is at the bottom. At the very bottom of the page, there is a link "Back to List".

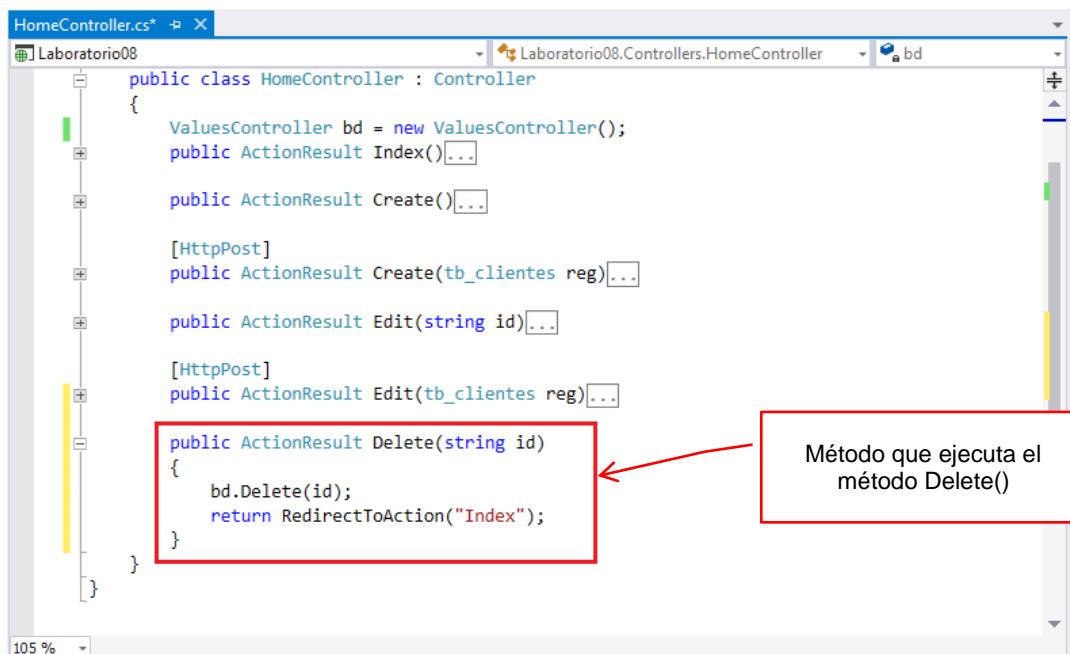
Visualizamos el registro actualizado

The screenshot shows a web browser window with the title bar "Index" and "Index". The address bar displays "localhost:59487". The main content area has a header "Index" and a "Create New" link. Below is a table with columns: IdCliente, NombreCia, Direccion, Idpais, Teléfono, and actions (Edit | Details | Delete). The table lists ten rows of client data. The first row, which corresponds to the edited record (A0003, Juan García Valdivia), is highlighted with a red border.

IdCliente	NombreCia	Direccion	Idpais	Teléfono	
A0003	Juan García Valdivia	Jr Trujillo 123, Yauyos	001	98856456	Edit   Details   Delete
A0004	Juan Carlos	Lince	007	4567891	Edit   Details   Delete
A0005	Jorge Pérez	Lima	001	12345698	Edit   Details   Delete
A0123	Luis Alberto	Lince	001	456123	Edit   Details   Delete
A1234	Juan	Lima	003	5789999	Edit   Details   Delete
AA456	Aaron Álvarez García	Av. Lima #6633	005	9556655	Edit   Details   Delete
AB123	Abelardo García	Jr. Cusco 1222	001	5663232	Edit   Details   Delete
ALF10	Alfredo Futterkatz	Obera Str 57	002	030-0074321	Edit   Details   Delete
ANATR	Ana Trujillo Emparedados y helados	Avenida de la Constitución 2222	005	(5) 555-4729	Edit   Details   Delete
ANTONI	Antonio Monroy Trujillo	Mateo Salvo 2225	007	(5) 555-3002	Edit   Details   Delete

## Trabajando con el Controlador: Action Delete

En el HomeController, defina el ActionResult Delete, el cual ejecuta el método Delete del ValueController

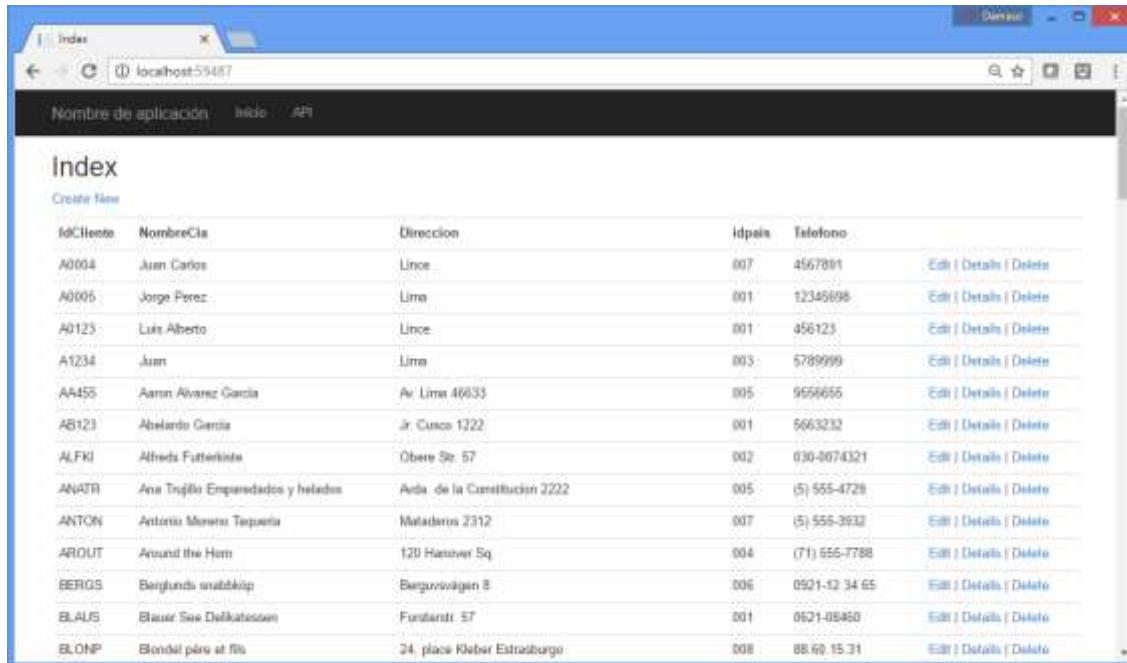


```

HomeController.cs*  X
Laboratorio08
public class HomeController : Controller
{
    ValuesController bd = new ValuesController();
    public ActionResult Index()...
    public ActionResult Create()...
    [HttpPost]
    public ActionResult Create(tb_clientes reg)...
    public ActionResult Edit(string id)...
    [HttpPost]
    public ActionResult Edit(tb_clientes reg)...
    public ActionResult Delete(string id)
    {
        bd.Delete(id);
        return RedirectToAction("Index");
    }
}

```

Ejecuta la vista Index, selecciona un registro de tb\_clientes, al presionar el link Delete, el registro se elimina y se visualiza la lista actualizada, tal como se muestra.



Index					
Create New					
IdCliente	NombreCia	Direccion	idpais	Telefono	
A0004	Juan Carlos	Lince	007	4567891	Edit   Details   Delete
A0005	Jorge Perez	Lima	001	12345698	Edit   Details   Delete
A0123	Luis Alberto	Lince	001	456123	Edit   Details   Delete
A1234	Juan	Lima	003	5789999	Edit   Details   Delete
AA456	Aarin Alvarez Garcia	Av. Lima 46633	005	9656655	Edit   Details   Delete
AB123	Abelardo Garcia	Jr. Cusco 1222	001	5663232	Edit   Details   Delete
ALFKI	Alfredo Futerlaria	Obere Str. 57	002	030-0074321	Edit   Details   Delete
ANATR	Ana Trujillo Emparedados y helados	Avenida de la Constitucion 2222	005	(5) 555-4729	Edit   Details   Delete
ANTON	Antonio Moreno Tequeria	Mataderos 2312	007	(5) 555-2932	Edit   Details   Delete
AROUT	Around the Horn	120 Hanover Sq.	004	(71) 555-7788	Edit   Details   Delete
BERGS	Berglunds snabbköp	Berguvsvägen 8	006	0921-12 34 65	Edit   Details   Delete
BLAUS	Blauer See Delikatessen	Furstenstr. 57	001	0621-08460	Edit   Details   Delete
BLONP	Bondil pâche et fils	24, place Kleber Estrasburgo	008	88.60.15.31	Edit   Details   Delete

The screenshot shows a Microsoft Edge browser window with the title bar "Index" and the address bar "C:\localhost:59487/Home/Index". The page content is titled "Index" and includes a "Create New" button. Below is a table with 14 rows of client data:

IdCliente	NombreCia	Direccion	idpais	Telefono	
A0003	Juan Garcia Valdivia	Jr Trujillo 123, Yurayos	001	98056456	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
A0004	Juan Carlos	Lima	007	4567891	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
A0005	Jorge Perez	Lima	001	12345698	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
A0123	Luis Alberto	Lima	001	456123	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
A1234	Juan	Lima	003	5769999	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
AA155	Aarón Alvarez García	Av. Lima 46633	005	9656655	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
AB123	Abelardo García	Jr. Cusco 1222	001	5663232	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
ALFKI	Alfredo Futterkatz	Oberoi Str. 57	002	030-0074321	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
ANATR	Ana Trujillo Emparedados y helados	Ave. de la Constitución 2222	005	(5) 555-4729	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
ANTON	Antonio Moreno Taquería	Mataderos 2312	007	(5) 555-3932	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
AROUT	Around the Horn	120 Hanover Sq.	004	(71) 555-7788	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BERGS	Berglunds snabbköp	Berguvsvägen 8	006	0921-12 34 65	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
BLAUS	Blauer See Delikatessen	Forstenstr. 67	001	0621-08460	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

# Resumen

- La Web API se vuelve cada vez más importante con la proliferación de dispositivos que utilizamos hoy en día. La mayoría de los dispositivos móviles, como teléfonos y tablets, se ejecutan aplicaciones que utilizan datos recuperados de la Web a través de HTTP. Las aplicaciones de escritorio también se están moviendo en esta dirección con más y más contenido en línea y sincronización y Windows 8 prometiendo una experiencia en aplicaciones similares.
- Del mismo modo, muchas aplicaciones web se basan en la funcionalidad del cliente enriquecido para crear y manipular la interfaz de usuario del navegador, usando AJAX en lugar de datos HTML generados por el servidor para cargar la interfaz de usuario con los datos.
- Si desea exponer los datos / información de su solicitud a sus clientes y otras personas luego de que otras personas puedan utilizar sus datos e interactuar con los datos / información se expone a ellos.

Por ejemplo

- Una aplicación móvil requiere un servicio.
- HTML 5 requiere un servicio.
- PC de escritorio y tabletas requieren servicios.

Actualmente la mayoría de las aplicaciones de dispositivos requieren servicios API Web.

El Marco ASP.Net aprovecha tanto los estándares web como HTTP, JSON y XML y proporciona una forma sencilla de construir y exponer a los servicios de datos basados en REST. Algunos conceptos básicos de ASP.Net MVC son similares a la API de Web ASP.Net, tales como enrutamiento y los controladores.

- Si desea ver mas

- <http://www.c-sharpcorner.com/UploadFile/4b0136/implement-Asp-Net-web-api-2-in-Asp-Net-mvc-5/>
- <https://www.codeproject.com/Articles/805931/ASP-NET-MVC-Features-WebAPI>
- <http://www.codemag.com/Article/1601031>
- <http://www.dotnetglobe.com/2012/03/crud-operation-using-aspnet-web-api-in.html>
- <https://www.mindstick.com/Articles/1472/crud-operation-using-asp-dot-net-web-api-and-entity-framework-in-asp-dot-net-mvc-4>
- <http://www.codesolution.org/crud-operations-using-asp-net-web-api/>
- <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api>
- <http://dotnetmentors.com/web-api/rest-based-crud-operations-with-asp-net-web-api.aspx>



# PATRONES DE DISEÑO CON ASP.NET MVC

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno conoce las técnicas del uso de patrones y mejores prácticas para construir software mantenible y durable en el tiempo, así también mejora la interacción de la aplicación de lado del cliente.

## TEMARIO

### Tema 09: Inversion of Control (IoC) (3 horas)

- 9.1 El patrón IoC
- 9.2 Implementaciones del IoC: Services Locator
- 9.3 Implementaciones del IoC: Dependency Injection (DI)

## ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando el patrón de diseño Inversion of Control y el patrón de diseño Modelo Vista Controlador.
- Los alumnos desarrollan los laboratorios de esta semana



## 9. Inversion of Control

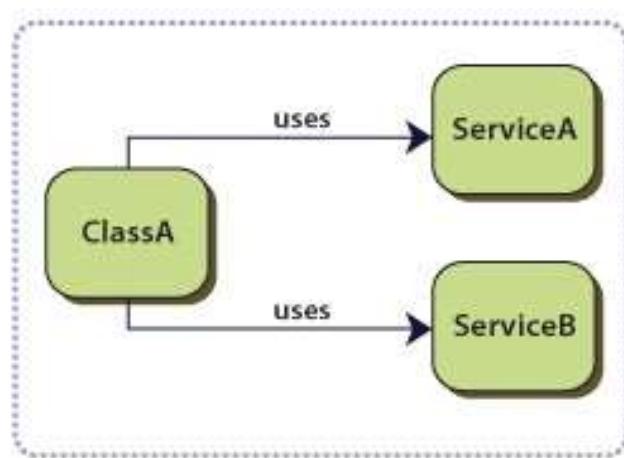
### 9.1 El patrón IoC

La Inversión de Control es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así el reuso de los mismos

#### Un problema, una solución

Como todo patrón, comienza planteando un problema y el viene a resolver.

Muchas veces, un componente tiene dependencias de servicios o componentes cuyos tipos concretos son especificados en tiempo de diseño



En la imagen previa, clase A depende de ServiceA y de ServiceB.

Los problemas que esto plantea son:

- Al reemplazar o actualizar las dependencias, se necesita cambiar el código fuente de la clase A.
- Las implementaciones concretas de las dependencias tienen que estar disponibles en tiempo de compilación.
- Las clases son difíciles de testear aisladamente porque tienen directas definiciones a sus dependencias. Esto significa que las dependencias no pueden ser reemplazadas por componentes stubs o mocks.
- Las clases tienen código repetido para crear, localizar y gestionar sus dependencias.

Aquí la solución pasa por delegar la función de seleccionar una implementación concreta de las dependencias a un componente externo.

El control de cómo un objeto A obtiene la referencia de un objeto B es invertido. El objeto A no es responsable de obtener una referencia al objeto B sino que es el Componente Externo el responsable de esto. Esta es la base del patrón IoC.

El patrón IOC aplica un principio de diseño denominado principio de Hollywood (No nos llames, nosotros te llamaremos).

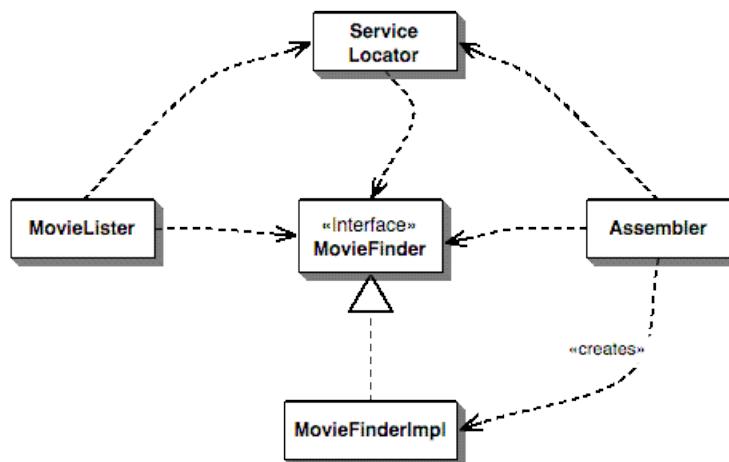
Actualmente existen dos técnicas de implementación para el IoC: Inyección de dependencias y Service Locutor; en este manual nos enfocaremos en la implementación de la Inyección de Dependencia (DI)

## 9.2 Implementaciones del IoC: Services Locator

Service Locator es un patrón que nos permite localizar servicios. Es una clase, el ServiceLocator a la cual le podemos pedir instancias de un servicio concreto.

Esto es útil para poder tener distintas implementaciones de un mismo servicio y cambiar, mediante configuración, la implementación que queremos que devuelva el ServiceLocator cuando le pidamos la instancia del servicio. ServiceLocator actúa como un catálogo central de instancias de servicios al que le podemos solicitar la instancia del servicio que necesitemos.

La idea básica detrás de un Localizador de Servicios es tener un objeto que sabe cómo obtener todos los servicios que una aplicación puede necesitar. Así que, el Localizador de Servicios para esta aplicación tendría un método que devuelve un objeto finder cuando esto es necesario.



Muchas veces he oído quejas sobre que este tipo de Localizador de Servicios no son buenos porque no permiten tests ya que no es posible reemplazar sus implementaciones.

Ciertamente, puedes diseñarlos mal y entrar en este tipo de dificultades, pero tu no tienes porque hacerlo. En este caso, la instancia del Localizador de Servicios es sólo un simple contenedor para los datos. Puedo crear fácilmente el Localizador de Servicios con las implementaciones de prueba de mis servicios.

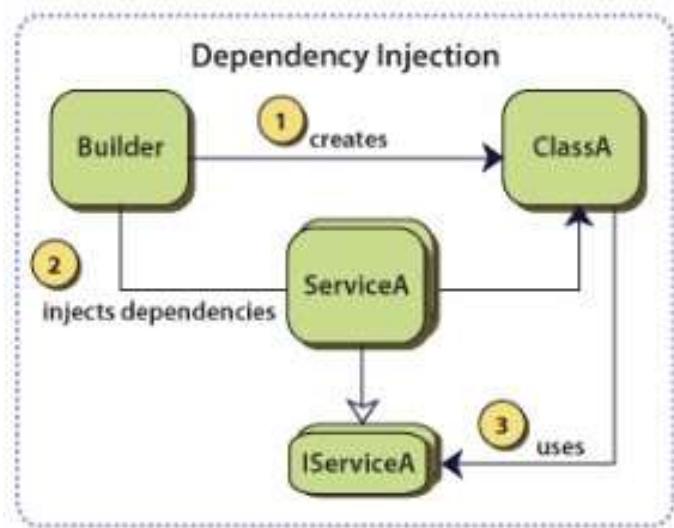
Para un localizador más sofisticado puedo heredar del Localizador de Servicios y pasar esta subclase en la variable de la clase de registro. Puedo cambiar los métodos estáticos para llamar a un método en la instancia en lugar de acceder directamente a las variables de instancia. Puedo proporcionar localizadores para un hilo específico usando una ubicación de almacenamiento específica para el hilo. Todo esto puede ser realizado sin cambios en los clientes del Localizador de Servicios.

## 9.3 Implementaciones del IoC: Dependency Injection (DI)

Una dependencia entre un componente y otro, puede establecerse estáticamente o en tiempo de compilación, o bien, dinámicamente o en tiempo de ejecución. Es en éste último escenario es donde cabe el concepto de inyección, y para que esto fuera posible, debemos referenciar interfaces y no implementaciones directas.

En general, las dependencias son expresadas en términos de interfaces en lugar de clases concretas. Esto permite un rápido reemplazo de las implementaciones dependientes sin modificar el código fuente de la clase.

Lo que propone entonces la Inyección de dependencias, es no instanciar las dependencias explícitamente en su clase, sino que declarativamente expresarlas en la definición de la clase. La esencia de la inyección de las dependencias es contar con un componente capaz de obtener instancias válidas de las dependencias del objeto y pasárselas durante la creación o inicialización del objeto.



Una manera de inyectar las dependencias es utilizando un constructor con parámetros del objeto dependiente. Éste constructor recibe las dependencias como parámetros y las establece en los atributos del objeto.

Considerando un diseño de dos capas donde tenemos una capa de BusinessFacade y otra de BusinessLogic, la capa BusinessFacade depende de la BusinessLogic para operar correctamente. Todas las clases de lógica de negocio implementan la interface IBusinessLogic.

En la inyección basada en un constructor, se creará una instancia de BusinessFacade usando un constructor parametrizado al cual se le pasará una referencia de un IBusinessLogic para poder inyectar la dependencia.

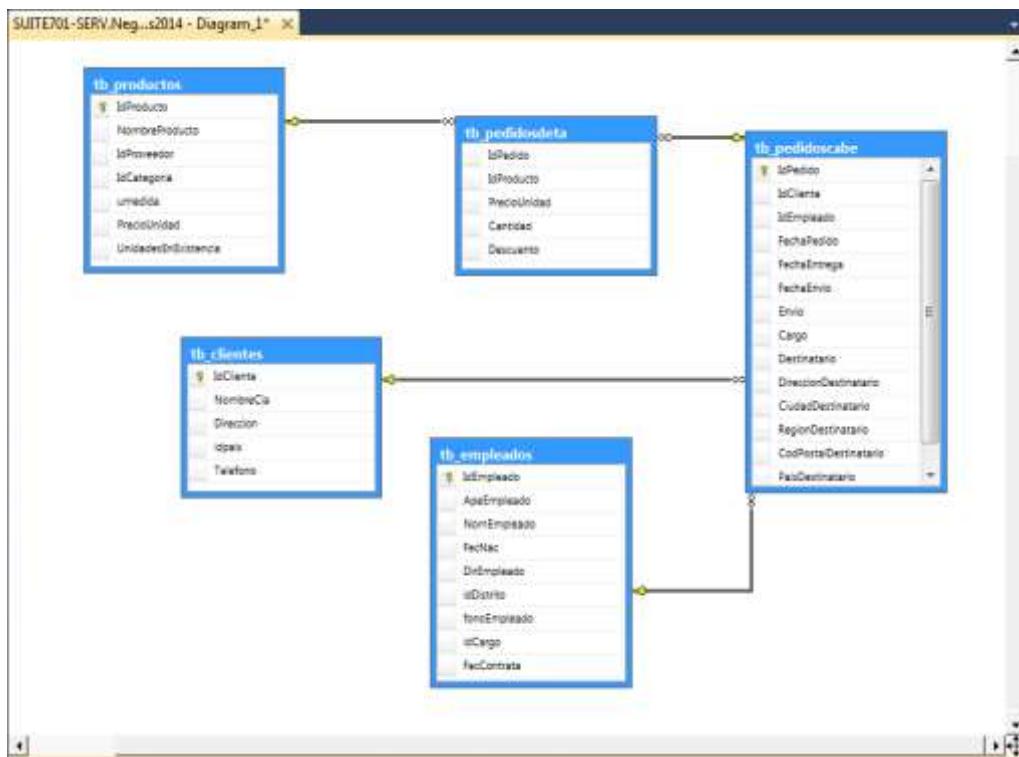
Finalmente, la inyección de dependencias no debería usarse siempre que una clase dependa de otra, sino más bien es efectiva en situaciones específicas como las siguientes:

- Inyectar datos de configuración en un componente.
- Inyectar la misma dependencia en varios componentes.
- Inyectar diferentes implementaciones de la misma dependencia.
- Inyectar la misma implementación en varias configuraciones
- Se necesitan alguno de los servicios provistos por un contenedor.

La IoC no es necesaria si uno va a utilizar siempre la misma implementación de una dependencia o la misma configuración, o al menos, no reportará grandes beneficios en estos casos.

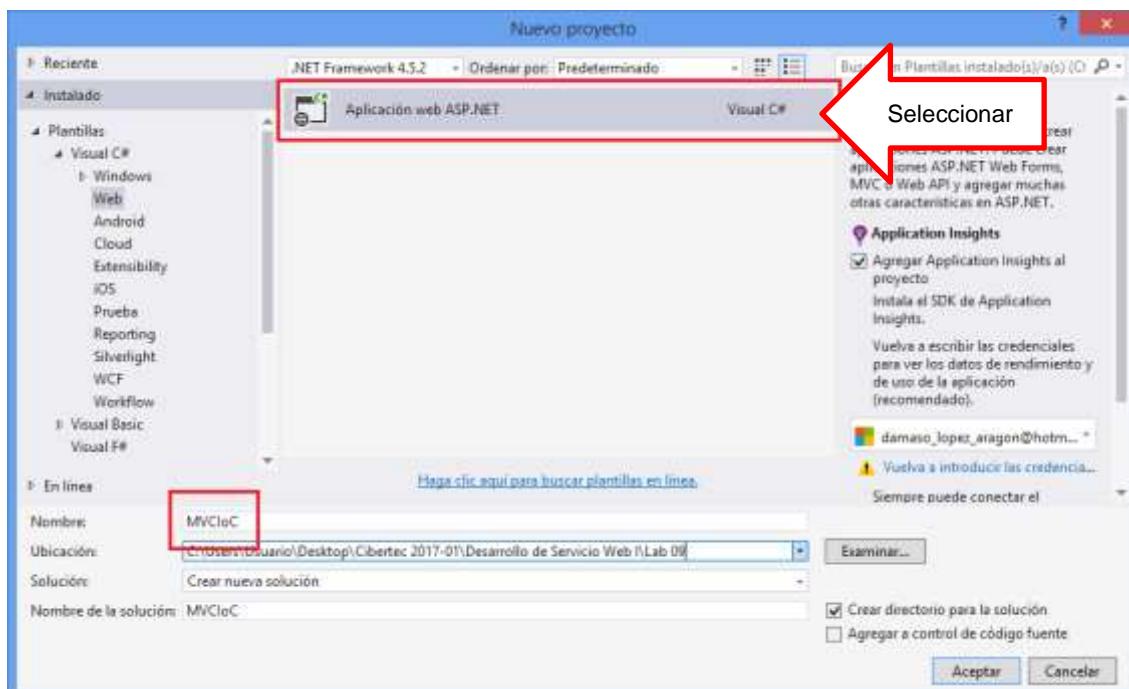
## 9.1 Implementación del DI en MVC

Se requiere implementar las consultas y listados a las tablas de la base de datos Negocios2014, tal como se muestra.

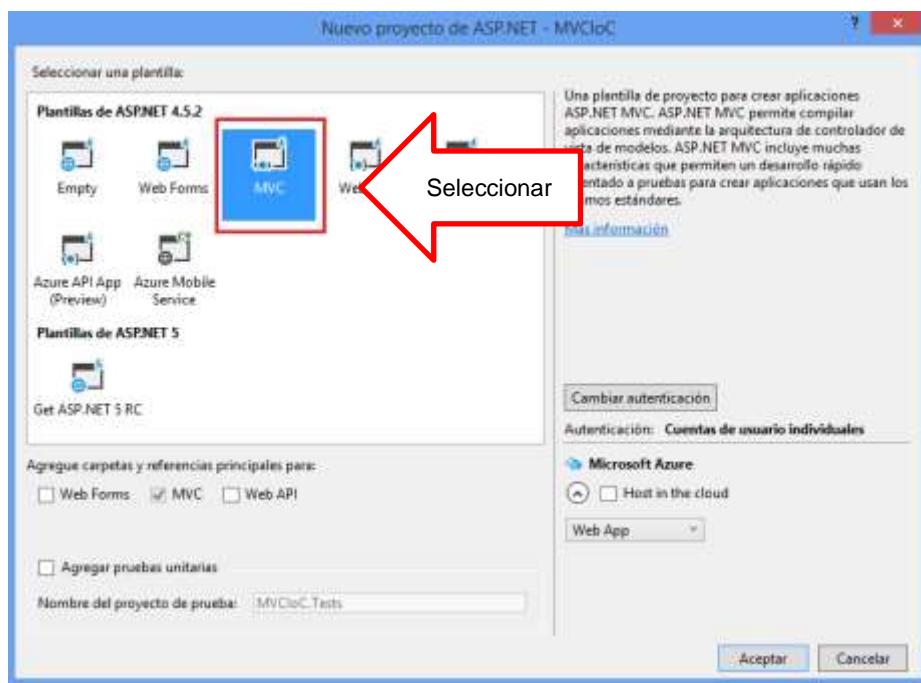


## Trabajando un proyecto en ASP.NET MVC

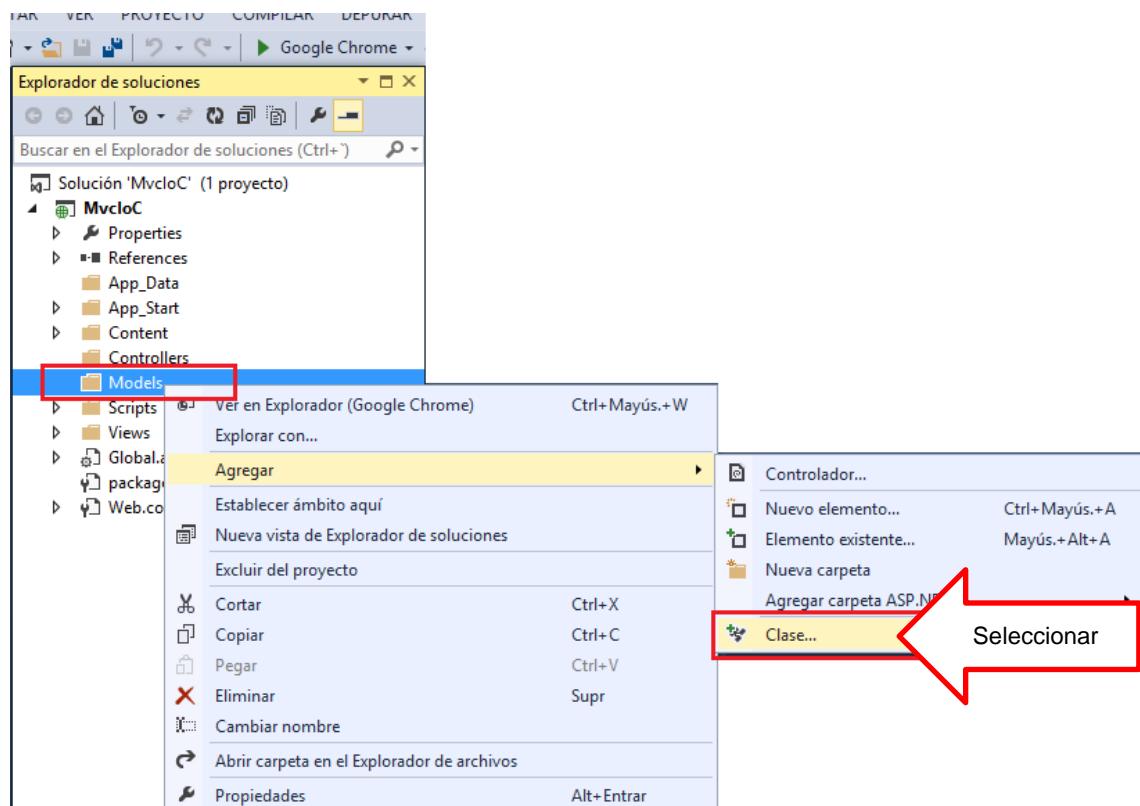
Crear un proyecto Web en C#, tal como se muestra



Selecciona la plantilla: Aplicación de Internet y el motor de Vista, tal como se muestra.

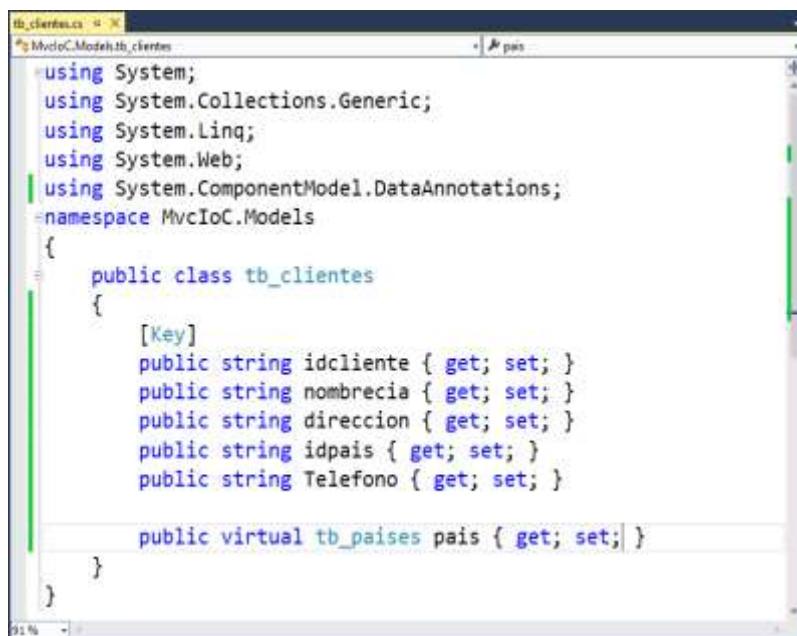


A continuación agregamos el modelo las clases del proyecto, tal como se muestra.



A continuación debemos crear las siguientes clases: tb\_clientes, tb\_paises, tal como se muestra en las siguientes imágenes.

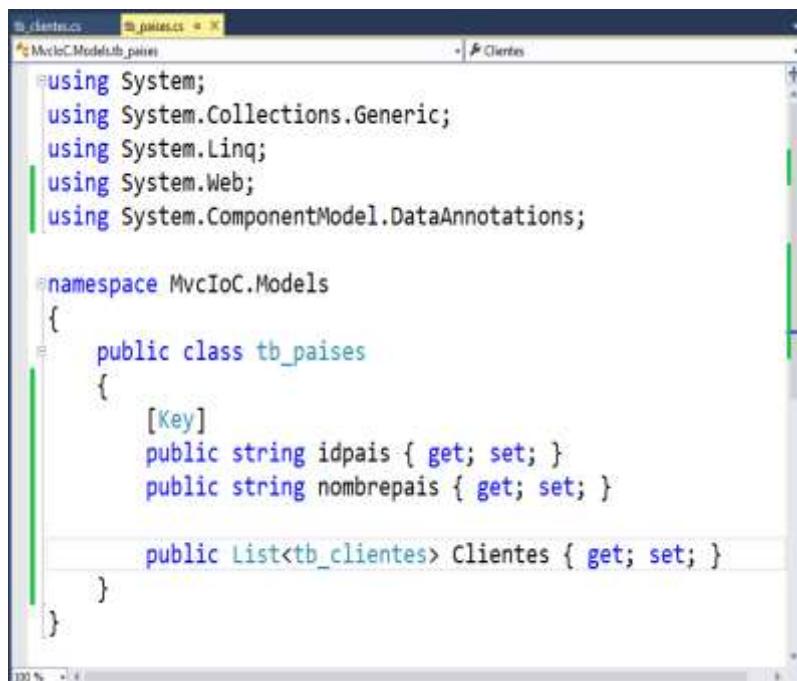
Definición de la clase tb\_paises



```
tb_clientes.cs  tb_paises.cs  ↻
MvcIoC.Models.tb_clientes  ↻  ↻
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
namespace MvcIoC.Models
{
    public class tb_clientes
    {
        [Key]
        public string idcliente { get; set; }
        public string nombrecia { get; set; }
        public string direccion { get; set; }
        public string idpais { get; set; }
        public string Telefono { get; set; }

        public virtual tb_paises pais { get; set; }
    }
}
```

Definición de la clase tb\_clientes

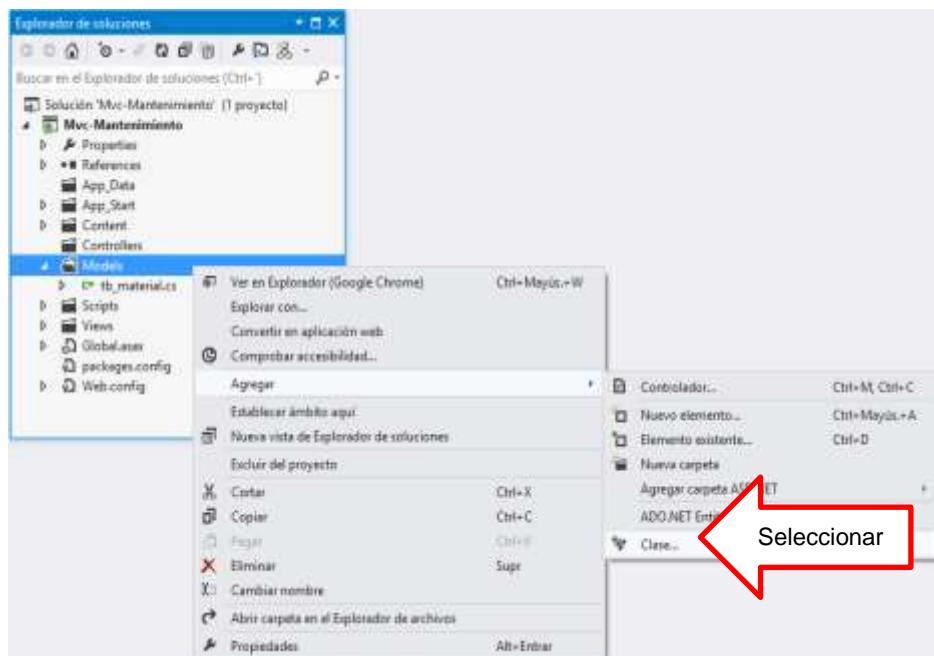


```
tb_clientes.cs  tb_paises.cs  ↻
MvcIoC.Models.tb_paises  ↻  ↻
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace MvcIoC.Models
{
    public class tb_paises
    {
        [Key]
        public string idpais { get; set; }
        public string nombrepais { get; set; }

        public List<tb_clientes> Clientes { get; set; }
    }
}
```

A continuación, vamos a agregar un modelo de Contexto: DbContext. Agregar una clase en la carpeta Models, tal como se encuentra



Importar las librerías, extienda la clase DbContext en Negocios2015Entities. A continuación defina los DbSet a cada una de las clases. Defina el método OnModelCreating para no pluralizar los nombres de las clases y sus tablas

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
namespace MvcIoC.Models
{
    public class Negocios2015Entities:DbContext
    {
        public DbSet<tb_clientes> Clientes { get; set; }
        public DbSet<tb_paises> Paises { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.
                Remove<PluralizingTableNameConvention>();
        }
    }
}

```

En el Web.config, publica la cadena de conexión. El nombre de la conexión es el mismo que el nombre del DbContext, defina el proveedor de datos: providerName=System.Data.SqlClient

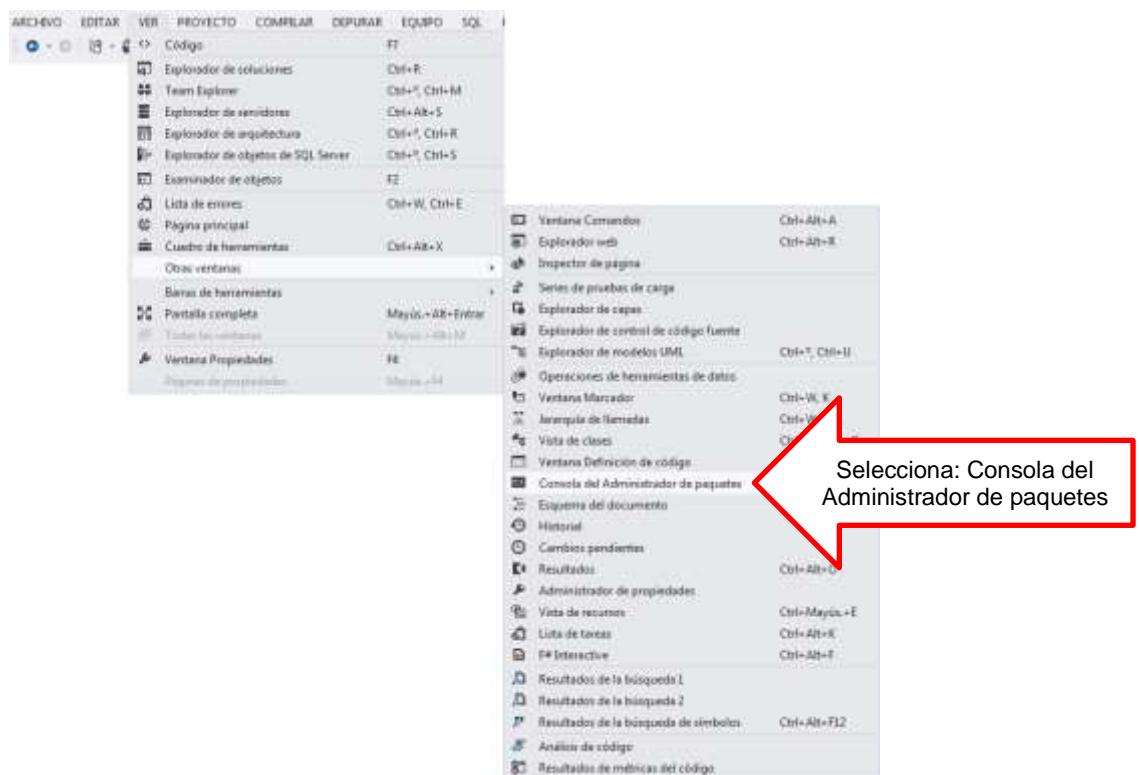


```

<configuration>
  <connectionStrings>
    <add name="DefaultConnection" providerName="System.Data.SqlClient" connectionString="server=.\\SQLEXPRESS; Database=Negocios2015; uid=sa; pwd=sq1" />
    <add name="Negocios2015Entities" providerName="System.Data.SqlClient" connectionString="server=.\\SQLEXPRESS; Database=Negocios2015; uid=sa; pwd=sq1" />
  </connectionStrings>
  ...

```

Para que los datos sean actualizados en el DbContext Negocios2014DB, abrimos la consulta del Administrador de paquetes



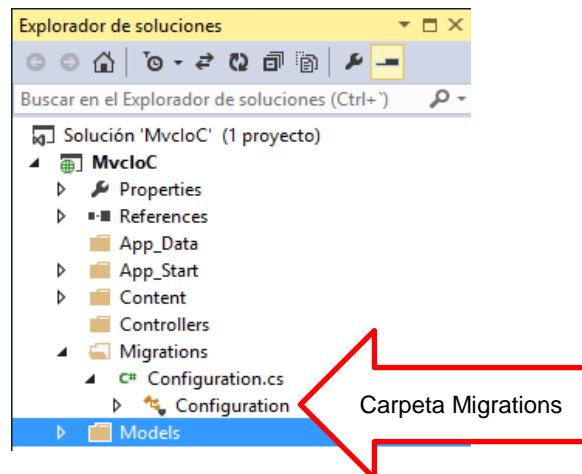
En el prompt del Package Manager (PM>) escribimos Enable-Migrations -ContextTypeName MvcIoC.Models.Negocios2015Entities, y presiona ENTER..

```
Console del Administrador de paquetes
Origen del proyecto: nuget.org | Proyecto predeterminado: MvcIoC
Cada uno de los paquetes se le otorga bajo licencia de su propietario respectivo. Microsoft no es + responsable de paquetes de terceros ni otorga ninguna licencia en relación con dichos paquetes.
Algunos paquetes pueden incluir dependencias que se rigen por licencias adicionales. Siga la dirección URL origen del paquete (fuente) para determinar cualquier dependencia.

PM> enable-migrations -contextTypeName MvcIoC.Models.Negocios2015Entities
Comprobando si el contexto indica una base de datos existente...
Se ha habilitado Migraciones de Code First para el proyecto MvcIoC.

PM>
```

Luego de la ejecución verificar que se haya creado la carpeta Migrations en el proyecto



Para no escribir los comandos de actualización por cada cambio que se hace al modelo, entonces activamos la actualización automática, para ello en la Clase Configuration escribimos en el constructor: AutomaticMigrationsEnabled = true;

```
Configuration.cs*  X  Web.config  Negocios2014DB.cs
MVCListado.Migrations.Configuration  Configuration()

namespace MVCListado.Migrations
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Migrations;
    using System.Linq;

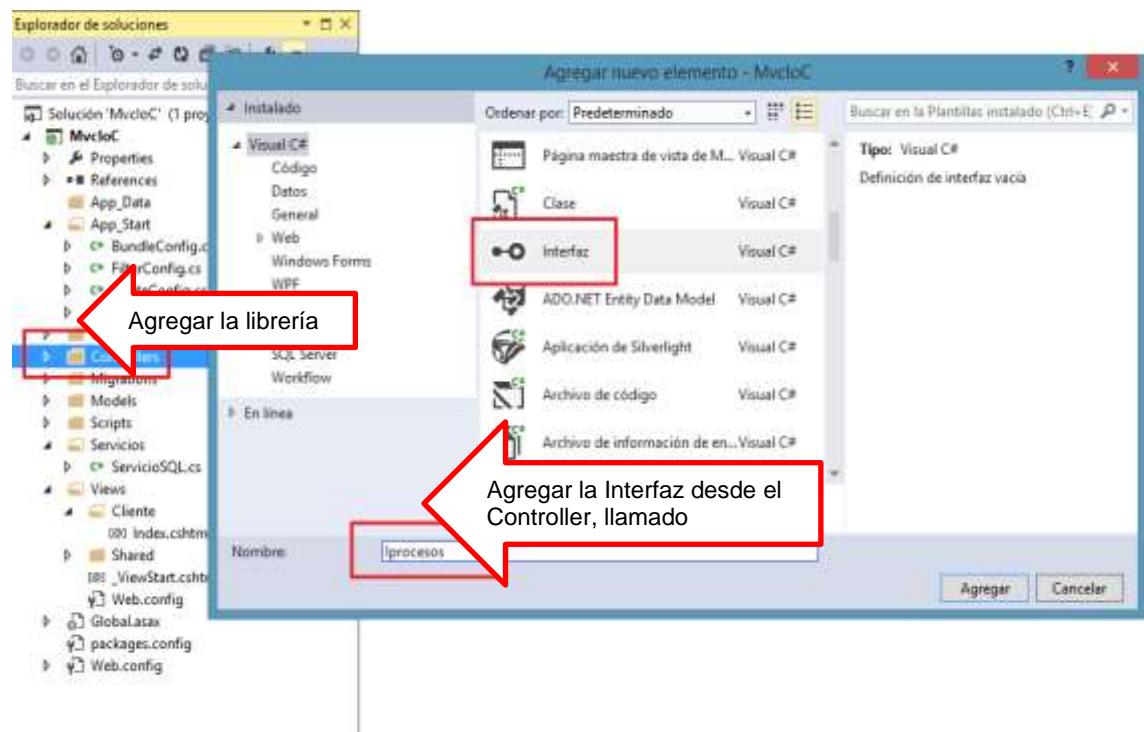
    internal sealed class Configuration : DbMigrationsConfiguration<MVCListado.Models.Negocios2014DB>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
        }

        protected override void Seed(MVCListado.Models.Negocios2014DB context)...
    }
}
```

A red box highlights the line 'AutomaticMigrationsEnabled = true;' with the text 'Modificar' (Modify).

## Inyección de Dependencia

El patrón Inyección de Dependencia sugiere crear una interfaz con los métodos necesitados, para que las clases implementen la funcionalidad requerida, por tanto a nuestro proyecto agregamos una Interfaz llamada IProcesos defina los métodos, tal como se muestra

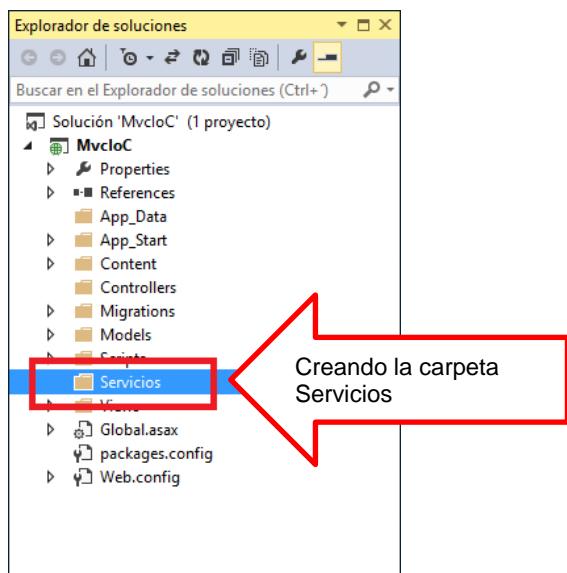


```
+MvcIoC.Controllers.IProcesos
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Linq.Expressions;
using MvcIoC.Models;

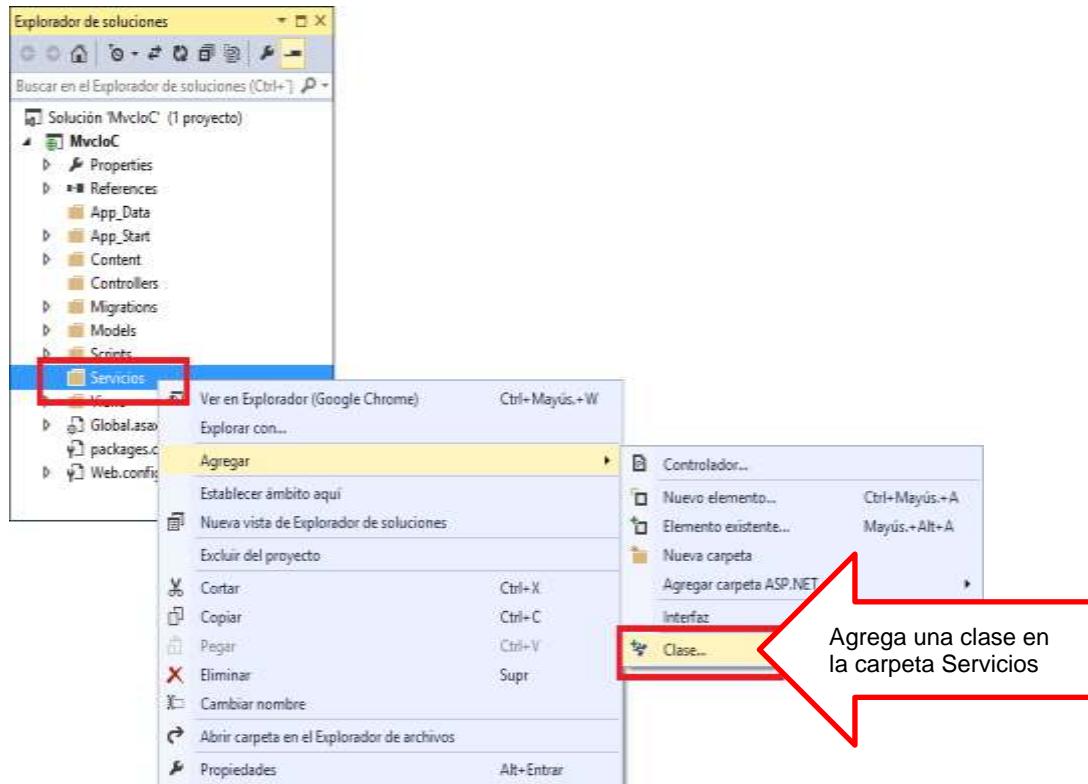
namespace MvcIoC.Controllers
{
    public interface IProcesos
    {
        List<tb_clientes> Listado();
        List<tb_clientes> Consulta(Expression<Func<tb_clientes, bool>> predicado);
        tb_clientes Buscar(Expression<Func<tb_clientes, bool>> predicado);
        void Agregar(tb_clientes entidad);
        void Actualizar(tb_clientes entidad);
        void Eliminar(tb_clientes entidad);
    }
}
```

A red box highlights the 'IProcesos' interface definition in the code editor, with the annotation 'Definiendo la interface' (Defining the interface).

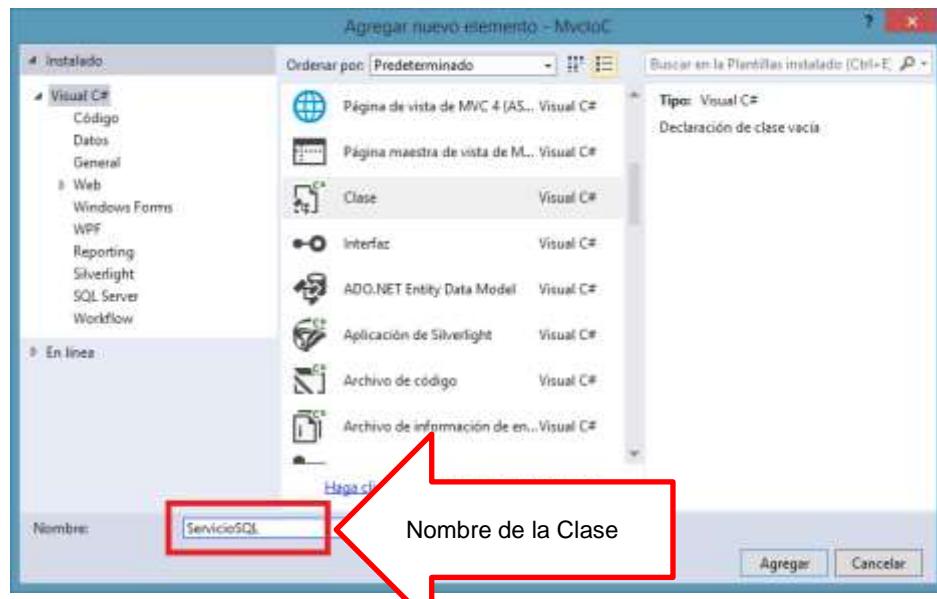
Para separar el modelo del controlador, primero creamos una carpeta, la cual llamaremos servicios, tal como se muestra.



En la carpeta Servicios, agrega una clase, tal como se muestra



Asignar el nombre a la clase: ServicioSQL, tal como se muestra, presiona el botón AGREGAR



Implementa la clase ServicioSQL con la interface IPproceso, donde se visualizan los métodos a ser implementados

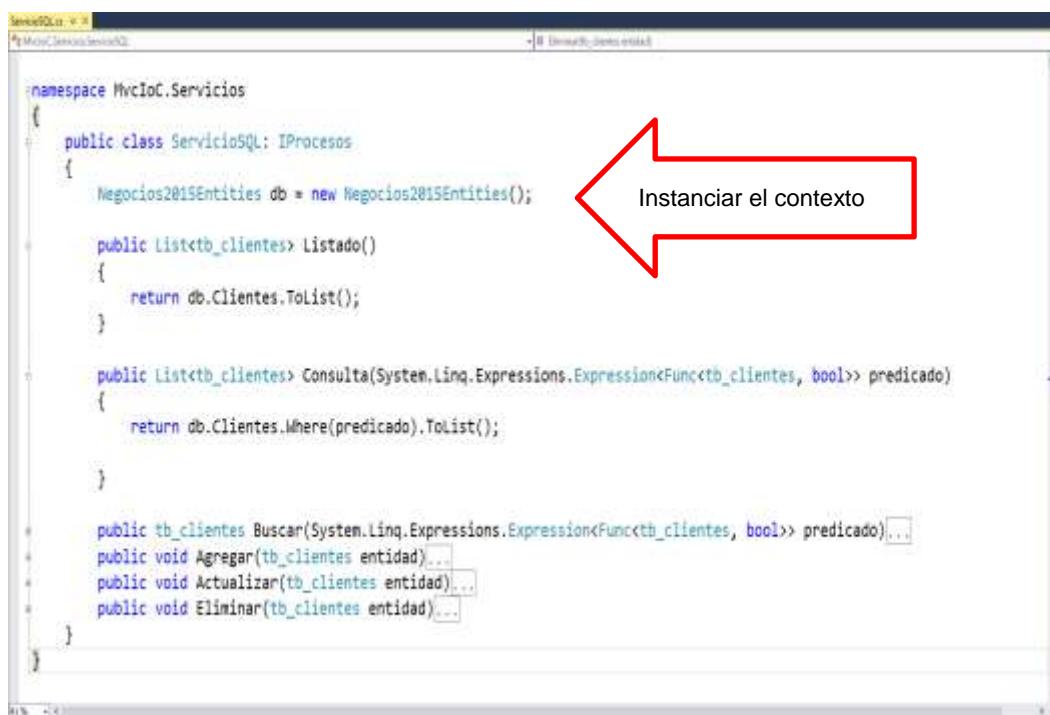
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using MvcIoC.Controllers;
using MvcIoC.Models;
using System.Data.Entity;

namespace MvcIoC.Servicios
{
    public class ServicioSQL : IPproceso
    {
        public List<tb_clientes> Listado()...
        public List<tb_clientes> Consulta(System.Linq.Expressions.Expression<Func<tb_
        public tb_clientes Buscar(System.Linq.Expressions.Expression<Func<tb_
        public void Agregar(tb_clientes entidad)...
        public void Actualizar(tb_clientes entidad)...
        public void Eliminar(tb_clientes entidad)...
    }
}

```

A continuación implementamos la clase tal como se muestra: Instanciamos el contexto e implementamos los métodos de la clase



```

namespace MvcIoC.Servicios
{
    public class ServicioSQL : IProcesos
    {
        Negocios2015Entities db = new Negocios2015Entities();

        public List<tb_clientes> Listado()
        {
            return db.Clientes.ToList();
        }

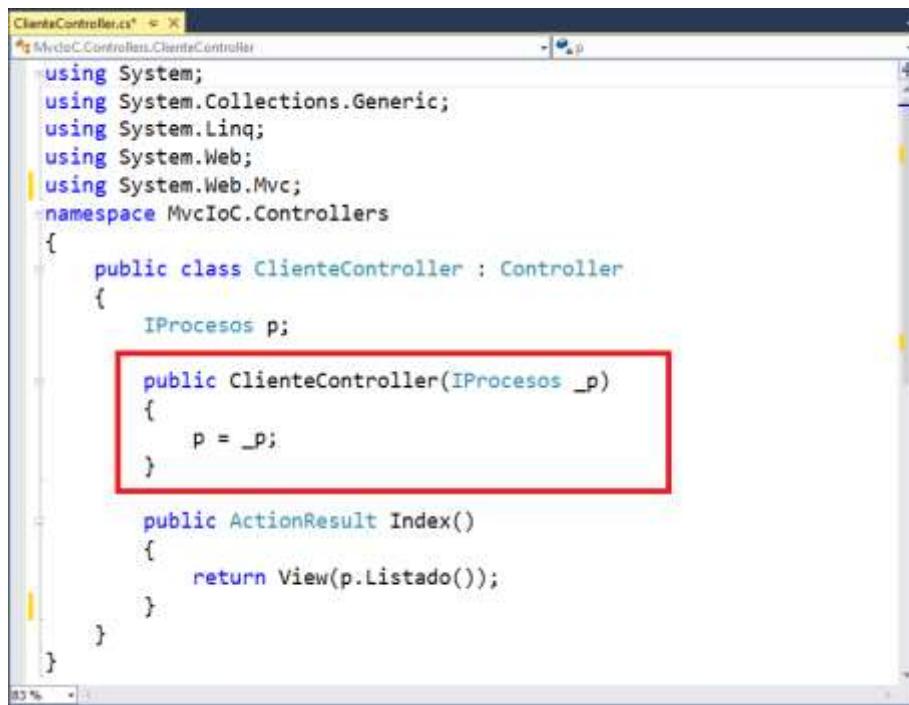
        public List<tb_clientes> Consulta(System.Linq.Expressions.Expression<Func<tb_clientes, bool>> predicado)
        {
            return db.Clientes.Where(predicado).ToList();
        }

        public tb_clientes Buscar(System.Linq.Expressions.Expression<Func<tb_clientes, bool>> predicado)...
        public void Agregar(tb_clientes entidad)...
        public void Actualizar(tb_clientes entidad)...
        public void Eliminar(tb_clientes entidad)...
    }
}

```

Terminado este proceso, creamos el controlador Cliente. Defina la interface del proceso para ser ejecutado por el servicioSQL: instanciar el Controlador tal como se muestra.

Defina el ActionResult Index y agregue su vista



```

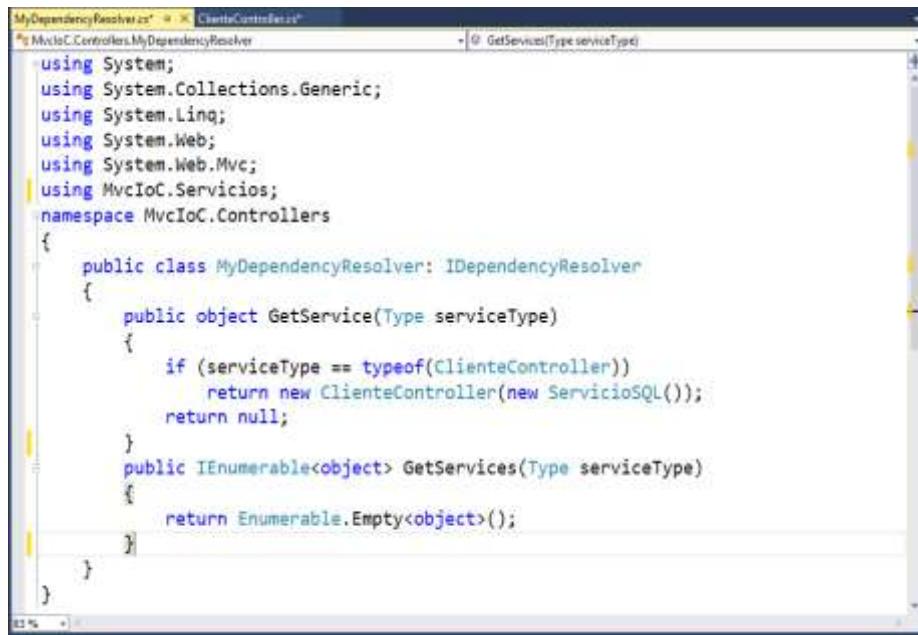
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace MvcIoC.Controllers
{
    public class ClienteController : Controller
    {
        IProcesos p;

        public ClienteController(IProcesos _p)
        {
            p = _p;
        }

        public ActionResult Index()
        {
            return View(p.Listado());
        }
    }
}

```

A continuación en el controlador defina la clase MyDependencyResolver, implementada por IDependencyResolver, el cual ejecuta la instancia del controlador, enviándole como parámetro el ServicioSQL()

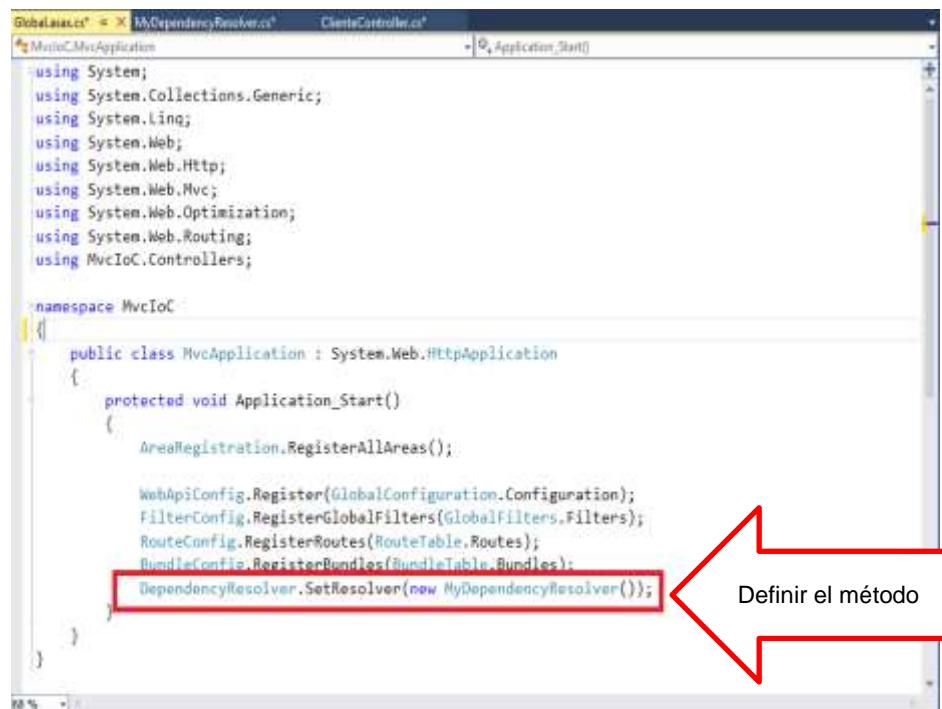


```

MyDependencyResolver.cs  ClientController.cs
MvcIoC.Controllers.MyDependencyResolver
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcIoC.Servicios;
namespace MvcIoC.Controllers
{
    public class MyDependencyResolver : IDependencyResolver
    {
        public object GetService(Type serviceType)
        {
            if (serviceType == typeof(ClienteController))
                return new ClienteController(new ServicioSQL());
            return null;
        }
        public IEnumerable<object> GetServices(Type serviceType)
        {
            return Enumerable.Empty<object>();
        }
    }
}

```

Abrir el Global.asax y publicar el método para que el sistema cargue los datos desde el servicio



```

Global.asax.cs  MyDependencyResolver.cs  ClientController.cs
MvcIoC.MvcApplication
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;
using MvcIoC.Controllers;

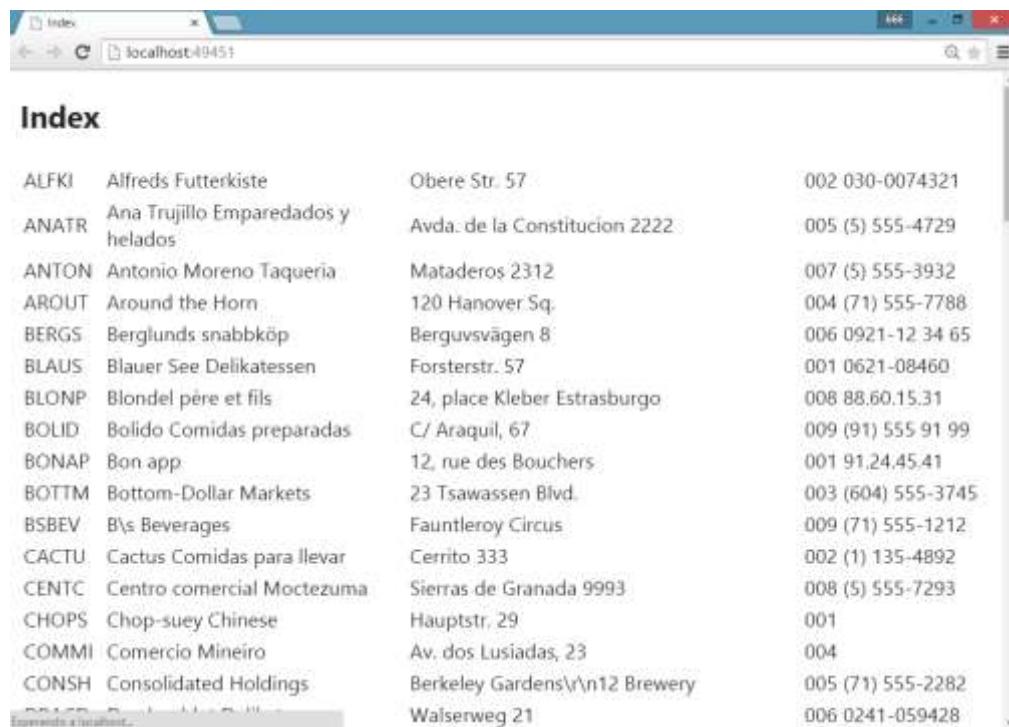
namespace MvcIoC
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();

            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            DependencyResolver.SetResolver(new MyDependencyResolver());
        }
    }
}

```

Definir el método

Ejecute el proyecto, donde se visualiza el listado de Clientes.



The screenshot shows a Microsoft Edge browser window with the title bar "Index" and the URL "localhost:40451". The main content area displays a table with 20 rows of customer information. The columns are: Customer ID (e.g., ALFKI, ANATR, ANTON, AROUT, BERGS, BLAUS, BLONP, BOLID, BONAP, BOTTM, BSBEV, CACTU, CENTC, CHOPS, COMM, CONSH), Company Name (e.g., Alfreds Futterkiste, Ana Trujillo Emparedados y helados, Antonio Moreno Taquería, Around the Horn, Berglunds snabbköp, Blauer See Delikatessen, Blondel père et fils, Bolido Comidas preparadas, Bon app, Bottom-Dollar Markets, B's Beverages, Cactus Comidas para llevar, Centro comercial Moctezuma, Chop-suey Chinese, Comercio Mineiro, Consolidated Holdings), Address (e.g., Obere Str. 57, Avda. de la Constitucion 2222, Mataderos 2312, 120 Hanover Sq., Berguvsvägen 8, Forsterstr. 57, 24, place Kleber Estrasburgo, C/ Araquil, 67, 12, rue des Bouchers, 23 Tsawassen Blvd., Fauntleroy Circus, Cerrito 333, Sierras de Granada 9993, Hauptstr. 29, Av. dos Lusiadas, 23, Berkeley Gardens\n12 Brewery, Waisenweg 21), and Phone Number (e.g., 002 030-0074321, 005 (5) 555-4729, 007 (5) 555-3932, 004 (71) 555-7788, 006 0921-12 34 65, 001 0621-08460, 008 88.60.15.31, 009 (91) 555 91 99, 001 91.24.45.41, 003 (604) 555-3745, 009 (71) 555-1212, 002 (1) 135-4892, 008 (5) 555-7293, 001, 004, 005 (71) 555-2282, 006 0241-059428). The table has a light gray background with alternating row colors.

ALFKI	Alfreds Futterkiste	Obere Str. 57	002 030-0074321
ANATR	Ana Trujillo Emparedados y helados	Avda. de la Constitucion 2222	005 (5) 555-4729
ANTON	Antonio Moreno Taquería	Mataderos 2312	007 (5) 555-3932
AROUT	Around the Horn	120 Hanover Sq.	004 (71) 555-7788
BERGS	Berglunds snabbköp	Berguvsvägen 8	006 0921-12 34 65
BLAUS	Blauer See Delikatessen	Forsterstr. 57	001 0621-08460
BLONP	Blondel père et fils	24, place Kleber Estrasburgo	008 88.60.15.31
BOLID	Bolido Comidas preparadas	C/ Araquil, 67	009 (91) 555 91 99
BONAP	Bon app	12, rue des Bouchers	001 91.24.45.41
BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	003 (604) 555-3745
BSBEV	B's Beverages	Fauntleroy Circus	009 (71) 555-1212
CACTU	Cactus Comidas para llevar	Cerrito 333	002 (1) 135-4892
CENTC	Centro comercial Moctezuma	Sierras de Granada 9993	008 (5) 555-7293
CHOPS	Chop-suey Chinese	Hauptstr. 29	001
COMM	Comercio Mineiro	Av. dos Lusiadas, 23	004
CONSH	Consolidated Holdings	Berkeley Gardens\n12 Brewery	005 (71) 555-2282
		Waisenweg 21	006 0241-059428

## Resumen

- └─ La Inversión de Control es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así el reuso de los mismos.
- └─ Actualmente existen dos técnicas de implementación para el IoC: Inyección de dependencias y Service Locutor; en este manual nos enfocaremos en la implementación de la Inyección de Dependencia (DI).
- └─ Service Locator es un patrón que nos permite localizar servicios. Es una clase, el ServiceLocator a la cual le podemos pedir instancias de un servicio concreto. Esto es útil para poder tener distintas implementaciones de un mismo servicio y cambiar, mediante configuración, la implementación que queremos que devuelva el ServiceLocator cuando le pidamos la instancia del servicio. ServiceLocator actúa como un catálogo central de instancias de servicios al que le podemos solicitar la instancia del servicio que necesitemos.
- └─ Para un localizador más sofisticado puedo heredar del Localizador de Servicios y pasar esta subclase en la variable de la clase de registro. Puedo cambiar los métodos estáticos para llamar a un método en la instancia en lugar de acceder directamente a las variables de instancia. Puedo proporcionar localizadores para un hilo específico usando una ubicación de almacenamiento específica para el hilo.
- └─ Una dependencia entre un componente y otro, puede establecerse estáticamente o en tiempo de compilación, o bien, dinámicamente o en tiempo de ejecución. Es en éste último escenario es donde cabe el concepto de inyección, y para que esto fuera posible, debemos referenciar interfaces y no implementaciones directas.
- └─ Lo que propone entonces la Inyección de dependencias, es no instanciar las dependencias explícitamente en su clase, sino que declarativamente expresarlas en la definición de la clase. La esencia de la inyección de las dependencias es contar con un componente capaz de obtener instancias válidas de las dependencias del objeto y pasárselas durante la creación o inicialización del objeto.
- └─ En la inyección basada en un constructor, se creará una instancia de BusinessFacade usando un constructor parametrizado al cual se le pasará una referencia de un IBusinessLogic para poder inyectar la dependencia

# **APÉNDICE**

## JQUERY Y AJAX

### Introducción al Jquery

jQuery es la librería JavaScript que ha irrumpido con más fuerza como alternativa a Prototype. Su autor original es John Resig, aunque como sucede con todas las librerías exitosas, actualmente recibe contribuciones de decenas de programadores. jQuery también ha sido programada de forma muy eficiente y su versión comprimida apenas ocupa 20 KB.

jQuery comparte con Prototype muchas ideas e incluso dispone de funciones con el mismo nombre. Sin embargo, su diseño interno tiene algunas diferencias drásticas respecto a Prototype, sobre todo el "encadenamiento" de llamadas a métodos.

```

_Layout.cshtml ▾ X
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    <script src="~/Scripts/jquery-1.7.1.js"></script>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
  </head>
  <body>
    @RenderBody()
    @Scripts.Render("~/bundles/jquery")
    @RenderSection("scripts", required: false)
  </body>
</html>

```

### Estructura de programación en Jquery

El símbolo \$ indica que este es una sentencia de jQuery, puede ser reemplazada por la palabra jQuery.

El evento **.ready()** tiene como finalidad ejecutar una función inmediatamente después de cargar todo el documento HTML y su DOM correspondiente, garantizando que el código sea ejecutado sobre los elementos que ya hayan sido desplegados.

En este script de código, invocamos el objeto jQuery con el parámetro "document" y le paso una función anónima para ser ejecutada cuando se dispare el evento "ready".

```

Index.cshtml ▾ X
@{
  ViewBag.Title = "Index";
}
<script type="text/javascript">
$(document).ready(function () {
  alert("Hola Mundo");
});
</script>

<h2>Index</h2>

```

### **Selectores**

Permiten obtener el contenido del documento para ser manipularlo. Al utilizarlos, los selectores retornan un arreglo de objetos que coinciden con los criterios especificados.

Este arreglo no es un conjunto de objetos del DOM, son objetos de jQuery con un gran número de funciones y propiedades predefinidas para realizar operaciones con los mismos.

Los selectores básicos están basados en la sintaxis CSS y funcionan mas o menos de la misma manera:

Select	Descripción
nombre de etiqueta	Encuentra elementos por etiqueta HTML
#id	Encuentra elementos por ID o identificador
.clase	Encuentra elementos por clase
etiqueta.clase	Encuentra elementos del tipo de la etiqueta que tenga la clase "clase"
etiqueta#id.clase	Encuentra los elementos del tipo de la etiqueta que tienen el ID y la clase
*	Encuentra todos los elementos de la página

### **Filtros**

Los filtros se utilizan para proveer un mayor control sobre como los elementos son seleccionados en el documento.

Los filtros en jQuery vienen en 6 categorías distintas: Básicos, Contenido, visibilidad, atributo, hijo, formulario. Entre los filtros básicos tenemos:

Select	Descripción
:first	Selecciona solo el primero de los elementos en la lista
:last	Selecciona solo el ultimo de los elementos en la lista
:even	Selecciona solo los elementos en posiciones pares de la lista
:odd	Selecciona solo los elementos en posiciones impares de la lista
:eq(n)	Obtiene elementos que están solo en el índice especificado
:gt(n)	Incluye elementos que están después del índice especificado
:lt(n)	Incluye elementos que están antes del índice especificado
:header	Selecciona todos los elementos tipo encabezado (H1, H2, etc.)
:animated	Selecciona todos los elementos que están siendo animados
:not(selector)	Incluye todos los elementos que no cumplen con el selector proporcionado

### **Manipulando contenido**

Cuando seleccionamos y filtramos contenido de una página web, lo hacemos normalmente porque queremos hacer algo con el: crear nuevo contenido y agregarlo dinámicamente a la página.

jQuery tiene funciones para crear, copiar, eliminar y mover contenido, incluso para envolver elementos dentro de otros. También provee soporte para trabajar con css.

Los métodos html() y text() permiten obtener y asignar contenido:

- **html()**: retorna el HTML contenido en el primer elemento seleccionado.
- **html(htmlString)**: le asigna el valor de la variable htmlString como contenido HTML a todos los elementos encontrados.
- **text()**: retorna el texto contenido en el primer elemento seleccionado.
- **text(htmlString)**: le asigna el valor de la variable htmlString como texto a todos los elementos encontrados.

Si pasamos HTML la función text(), el código será escapado y mostrado como texto.

### Manipulando Atributos

jQuery permite la manipulación de atributos de uno o varios elementos HTML mediante las siguientes funciones:

- attr(nombre): Retorna el valor del atributo "nombre" del elemento seleccionado.
- attr({nombre: valor}): Asigna varios atributos del elemento seleccionado. Para asignar los atributos se usa la notación de objeto de javascript (JSON).
- attr(nombre, valor): Asigna "valor" al atributo "nombre" del elemento seleccionado.
- removeAttr(nombre): Elimina el atributo "nombre" del elemento seleccionado.

Un ejemplo es asignarle a la etiqueta <img> los atributos del origen y alt



```

Index.cshtml
@{
    ViewBag.Title = "Index";
}
<script type="text/javascript">
    $(document).ready(function () {
        $('img').attr('src', '../imagenes/teclado.jpg');
        $('img').attr('alt', 'Teclado');
    });
</script>

<h2>Index</h2>
<img />

```

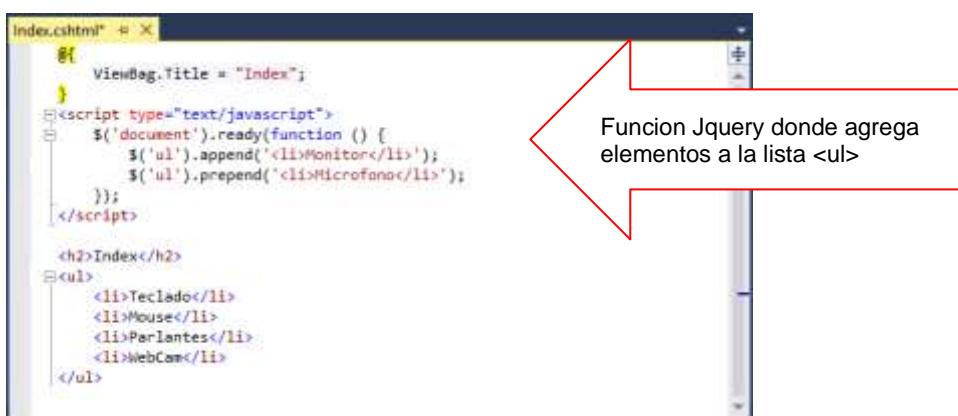
Fucion Jquery donde asigna propiedades a la etiqueta <img>

### Insertando Contenido

Las siguientes funciones permiten para agregar contenido a los elementos seleccionados:

- append(contenido): agrega el contenido dentro del los elementos seleccionados.
- appendTo(selector): agrega el contenido a otros elementos especificados.
- prepend(contenido): agrega el contenido de primero dentro de los elementos seleccionados.
- prependTo(selector): agrega el contenido de primero a otros elementos especificados.
- after(contenido): agrega el contenido después del elemento seleccionado.
- before(contenido): agrega el contenido antes del elemento seleccionado.
- insertAfter(selector): agrega el contenido después de otros elementos elementos seleccionados.
- insertBefore(contenido): agrega el contenido antes de otros elementos elementos seleccionados.

En el siguiente ejemplo, agregamos elementos a la lista <ul>



```

Index.cshtml
@{
    ViewBag.Title = "Index";
}
<script type="text/javascript">
    $(document).ready(function () {
        $('#ul').append('<li>Monitor</li>');
        $('#ul').prepend('<li>Microfono</li>');
    });
</script>

<h2>Index</h2>
<ul>
    <li>Teclado</li>
    <li>Mouse</li>
    <li>Parlantes</li>
    <li>WebCam</li>
</ul>

```

Fucion Jquery donde agrega elementos a la lista <ul>

### **Manejo de eventos**

jQuery define una lista de eventos y funciones para la administración de los mismos, la sintaxis por defecto para un manejador de evento es de la siguiente forma `$ fn.nombreEvento`.

```
$( 'Selector' ).nombreEvento(
    function (event) {
});
```

Entre los eventos más comunes:

Evento	Descripción
.blur()	Se lanza sobre un elemento que acaba de perder el foco. Aplicable a los inputs de formularios
.click()	Se lanza cuando pinchamos sobre el elemento que hemos asociado el evento.
.dblclick()	Se lanza cuando hay un doble click sobre el elemento
.focus()	Evento que permite saber cuando un elemento recibe el foco.
.hover()	Se lanza cuando el mouse esta encima del elemento del evento.
.keydown()	Se lanza cuando el usuario pulsa una tecla
.keypress()	Se lanza cuando se mantiene presionada la tecla, es decir, se lanza cada vez que se escriba el carácter.
.load()	Se lanza tan pronto el elemento ha terminado de cargarse por completo.
.mousedown()	Se lanza cuando pinchamos un elemento
.mousemove()	Se lanza cuando el mouse esta encima del elemento.
.mouseover()	Se lanza cuando el mause entra por primera vez en el elemento.
.one()	Igual que el bind() pero el evento se ejecuta una vez
.select()	Se lanza cuando el usuario selecciona un texto.
.toggle()	Se utiliza para generar comportamientos de cambio de estado generados al pinchar sobre un elemento

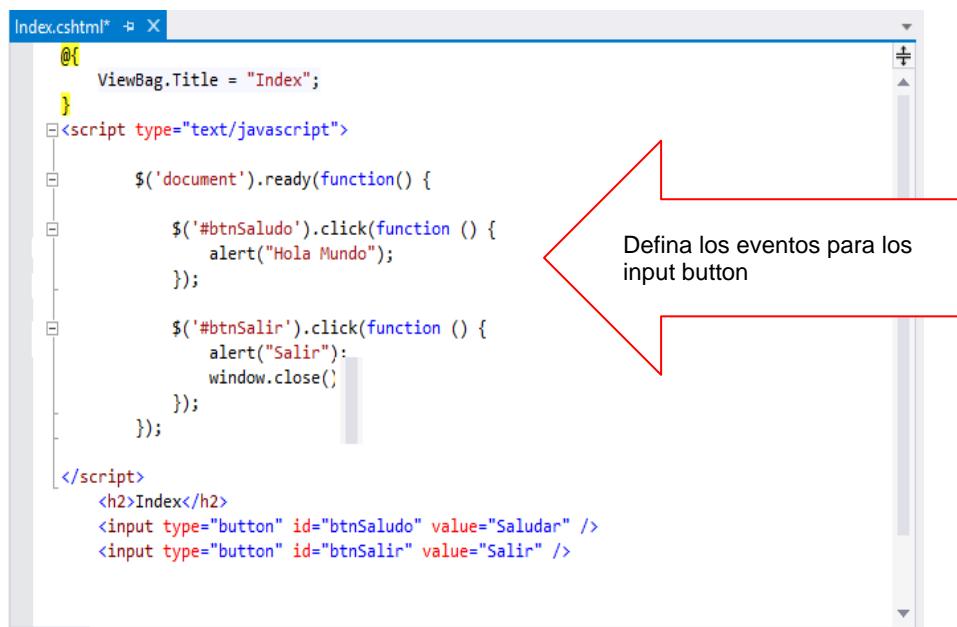
### **Atributos del objeto Event**

El objeto Event es pasado a todos los eventos que se lanzan y pone a nuestra disposición una serie de atributos muy útiles a la hora de trabajar con eventos.

Atributo	Descripcion
event.currentTarget	Devuelve el elemento sobre el que se ha lanzado el evento. Por ejemplo, si el evento es un onclick de un enlace, el currentTarget sería el enlace.
event.data	Devuelve los datos que hayamos podido pasar al evento cuando se asocia con bind
event.isDefaultPrevented()	Devuelve si se ha lanzado el método preventDefault() o no.

event.isImmediatePropagationStopped()	Devuelve si el método stopImmediatePropagation() se ha llamado, o no, en este objeto.
event.isPropagationStopped()	Devuelve si el método stopPropagation() ha sido llamado
event.pageX	Devuelve la posición relativa del ratón en relación a la esquina izquierda del documento. Esta propiedad es muy útil cuando trabajamos con efectos.
event.pageY	Devuelve la posición relativa del ratón con respecto a la esquina superior del documento.
event.preventDefault()	Si llamamos a este método dentro de un evento, la acción predeterminada que se ejecutaría por este evento nunca será ejecutada.
event.stopImmediatePropagation()	Previene que se ejecuten otras acciones que pudieran estar asociadas al evento.
event.stopPropagation()	Previene que se ejecute cualquier evento que pudiera estar asociado a los padres del elemento dentro del árbol DOM.
event.target	Es el elemento DOM que inició el evento
event.timeStamp	Número en milisegundos desde el 1 de enero de 1970, desde que el evento fue lanzado. Esto podría ayudarnos para realizar pruebas de rendimiento de nuestros scripts.
event.which	Para eventos de teclado y ratón, este atributo indica el botón o la tecla que ha sido pulsada.

En el siguiente ejemplo definidos el evento click para los botones Saludos y Salir utilizando jquery



```

@{
    ViewBag.Title = "Index";
}
<script type="text/javascript">
    $(document).ready(function () {
        $('#btnSaludo').click(function () {
            alert("Hola Mundo");
        });

        $('#btnSalir').click(function () {
            alert("Salir");
            window.close();
        });
    });
</script>
<h2>Index</h2>
<input type="button" id="btnSaludo" value="Saludar" />
<input type="button" id="btnSalir" value="Salir" />

```

Defina los eventos para los input button

### Recorrer los elementos del proyecto

Para iterar sobre la información obtenida del documento disponemos de las siguientes funciones:

- **size()**: Retorna el numero de elementos en la lista de resultados. También se puede obtener a través de la propiedad length;
- **get()**: Retorna una lista de elementos del DOM. Esta función es útil cuando se necesitan hacer operaciones en el DOM en lugar de usar funciones de jQuery.

- **get(posición)**: Retorna un elemento del DOM que esta en la posición especificada.
- **find({expresión})**: Busca elementos que cumplen con la expresión especificada.
- **each(callback(i, element))**: Ejecuta una función dentro del contexto de cada elemento seleccionado. Ejecuta un callback recibiendo como parámetro la posición de cada elemento y el propio elemento.

## Jquery y Ajax

AJAX significa Asynchronous JavaScript and XML. Esta tecnología nos permite comunicarnos con un servicio web sin tener que recargar la página. Con jQuery, hacer uso de AJAX es muy sencillo. JQuery provee varias funciones para trabajar con AJAX. La mas común es usar **\$.ajax()**.



```

Index.cshtml * X
[{
    ViewBag.Title = "Index";
}
<script type="text/javascript">
    function test2() {
        var jajax = $.getJSON("http://api.openweathermap.org/data/2.1/weather/city/caracas?callback=?", function () {
            alert("success");
        })
        .done(function () {
            alert("second success");
        })
        .fail(function () {
            alert("error");
        });
    }
</script>
<h2>Index</h2>
<input type="button" onclick="test2()" value="Testear" />

```

### Parametros de la función Ajax

- **url**: La dirección a donde enviar la solicitud.
- **type**: Tipo de request (solicitud). Ejemplo: GET, POST, PUT, DELETE, etc. En caso de utilizar POST o PUT, por ejemplo, se puede enviar un objeto en otro parámetro a la misma función llamado data. Ej: data: {'clave': 'valor'},
- **datatype**: El tipo de respuesta que se espera, en este caso es json.

### Funciones

**\$.get()**: Realiza una llamada GET a una dirección específica. Esta función nos indica si la operación fue exitosa y ha tenido errores.

```

$.get("http://myURL.com/", function () {
    alert("Proceso Ejecutado");
})
.done(function () {
    alert("Funciona");
})
.fail(function () {
    alert("Este proceso tiene errores");
});

```

**\$.getJSON()**: es muy similar a la anterior, solo que es específica para cuando se espera una respuesta tipo json. Para esta función se debe agregar 'callback=?' a la url.

```
$.getJSON("http://api.openweathermap.org/data/2.1/weather/city/caracas?callback=?", function () {
    alert("Exito");
});
```

`$.getScript()`: Carga un archivo Javascript a una dirección específica.

```
$.getScript("http://myURL.com/ajax/myScript.js", function (script, textStatus, jqxhr) {
    alert("Exito");
});
```

`$.post()`: realiza una llamada POST a una dirección URL

```
$.post("http://myURL.com/usuario", { 'nombre': 'Oscar', 'apellido': 'ramirez' },
function (data, textStatus, jqxhr) {

    alert("Exito");
});
```

`$.load()`: carga una dirección url y coloca los datos retornados en los elementos seleccionados.

```
$( 'body' ).load("http://myURL.com/public/datos.txt");
```

### **Eventos Globales**

Jquery ofrece un conjunto de funciones que se invocan automáticamente cuando se dispara su evento correspondiente.

`$.ajaxComplete()`: es llamada cuando una función AJAX es completada

```
$( 'document' ).ajaxComplete(function () {
    alert('Ajax completado');
});
```

`$.ajaxError()`: es llamada cuando una función AJAX es completada pero con errores

```
$( 'document' ).ajaxError(function () {
    alert('Error en el Proceso');
});
```

`$.ajaxSend()`: se invoca cuando un AJAX es enviado

```
$( 'document' ).ajaxSend(function () {
    alert('Enviado');
});
```

\$.ajaxStart(): JQuery lleva un control de todas las llamadas AJAX que ejecutas. Si ninguna está en curso, esta función es invocada.

```
$(document).ajaxStart(function () {
    alert('Iniciando');
});
```

\$.ajaxStop(): se invoca cada vez que una función AJAX es completada y quedas otras en curso. Incluso es invocada cuando la ultima función AJAX es cancelada.

```
$(document).ajaxStop(function () {
    alert('Detenido');
});
```

\$.ajaxSuccess(): es invocada cuando una función AJAX termina exitosamente.

```
$(document).ajaxSuccess(function () {
    alert('Detenido');
});
```

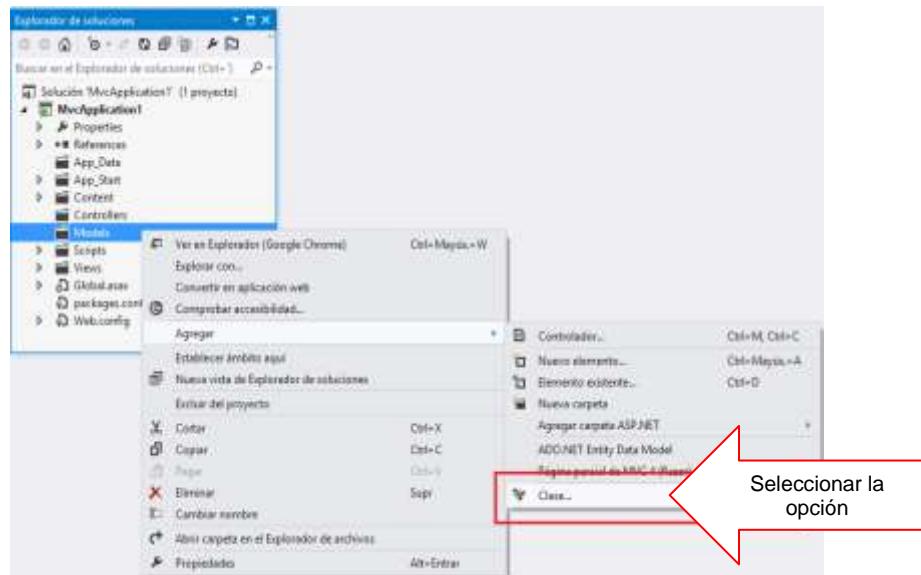
## Laboratorio

### Creando una aplicación ASP.NET MVC y Jquery

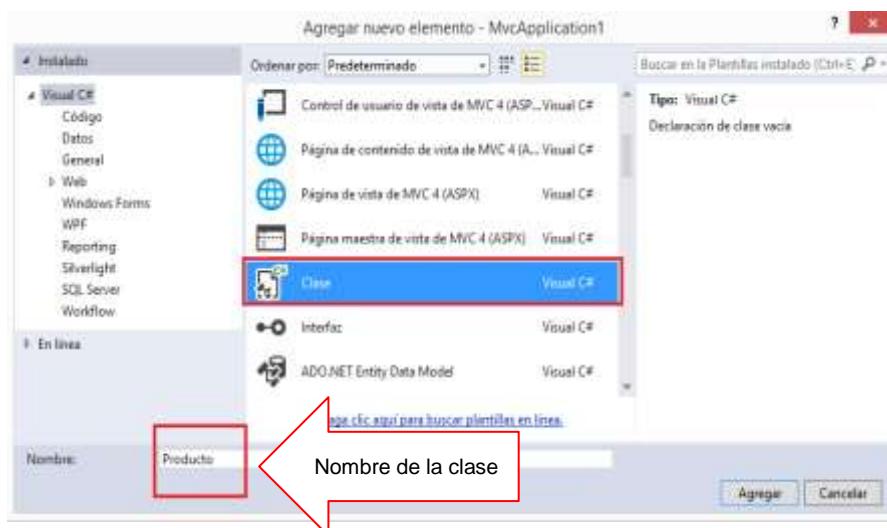
Implemente un proyecto ASP.NET MVC donde permita realizar operaciones de listado y actualización de productos utilizando anotaciones y ventanas de dialogo en jquery.

#### Trabajando con el Modelo

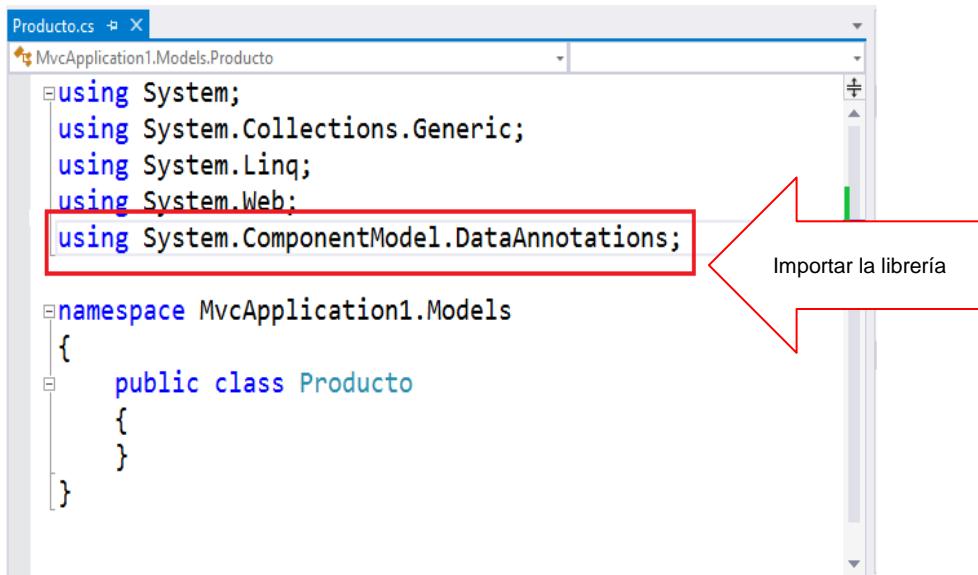
Primeramente definimos la clase en la carpeta Models: Agregar una clase llamada Producto



En la plantilla selecciona Clase y asigne su nombre: Producto, presiona el botón Agregar



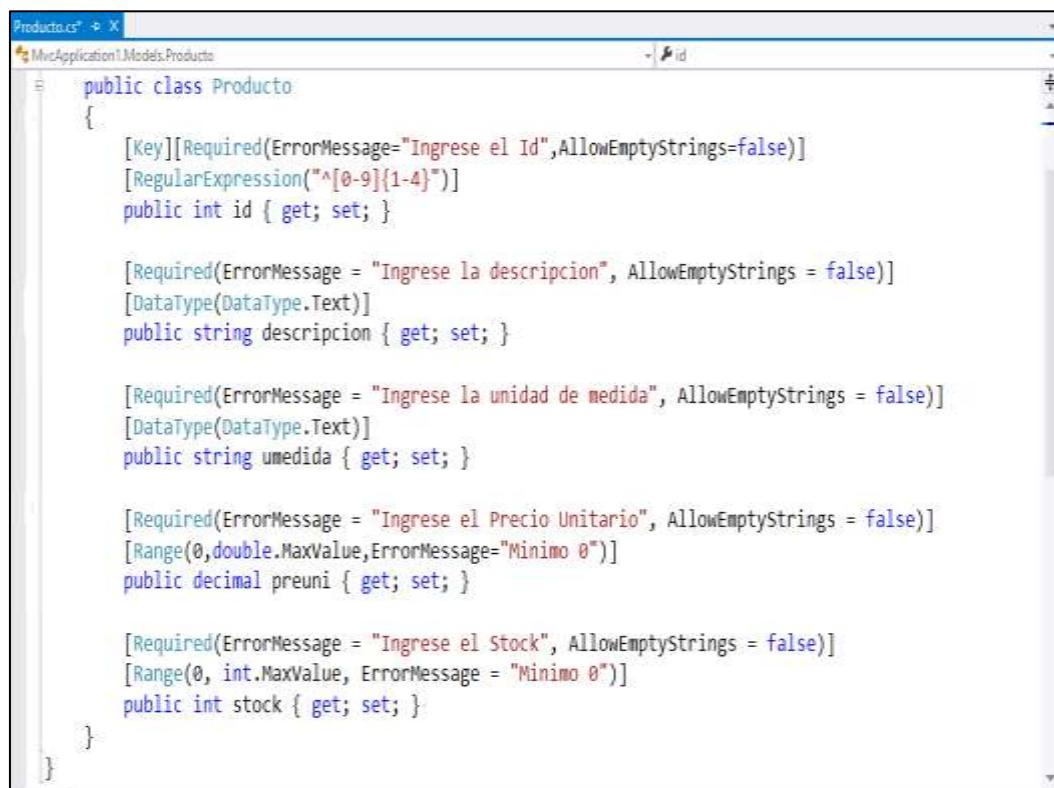
En la clase defina la librería de Anotaciones y Validaciones



```
Producto.cs  X
MvcApplication1.Models.Producto
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace MvcApplication1.Models
{
    public class Producto
    {
    }
}
```

A continuación defina la estructura de la clase Producto, validando el ingreso de sus datos.



```
Producto.cs  X
MvcApplication1.Models.Producto
public class Producto
{
    [Key][Required(ErrorMessage="Ingrese el Id",AllowEmptyStrings=false)]
    [RegularExpression("^[0-9]{1-4}")]
    public int id { get; set; }

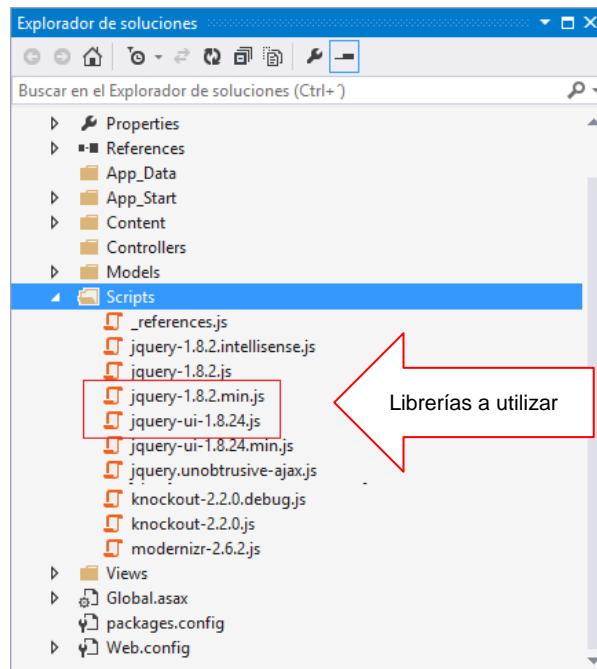
    [Required(ErrorMessage = "Ingrese la descripcion", AllowEmptyStrings = false)]
    [DataType(DataType.Text)]
    public string descripcion { get; set; }

    [Required(ErrorMessage = "Ingrese la unidad de medida", AllowEmptyStrings = false)]
    [DataType(DataType.Text)]
    public string umedida { get; set; }

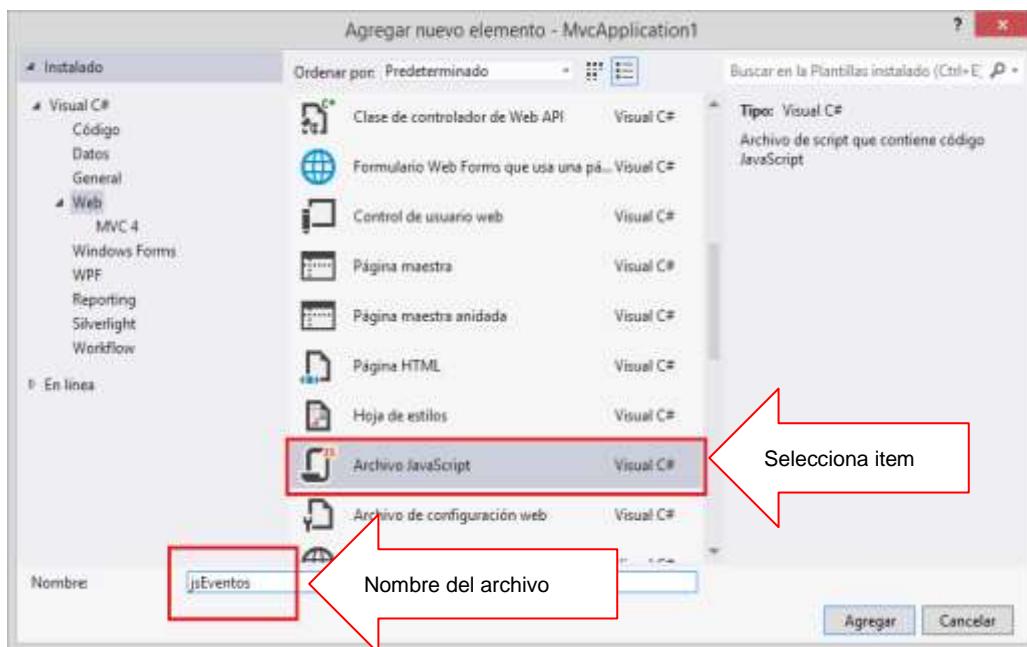
    [Required(ErrorMessage = "Ingrese el Precio Unitario", AllowEmptyStrings = false)]
    [Range(0,double.MaxValue,ErrorMessage="Minimo 0")]
    public decimal preuni { get; set; }

    [Required(ErrorMessage = "Ingrese el Stock", AllowEmptyStrings = false)]
    [Range(0, int.MaxValue, ErrorMessage = "Minimo 0")]
    public int stock { get; set; }
}
```

Para trabajar con las ventanas modales con Jquery, utilizaremos dos librerías: jquery-1.8.2 y jquery-ui-1.8.2.min, tal como se muestra.



En la carpeta Scripts, agregue un archivo JavaScript llamado jsEventos, tal como se muestra



A continuación defina cada uno de las acciones a los objetos que utilizaremos en el proceso



```

$(document).ready(function () {
    $(".btn-create").click(function (e)
    {
        $("#modal").load("/Home/Create").attr("title", "Nuevo Producto").dialog();
    });

    $(".btn-details").click(function ()
    {
        var codigo = $(this).attr("data-codigo");
        $("#modal").load("/Home/details/" + codigo).attr("title", "Visualizar").dialog();
    });

    $(".btn-edit").click(function ()
    {
        var codigo = $(this).attr("data-codigo");
        $("#modal").load("/Home/Edit/" + codigo).attr("title", "Editar Producto").dialog();
    });

    $(".btn-delete").click(function ()
    {
        var codigo = $(this).attr("data-codigo");
        $("#modal").load("/Home/Delete/" + codigo).attr("title", "Eliminar Producto").dialog();
    });
});

```

En el \_Layout.cshtml, agregue las librerías de trabajo, tal como se muestra y comentar la línea @Script.Render.



Annotations in the screenshot:

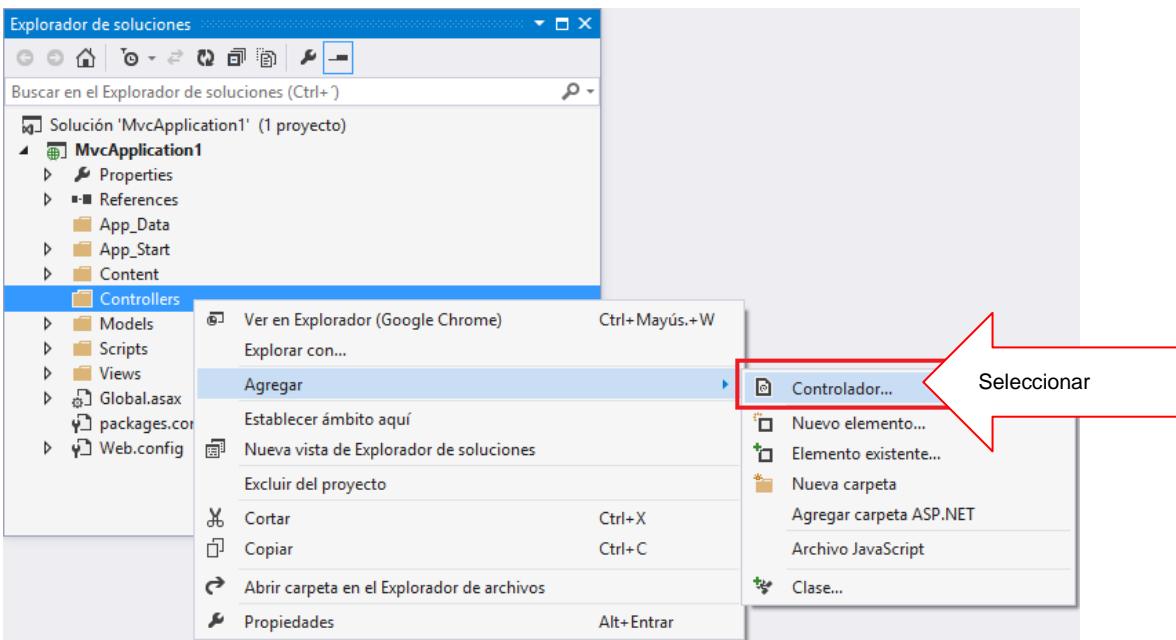
- A red box highlights the section of code under the heading "Agregar las librerías al \_Layout".
- A red box highlights the line "@\*@Scripts.Render("~/bundles/jquery")" under the heading "Comentar la línea".

```

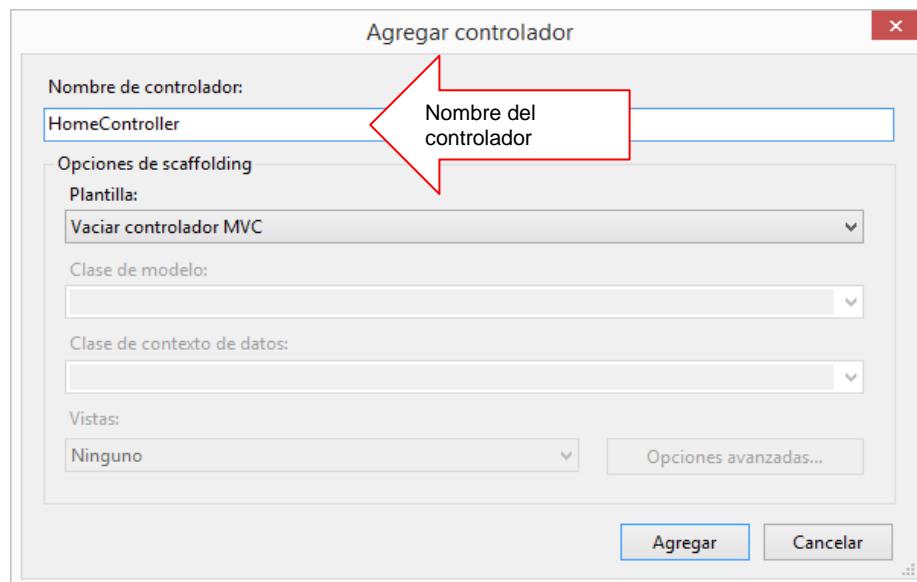
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    <link href("~/Content/themes/base/jquery-ui.css" rel="stylesheet" />
    <link href("~/Content/themes/base/jquery.ui.theme.css" rel="stylesheet" />
    <link href "~/Content/themes/base/jquery.ui.dialog.css" rel="stylesheet" />
    <script src="~/Scripts/jquery-1.8.2.min.js"></script>
    <script src="~/Scripts/jquery-ui-1.8.24.min.js"></script>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    @RenderBody()
    @*@Scripts.Render("~/bundles/jquery")*
    @RenderSection("scripts", required: false)
</body>
</html>

```

A continuación agregamos, en la carpeta Controllers, un controlador llamado Home, tal como se muestra.



Agregar el controlador llamado HomeController, tal como se muestra



En el controlador Home, primero importamos la librería Models.

A continuación definimos la lista de Producto, llamada Productos, tal como se muestra.

Definimos un método para cargar los datos a los productos llamado cargarProductos()

En el Action Index, enviamos la lista de Productos.

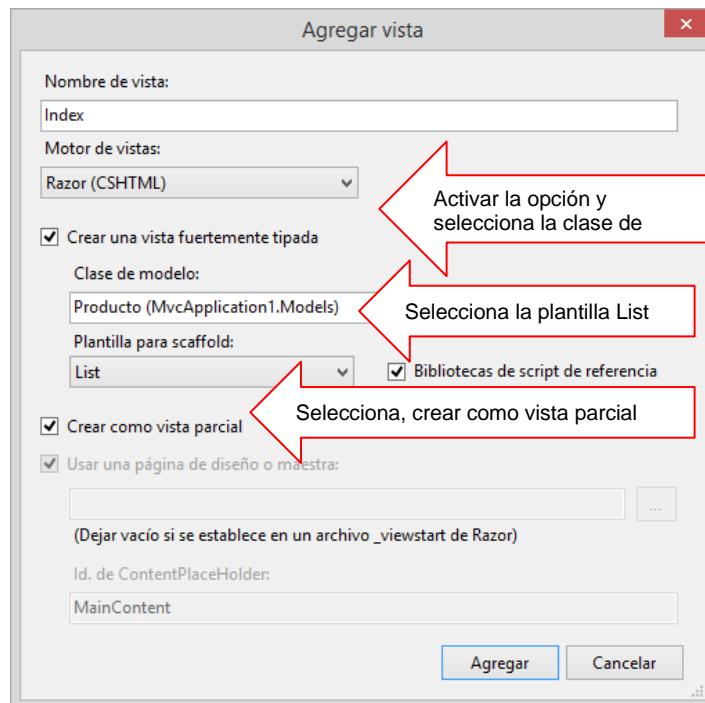
```

HomeController.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcApplication1.Models;
namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        static List<Producto> Productos = new List<Producto>();
        void cargarProductos()
        {
            Productos.Add(new Producto() { id = 1, descripcion = "silla", medida = "unidad", preuni = 100, stock = 25 });
            Productos.Add(new Producto() { id = 2, descripcion = "mesa", medida = "unidad", preuni = 120, stock = 15 });
            Productos.Add(new Producto() { id = 3, descripcion = "comoda", medida = "unidad", preuni = 130, stock = 25 });
            Productos.Add(new Producto() { id = 4, descripcion = "sillon", medida = "unidad", preuni = 220, stock = 25 });
            Productos.Add(new Producto() { id = 5, descripcion = "camarote", medida = "unidad", preuni = 450, stock = 5 });
        }
        public ActionResult Index()
        {
            cargarProductos();
            return View(Productos.ToList());
        }
    }
}

```

A continuación agregamos la Vista a la acción Index. En la ventana

- Activar la opción **crear una vista fuertemente tipada** y selecciona la clase Producto.
- En la **plantilla Scaffold**, selecciona la opción List, tal como se muestra
- Activar la opción Crear como vista parcial, presiona el botón Agregar



A continuación se diseña la Vista Index, donde aparece en el encabezado el modelo de la clase, la lista de productos y las opciones para Agregar, Editar, Actualizar y Eliminar

```

Index.cshtml*  X
@model IEnumerable<MvcApplication1.Models.Producto>
@{ViewBag.Title = "Productos";}

<script src("~/Scripts/jseventos.js")></script> // Script para ejecutar los botones

<h2>Index</h2>
<p><button class="btn-create">Nuevo Producto</button></p>

<table>
    <tr>
        <th>@Html.DisplayNameFor(model => model.id)</th>
        <th>@Html.DisplayNameFor(model => model.descripcion)</th>
        <th>@Html.DisplayNameFor(model => model.umedida)</th>
        <th>@Html.DisplayNameFor(model => model.preuni)</th>
        <th>@Html.DisplayNameFor(model => model.stock)</th>
        <th></th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.id)</td>
            <td>@Html.DisplayFor(modelItem => item.descripcion)</td>
            <td>@Html.DisplayFor(modelItem => item.umedida)</td>
            <td>@Html.DisplayFor(modelItem => item.preuni)</td>
            <td>@Html.DisplayFor(modelItem => item.stock)</td>
            <td>
                <button class="btn-details" data-codigo="@item.id">Visualizar</button> | 
                <button class="btn-edit" data-codigo="@item.id">Editar</button> | 
                <button class="btn-delete" data-codigo="@item.id">Eliminar</button>
            </td>
        </tr>
    }
</table>
<div id="modal"></div> // Bloque donde se muestran las ventanas modales

```

Script para ejecutar los botones

Botones que ejecutan los procesos

Bloque donde se muestran las ventanas modales

### Action Create

En el controlador defina la opción Create, el cual agrega un nuevo producto a la Lista

```

HomeController.cs  X
MyApplication1.Controllers.HomeController
+ Products

using ...

namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        static List<Producto> Productos = new List<Producto>();
        public ActionResult Index()
        {
            return View();
        }

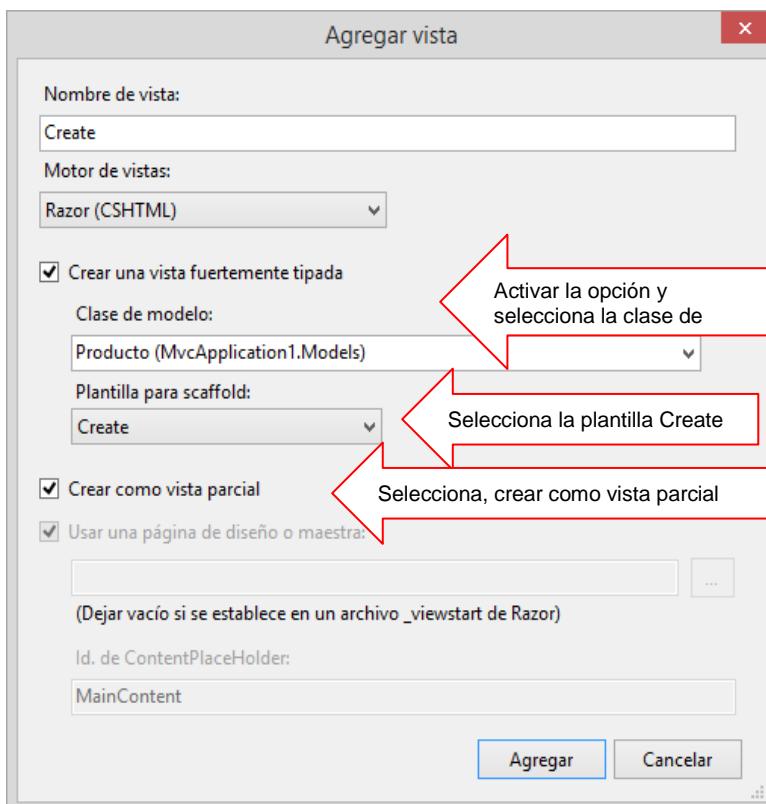
        [HttpPost]
        public ActionResult Create(Producto reg)
        {
            if (!ModelState.IsValid)
            {
                return View(reg);
            }
            Productos.Add(reg);
            return RedirectToAction("Index");
        }
    }
}

```

Acción que envía la Vista

Acción que recibe los datos de la Vista y agrega el registro

A continuación agrega la Vista a la acción Create, tal como se muestra, donde seleccionamos la clase de modelo: producto; selecciona la plantilla para scaffold: Create



En la vista Create, diseñarla tal como se muestra en la figura.

```
Create.cshtml  X HomeController.cs
@model MvcApplication1.Models.Producto

@using (Html.BeginForm())
{
    <div class="editor-label">@Html.LabelFor(model => model.id)</div>
    <div class="editor-field">@Html.EditorFor(model => model.id)</div>

    <div class="editor-label">@Html.LabelFor(model => model.descripcion)</div>
    <div class="editor-field">@Html.EditorFor(model => model.descripcion)</div>

    <div class="editor-label">@Html.LabelFor(model => model.umedida)</div>
    <div class="editor-field">@Html.EditorFor(model => model.umedida)</div>

    <div class="editor-label">@Html.LabelFor(model => model.preuni)</div>
    <div class="editor-field">@Html.EditorFor(model => model.preuni)</div>

    <div class="editor-label">@Html.LabelFor(model => model.stock)</div>
    <div class="editor-field">@Html.EditorFor(model => model.stock)</div>

    <p><input type="submit" value="Create" /></p>
    <p>@Html.ValidationSummary()</p>
}

<div>@Html.ActionLink("Back to List", "Index")</div>
```

## Action Details

En el controlador defina la acción Details, el cual retorna el registro de Productos por su campo id. Defina el proceso tal como se muestra.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcApplication1.Models;

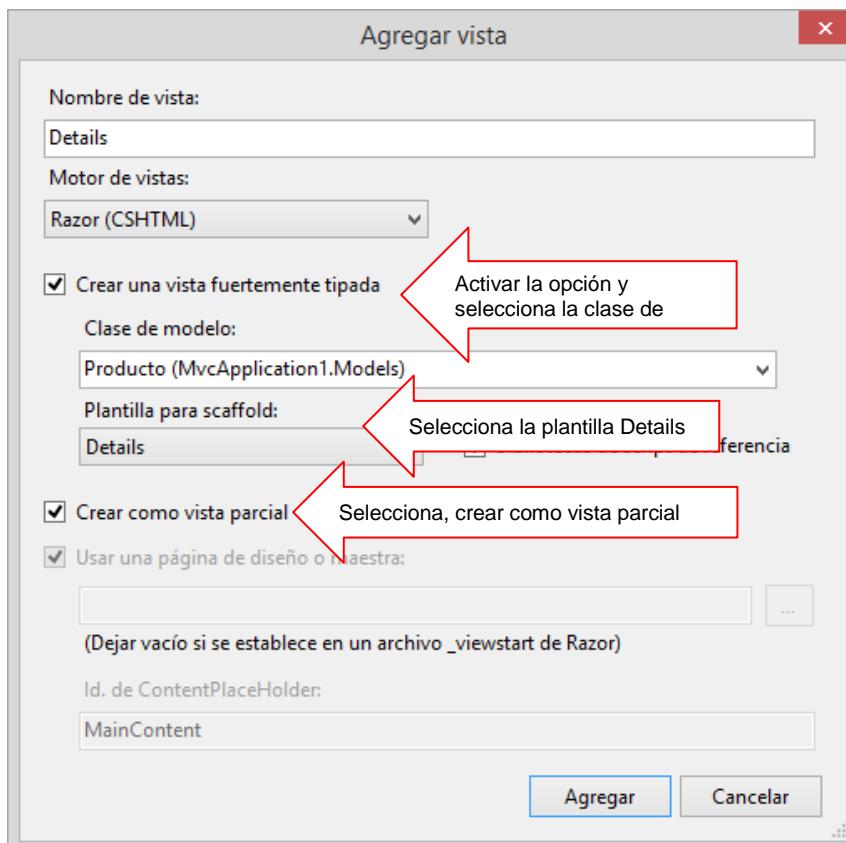
namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        static List<Producto> Productos = new List<Producto>();
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult Create(Producto reg)
        {
            Productos.Add(reg);
            return RedirectToAction("Index");
        }

        public ActionResult Details(int? id=null)
        {
            Producto reg = Productos.Where(p => p.id == id).FirstOrDefault();
            return View(reg);
        }
    }
}

```

A continuación agrega la Vista a la acción Details, tal como se muestra, donde seleccionamos la clase de modelo: producto; selecciona la plantilla para scaffold: Details



A continuación diseña la vista Details, tal como se muestra



```

@model MvcApplication1.Models.Producto

<div class="display-label">@Html.DisplayNameFor(model => model.descripcion)</div>
<div class="display-field">@Html.DisplayFor(model => model.descripcion)</div>

<div class="display-label">@Html.DisplayNameFor(model => model.umedida)</div>
<div class="display-field">@Html.DisplayFor(model => model.umedida)</div>

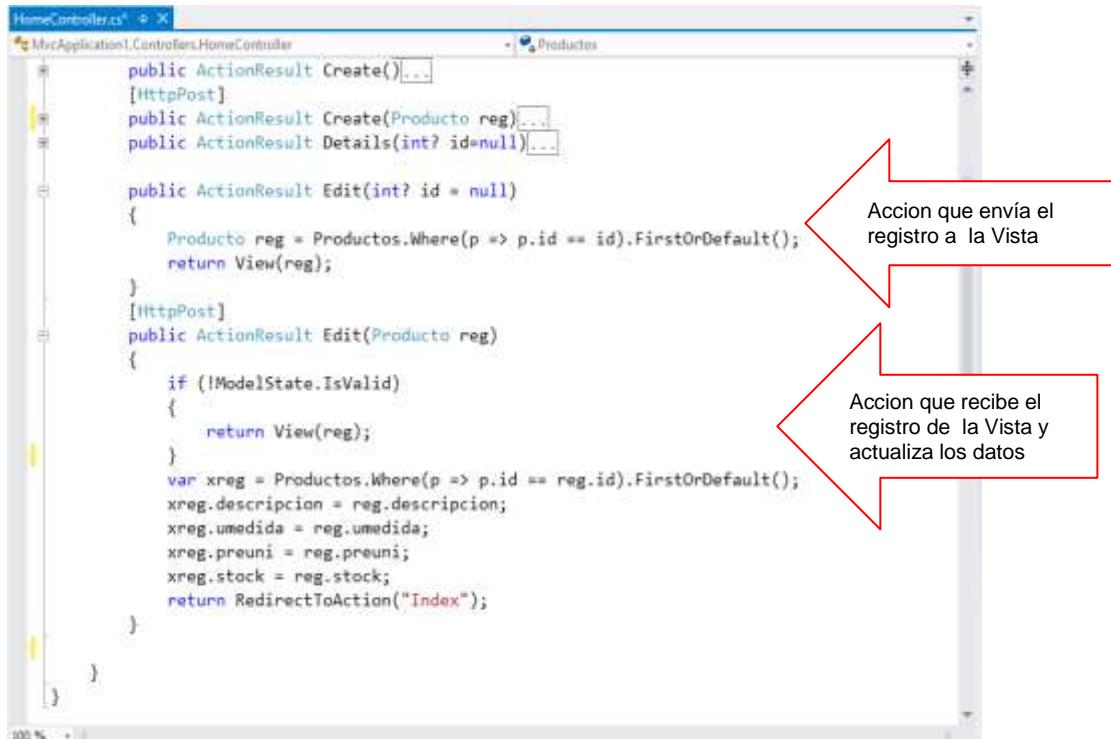
<div class="display-label">@Html.DisplayNameFor(model => model.preuni)</div>
<div class="display-field">@Html.DisplayFor(model => model.preuni)</div>

<div class="display-label">@Html.DisplayNameFor(model => model.stock)</div>
<div class="display-field">@Html.DisplayFor(model => model.stock)</div>
<p>
    @Html.ActionLink("Retornar", "Index")
</p>

```

## Action Edit

En el controlador defina la acción Edit, el cual retorna el registro de Productos por su campo id, y recibe el registro para actualizar sus datos. Defina el proceso tal como se muestra



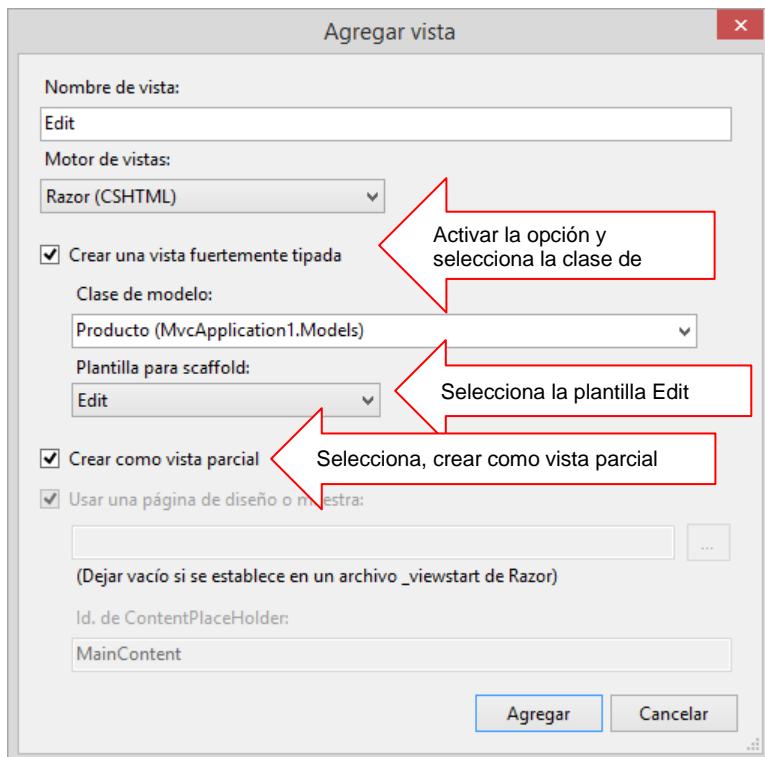
```

public ActionResult Create()...
[HttpPost]
public ActionResult Create(Producto reg)...
public ActionResult Details(int? id=null)...

public ActionResult Edit(int? id = null)
{
    Producto reg = Productos.Where(p => p.id == id).FirstOrDefault();
    return View(reg);
}
[HttpPost]
public ActionResult Edit(Producto reg)
{
    if (!ModelState.IsValid)
    {
        return View(reg);
    }
    var xreg = Productos.Where(p => p.id == reg.id).FirstOrDefault();
    xreg.descripcion = reg.descripcion;
    xreg.umedida = reg.umedida;
    xreg.preuni = reg.preuni;
    xreg.stock = reg.stock;
    return RedirectToAction("Index");
}
}

```

A continuación agrega la Vista a la acción Edit, tal como se muestra.



A continuación diseña la Vista, tal como se muestra.

```
Editor.cshtml > X HomeController.cs
@model MvcApplication1.Models.Producto



@Html.ActionLink("Back to List", "Index")



@using (Html.BeginForm()) {
    @Html.HiddenFor(model => model.id)

    <div class="editor-label">@Html.LabelFor(model => model.descripcion)</div>
    <div class="editor-field">@Html.EditorFor(model => model.descripcion)</div>

    <div class="editor-label">@Html.LabelFor(model => model.umedida)</div>
    <div class="editor-field">@Html.EditorFor(model => model.umedida)</div>

    <div class="editor-label">@Html.LabelFor(model => model.preuni)</div>
    <div class="editor-field">@Html.EditorFor(model => model.preuni)</div>

    <div class="editor-label">@Html.LabelFor(model => model.stock)</div>
    <div class="editor-field">@Html.EditorFor(model => model.stock)</div>

    <p><input type="submit" value="Actualizar" /></p>
    <p>@Html.ValidationSummary(true)</p>
}
```

## Action Delete

En el controlador defina la acción Delete, el cual retorna el registro de Productos por su campo id, y confirma la eliminación del registro por su id. Defina el proceso tal como se muestra

```

HomeController.cs  X
MvcApplication1.Controllers.HomeController
  ↳ Products

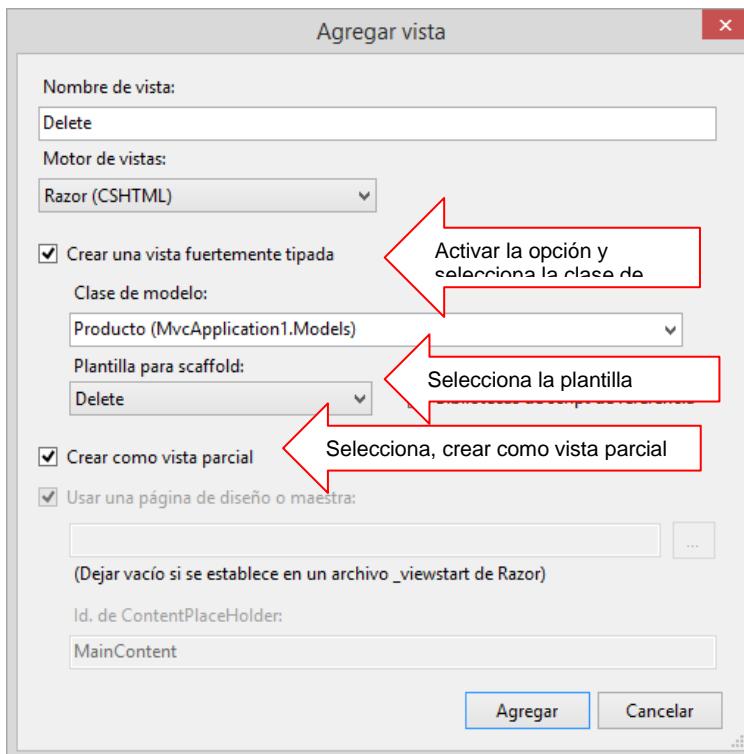
using ...
namespace MvcApplication1.Controllers
{
    public class HomeController : Controller
    {
        static List<Producto> Productos = new List<Producto>();
        public ActionResult Index()
        {
            return View();
        }
        public ActionResult Create()
        {
            return View();
        }
        [HttpPost]
        public ActionResult Create(Producto reg)
        {
            Productos.Add(reg);
            return RedirectToAction("Index");
        }
        public ActionResult Details(int? id = null)
        {
            if (id == null)
                return HttpNotFound();
            var reg = Productos.Find(id);
            if (reg == null)
                return HttpNotFound();
            return View(reg);
        }
        public ActionResult Edit(int? id = null)
        {
            if (id == null)
                return HttpNotFound();
            var reg = Productos.Find(id);
            if (reg == null)
                return HttpNotFound();
            return View(reg);
        }
        [HttpPost]
        public ActionResult Edit(Producto reg)
        {
            var prod = Productos.Find(reg.id);
            if (prod != null)
                prod.nombre = reg.nombre;
            return RedirectToAction("Index");
        }
        public ActionResult Delete(int? id = null)
        {
            if (id == null)
                return HttpNotFound();
            var reg = Productos.Find(id);
            if (reg == null)
                return HttpNotFound();
            Productos.Remove(reg);
            return RedirectToAction("Index");
        }
    }
}

```

Acción que envía el registro a la Vista

Acción que recibe el registro de la Vista y elimina el registro

A continuación agrega la Vista Delete, tal como se muestra



Defina la vista tal como se muestra.

```
Deleted.cshtml # X HomeController.cs
@model MvcApplication1.Models.Producto

@Html.ActionLink("Back to List", "Index")

<h3>Deseas Eliminar?</h3>

@using (Html.BeginForm()) {
    @Html.HiddenFor(model => model.id)

    <div class="display-label">@Html.DisplayNameFor(model => model.descripcion)</div>
    <div class="display-field">@Html.DisplayFor(model => model.descripcion)</div>

    <p><input type="submit" value="Delete" /> </p>
}
```

Para ejecutar el proceso presiona F5, donde se procede a: Agregar, Visualizar, Editar y Eliminar productos utilizando pantallas modales



## BOOTSTRAP

### Introducción

Bootstrap es un framework CSS lanzado por un grupo de diseñadores de Twitter, para maquetar y diseñar proyectos web, cuya particularidad es la de adaptar la interfaz del sitio web al tamaño del dispositivo en que se visualice. Es decir, el sitio web se adapta automáticamente al tamaño de una PC, una Tablet u otro dispositivo. Esta técnica de diseño y desarrollo se conoce como “responsive design” o diseño adaptativo.

Bootstrap brinda una base pre-codificada de HTML y CSS para armar el diseño de una página web o una aplicación web, y al ofrecerse como un recurso de código abierto es fácil de personalizar y adaptar a múltiples propósitos.

Al incorporar estilos para una enorme cantidad de elementos utilizados en websites y aplicaciones modernas, reduce enormemente el tiempo necesario para implementar un site al mismo tiempo que mantiene la capacidad para ser flexible y adaptable.

Una ventaja es que, por lo mismo que en sí mismo no requiere conexión con una base de datos, un sitio web implementado en Bootstrap corre perfectamente desde una versión local, sin acceso a un servidor.

### Estructura y función

Bootstrap es modular y consiste esencialmente en una serie de hojas de estilo LESS que implementan la variedad de componentes de la herramienta. Una hoja de estilo llamada bootstrap.less incluye los componentes de las hojas de estilo. Los desarrolladores pueden adaptar el mismo archivo de Bootstrap, seleccionando los componentes que deseen usar en su proyecto.

Los ajustes son posibles en una medida limitada a través de una hoja de estilo de configuración central. Los cambios más profundos son posibles mediante las declaraciones LESS. El uso del lenguaje de hojas de estilo LESS permite el uso de variables, funciones y operadores, selectores anidados, así como clases mixin.

Desde la versión 2.0, la configuración de Bootstrap también tiene una opción especial de "Personalizar" en la documentación. Por otra parte, los desarrolladores eligen en un formulario los componentes y ajustes deseados, y de ser necesario, los valores de varias opciones a sus necesidades. El paquete consecuentemente generado ya incluye la hoja de estilo CSS pre-compilada.

### Sistema de cuadrilla y diseño sensible

Bootstrap viene con una disposición de cuadrilla estándar de 940 píxeles de ancho. Alternativamente, el desarrollador puede usar un diseño de ancho-variable. Para ambos casos, la herramienta tiene cuatro variaciones para hacer uso de distintas resoluciones y tipos de dispositivos: teléfonos móviles, formato de retrato y paisaje, tabletas y computadoras con baja y alta resolución (pantalla ancha). Esto ajusta el ancho de las columnas automáticamente.

### Entendiendo la hoja de estilo CSS

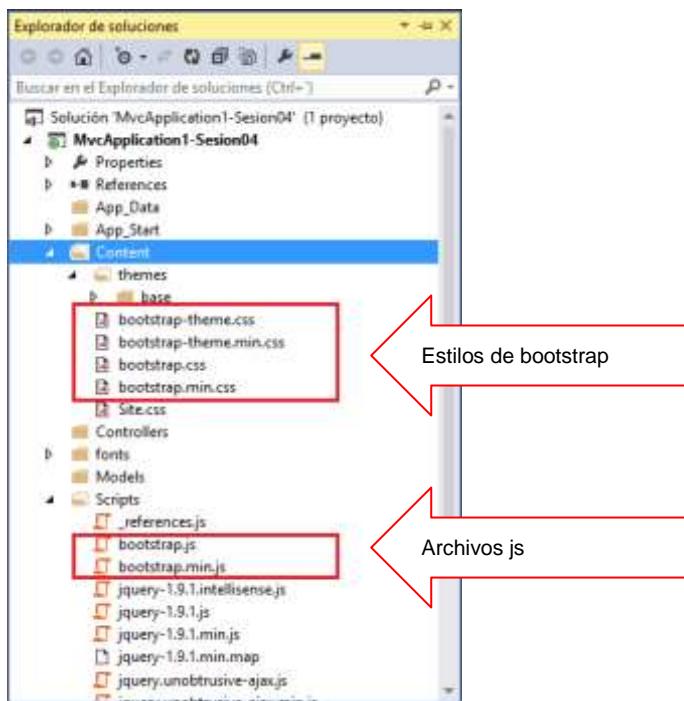
Bootstrap proporciona un conjunto de hojas de estilo que proveen definiciones básicas de estilo para todos los componentes de HTML. Esto otorga una uniformidad al navegador y al sistema de anchura, da una apariencia moderna para el formateo de los elementos de texto, tablas y formularios.

### Componentes re-usables

En adición a los elementos regulares de HTML, Bootstrap contiene otra interfaz de elementos comúnmente usados. Ésta incluye botones con características avanzadas (e.g grupo de botones o botones con opción de menú desplegable, listas de navegación, etiquetas horizontales y verticales, ruta de navegación, paginación, etc.), etiquetas, capacidades avanzadas de miniaturas tipográficas, formatos para mensajes de alerta y barras de progreso.

### Plug-ins de JavaScript

Los componentes de JavaScript para Bootstrap están basados en la librería jQuery de JavaScript. Los plug-ins se encuentran en la herramienta de plug-in de jQuery. Proveen elementos adicionales de interfaz de usuario como diálogos, tooltips y carruseles. También extienden la funcionalidad de algunos elementos de interfaz existentes, incluyendo por ejemplo una función de auto-completar para campos de entrada (input).



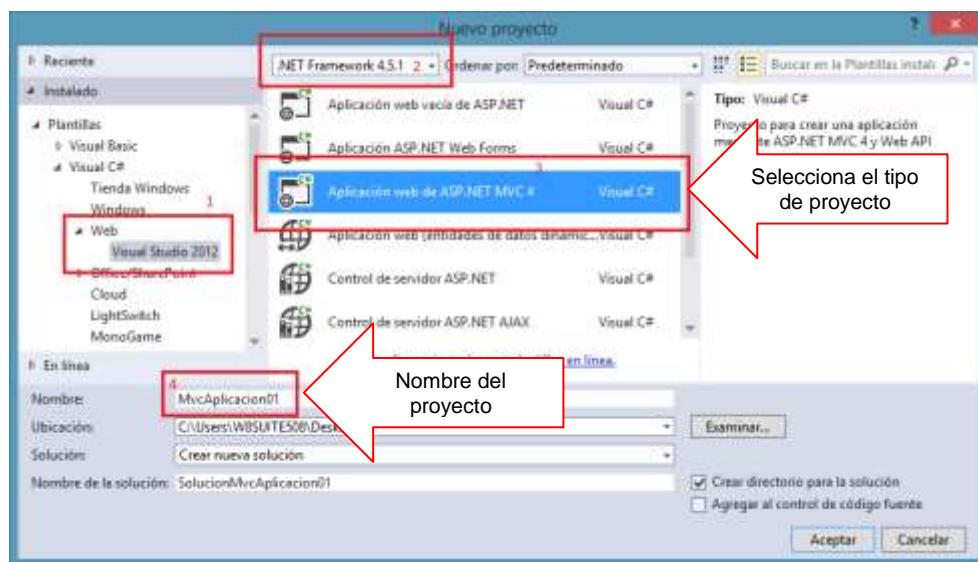
## Aplicación ASP.NET MVC con Bootstrap

Implemente un proyecto ASP.NET MVC con bootstrap que permita abrir ventanas modales.

### Creando el proyecto

Iniciamos Visual Studio 2012 y creamos un nuevo proyecto:

1. Seleccionar el proyecto Web Visual Studio 2012
2. Seleccionar el FrameWork: 4.5.1
3. Seleccionar la plantilla Aplicación web de ASP.NET MVC 4
4. Asignar el nombre del proyecto
5. Presionar el botón ACEPTAR

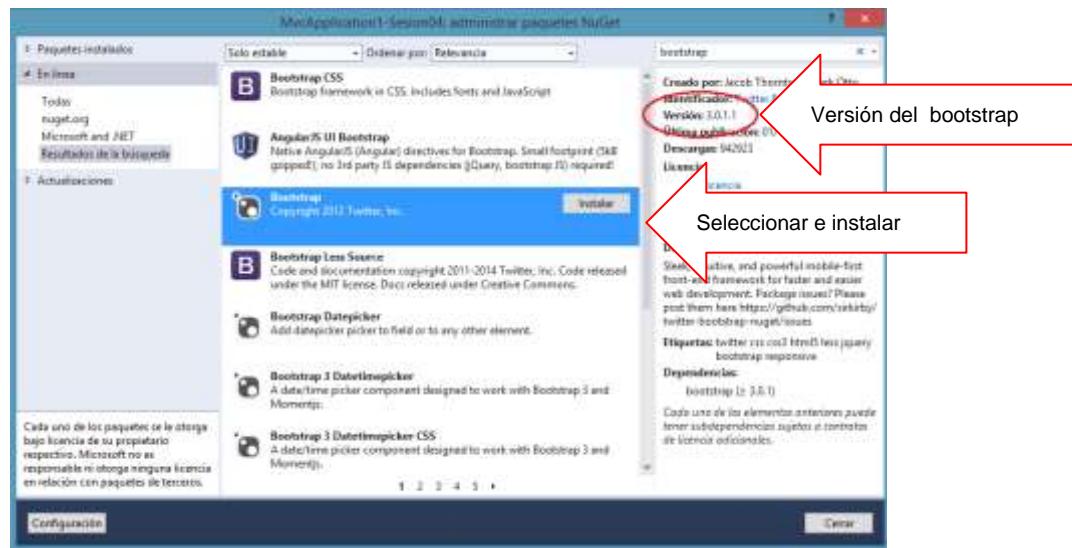


A continuación, seleccionar la plantilla del proyecto:

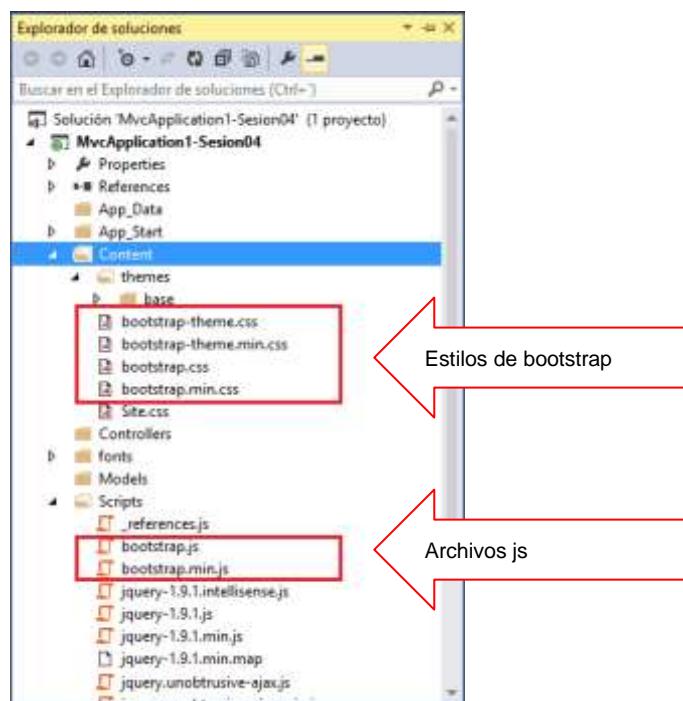
- Seleccionar la plantilla Básico
- Seleccionar el motor de vistas: Razor



En el proyecto agregar el FrameWork Bootstrap para ser implementado en la aplicación ASP.NET MVC, tal como se muestra



Instalado el framework, se podrá visualizar los archivos dentro del explorador de soluciones



En el archivo **BundleConfig** vamos a incluir varios archivos y librerías js y cs, tal como se muestra. En lugar de incluir directamente las etiquetas <link> o <script> para referenciar los archivos externos, lo que llamamos es Styles.Render Scripts.Render

```

Index.cshtml HomeController.cs BundleConfig.cs * Layout.cshtml
MvcApplication1_Sesion84\BundleConfig.cs
using System;
using System.Web;
using System.Web.Optimization;

namespace MvcApplication1_Sesion84
{
    public class BundleConfig
    {
        // Para obtener más información acerca de Bundling, consulte http://go.microsoft.com/fwlink/?LinkId=254725
        public static void RegisterBundles(BundleCollection bundles)
        {
            bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                        "~/Scripts/jquery-{version}.js"));

            bundles.Add(new ScriptBundle("~/bundles/jqueryui").Include(
                        "~/Scripts/jquery-ui-{version}.js"));

            bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
                        "~/Scripts/jquery.unobtrusive*", 
                        "~/Scripts/jquery.validate*"));

            bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
                        "~/Scripts/modernizr-*"));

            bundles.Add(new ScriptBundle("~/bundles/bootstrapjs").Include(
                        "~/Scripts/bootstrap.js", "~/Scripts/bootstrap-modal.js", "~/Scripts/bootstrap.min.js"));

            bundles.Add(new StyleBundle("~/Content/bootstrapcss").Include(
                        "~/Content/bootstrap.css", "~/Content/bootstrap.min.css"));

            bundles.Add(new StyleBundle("~/Content/css").Include(
                        "~/Content/site.css"));

            bundles.Add(new StyleBundle("~/Content/themes/base/css").Include(
                        "~/Content/themes/base/jquery.ui.core.css",
                        "~/Content/themes/base/jquery.ui.resizable.css",
                        "~/Content/themes/base/jquery.ui.selectable.css",
                        "~/Content/themes/base/jquery.ui.accordion.css",
                        "~/Content/themes/base/jquery.ui.autocomplete.css",
                        "~/Content/themes/base/jquery.ui.button.css",
                        "~/Content/themes/base/jquery.ui.slider.css",
                        "~/Content/themes/base/jquery.ui.tabs.css",
                        "~/Content/themes/base/jquery.ui.datepicker.css",
                        "~/Content/themes/base/jquery.ui.progressbar.css",
                        "~/Content/themes/base/jquery.ui.tooltip.css"));
        }
    }
}

```

En la pagina \_Layout.cshtml, referenciamos las librerías utilizando @Scripts.Render y @Styles.Render, tal como se muestra

```

Index.cshtml HomeController.cs Layout.cshtml
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>

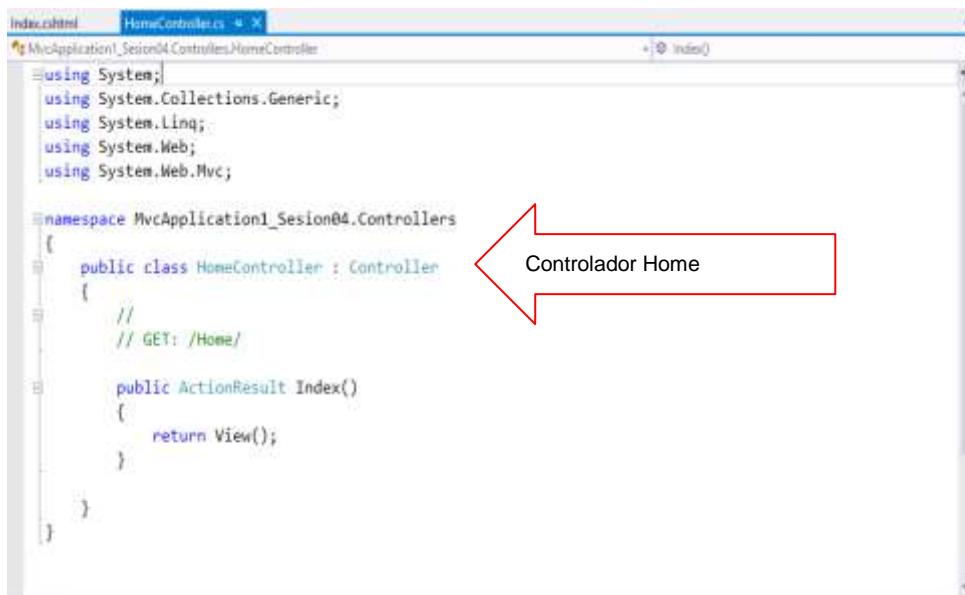
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrapjs")

    @* @Styles.Render("~/Content/css")*@
    @Scripts.Render("~/bundles/modernizr")
    @Styles.Render("~/Content/bootstrapcss")
</head>
<body>
    @RenderBody()

    @RenderSection("scripts", required: false)
</body>
</html>

```

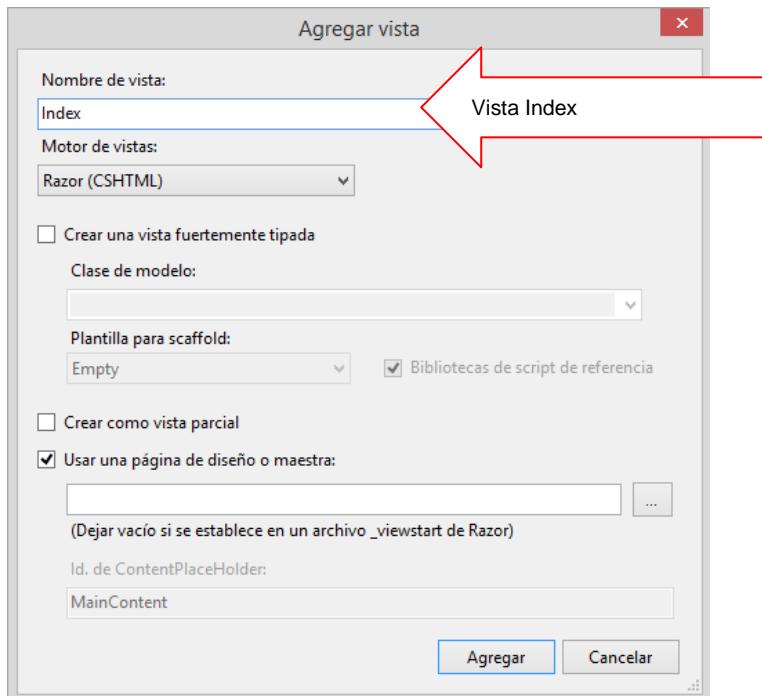
En la carpeta Controllers, agrega un controlador llamado Home.



```
Index.cshtml HomeController.cs * 
MvcApplication1_Session04.Controllers.HomeController
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication1_Session04.Controllers
{
    public class HomeController : Controller
    {
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

En la acción Index, agrega una vista, tal como se muestra.



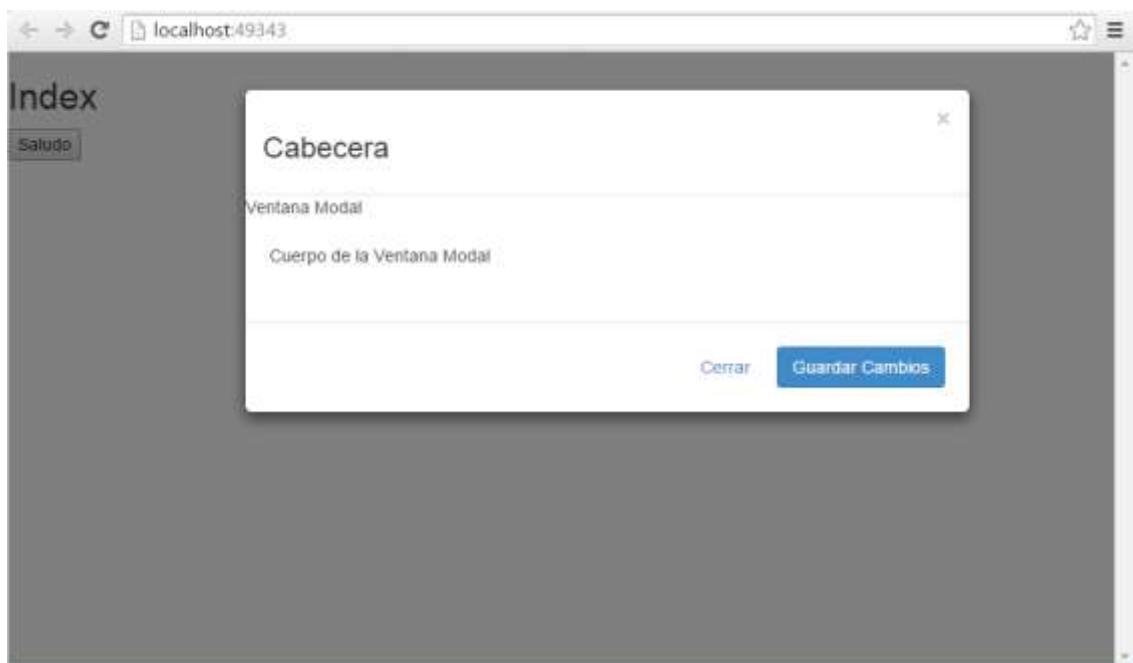
En la ventana Index, agrega el código script, tal como se muestra

```
Index.cshtml*  ↳ X
    @ViewBag.Title = "Index";
    <script type="text/javascript">
        $(document).ready(function () {
            $('#btnSaludo').click(function () {
                $('#test_modal').modal('show');
            });
            $('.btn-primary').click(function () {
                alert("Proceso");
            });
        });
    </script>

    <h2>Index</h2>
    <input type="button" id="btnSaludo" value="Saludo" />

    <div class="modal fade" id="test_modal">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <a class="close" data-dismiss="modal">&times;</a>
                    <h3>Cabecera</h3>
                </div>
                <div class="modal-title">Ventana Modal</div>
                <div class="modal-body">
                    <p>Cuerpo de la Ventana Modal</p>
                </div>
                <div class="modal-footer">
                    <a href="#" class="btn" data-dismiss="modal">Cerrar</a>
                    <a href="#" class="btn btn-primary">Guardar Cambios</a>
                </div>
            </div>
        </div>
    </div>
```

Ejecuta el proyecto, presionando F5. Al presionar el botón Saludo, se visualiza una ventana Modal, tal como se muestra.



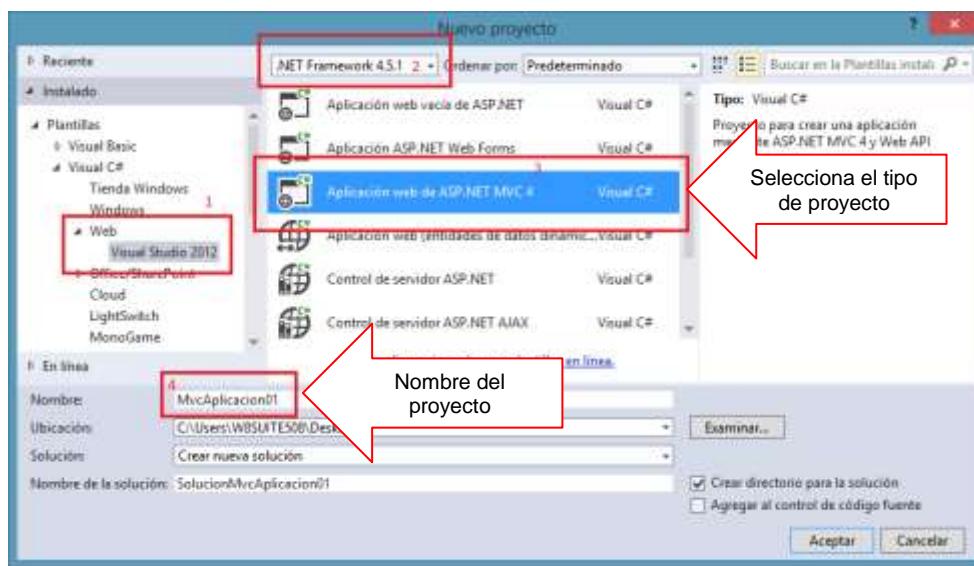
## Aplicación ASP.NET MVC con Bootstrap

Implemente un proyecto ASP.NET MVC con bootstrap que permita realizar el mantenimiento de datos utilizando ventanas modales.

### Creando el proyecto

Iniciamos Visual Studio 2012 y creamos un nuevo proyecto:

1. Seleccionar el proyecto Web Visual Studio 2012
2. Seleccionar el FrameWork: 4.5.1
3. Seleccionar la plantilla Aplicación web de ASP.NET MVC 4
4. Asignar el nombre del proyecto
5. Presionar el botón ACEPTAR

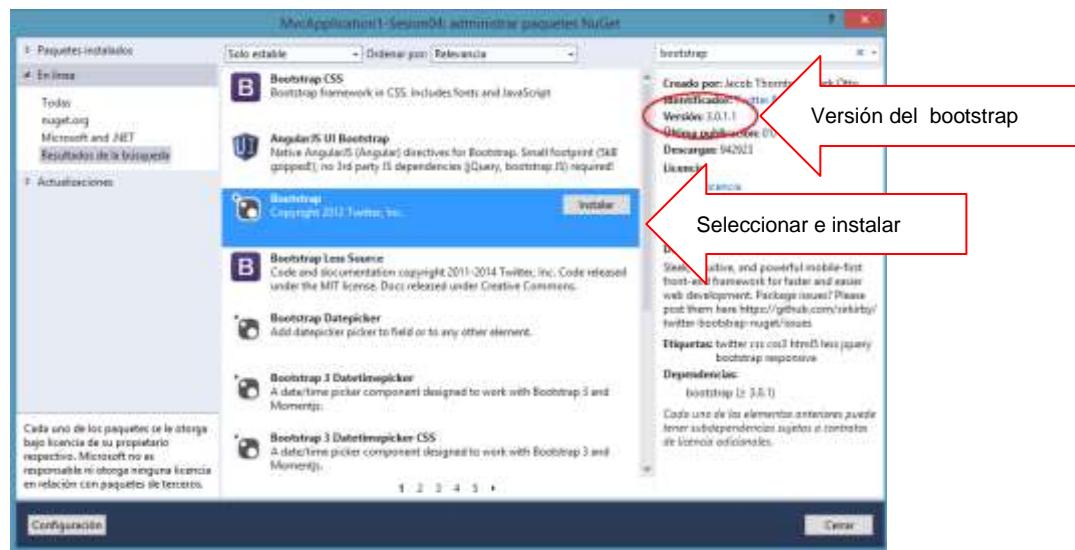


A continuación, seleccionar la plantilla del proyecto:

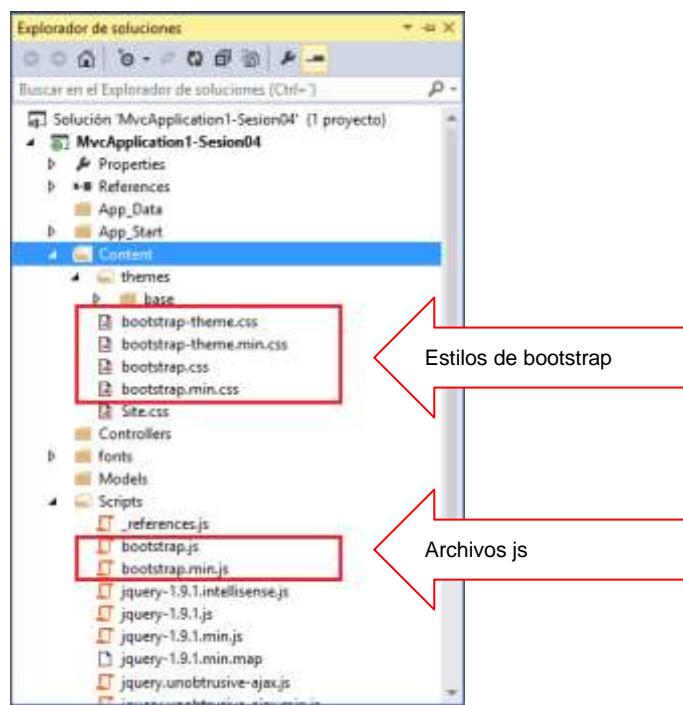
- Seleccionar la plantilla Básico
- Seleccionar el motor de vistas: Razor



En el proyecto agregar el FrameWork Bootstrap para ser implementado en la aplicación ASP.NET MVC, tal como se muestra

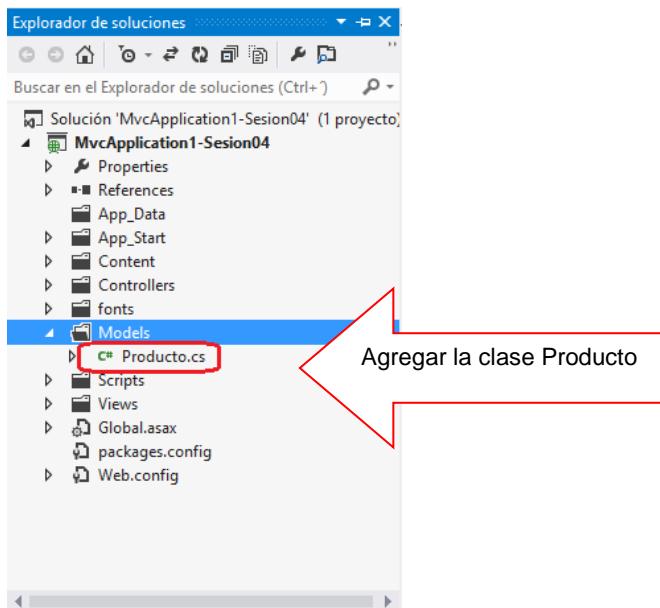


Instalado el framework, se podrá visualizar los archivos dentro del explorador de soluciones



## Trabajando con el modelo de datos

En la carpeta Models agregar una clase llamada Producto, tal como se muestra en la figura



En la clase Producto, defina su estructura y sus validaciones, tal como se muestra

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace MvcApplication1_Sesion04.Models
{
    public class Producto
    {
        [Required(ErrorMessage = "Ingrese el id", AllowEmptyStrings = false)]
        public string id { get; set; }

        [Required(ErrorMessage = "Ingrese la descripcion", AllowEmptyStrings = false)]
        public string descripcion { get; set; }

        [Required(ErrorMessage = "Ingrese la unidad de medida", AllowEmptyStrings = false)]
        public string umedida { get; set; }

        [Required(ErrorMessage = "Ingrese el precio unitario", AllowEmptyStrings = false)]
        [Range(0, double.MaxValue, ErrorMessage = "Minimo 0")]
        public double preuni { get; set; }

        [Required(ErrorMessage = "Ingrese el stock", AllowEmptyStrings = false)]
        [Range(0, int.MaxValue, ErrorMessage = "Minimo 0")]
        public int stock { get; set; }
    }
}

```

The screenshot shows the 'Producto.cs' file in the code editor. The 'using System.ComponentModel.DataAnnotations;' line is highlighted with a red box. A red arrow points from the text 'Validaciones y notaciones' to this line. Another red arrow points from the text 'Estructura de la clase producto' to the start of the 'public class Producto' definition.

En el archivo compartido \_Layout.cshtml, agregar los Scripts y Styles del bootstrap, tal como se muestra

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>

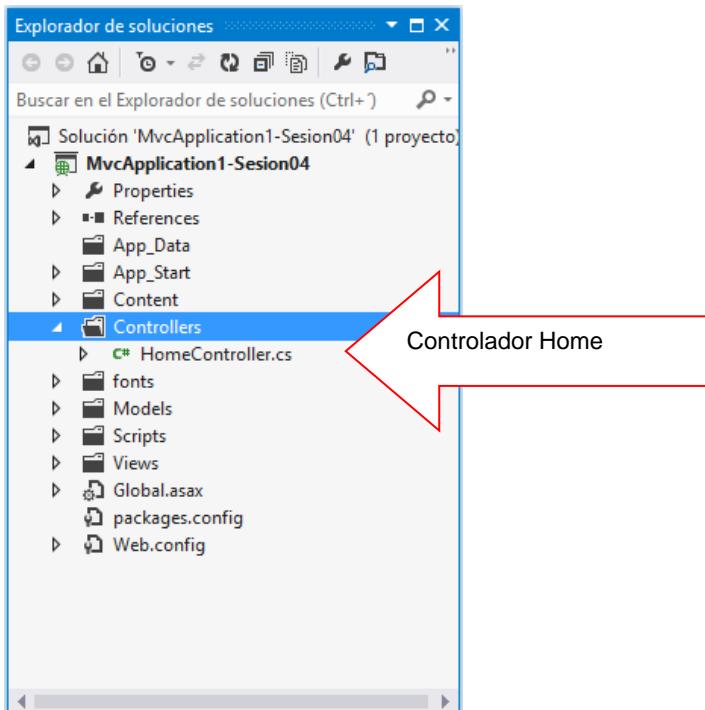
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrapjs")

    @Scripts.Render("~/bundles/modernizr")
    @Styles.Render("~/Content/bootstrapcss")
</head>
<body>
    @RenderBody()

    @RenderSection("scripts", required: false)
</body>
</html>
```

## Trabajando con el Controlador y las Vistas

En la carpeta Controllers, agregar el controlador llamado HomeController, tal como se muestra en la figura.



## ActionResult Index

En la ventana del controlador, defina el ActionResult Index, el cual retorna la lista de los registros de Producto.

```

HomeController.cs  X
MvcApplication1_Sesion04.Controllers.HomeController  Productos

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using MvcApplication1_Sesion04.Models;

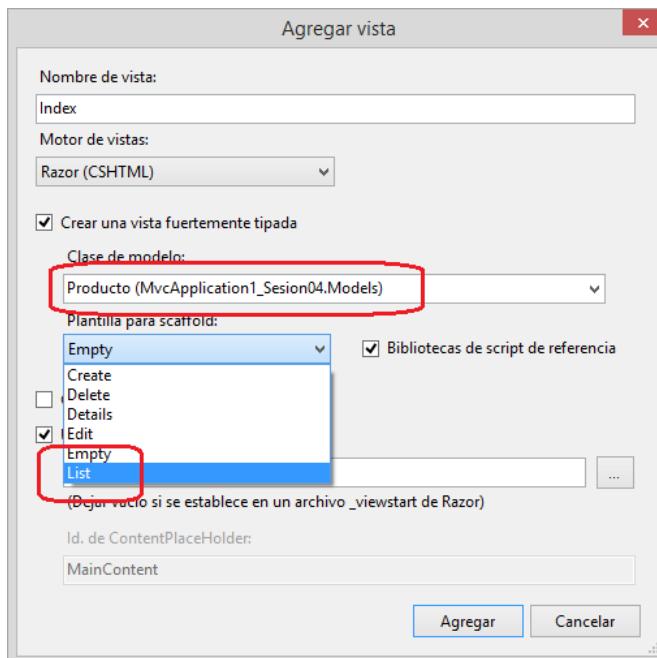
namespace MvcApplication1_Sesion04.Controllers
{
    public class HomeController : Controller
    {
        static List<Producto> Productos = new List<Producto>();

        public ActionResult Index()
        {
            return View(Productos.ToList());
        }
    }
}

```

The screenshot shows the `HomeController.cs` file in a code editor. A red box highlights the `using MvcApplication1_Sesion04.Models;` statement, with a callout pointing to it labeled "Referencia a la carpeta Models". Another red box highlights the `static List<Producto> Productos = new List<Producto>();` declaration, with a callout pointing to it labeled "Lista de Productos".

A continuación agrega una Vista al Action Index, donde seleccionamos la clase Producto y la plantilla List, tal como se muestra.



Vista Index.cshtml, tal como se muestra.

```
Index.cshtml * X
@model IEnumerable<MvcApplication1_Sesion04.Models.Producto>
@{ ViewBag.Title = "Index"; }

<h2>Index</h2>

<p>@Html.ActionLink("Create New", "Create")</p>


| Código                                 | Descripción                                     | Unidad de Medida                             | Precio Unitario                            | Stock                                     |                                                |
|----------------------------------------|-------------------------------------------------|----------------------------------------------|--------------------------------------------|-------------------------------------------|------------------------------------------------|
| @Html.DisplayFor(modelItem => item.id) | @Html.DisplayFor(modelItem => item.descripcion) | @Html.DisplayFor(modelItem => item.unmedida) | @Html.DisplayFor(modelItem => item.preuni) | @Html.DisplayFor(modelItem => item.stock) | <a href="#">Edit</a>   <a href="#">Details</a> |


```

Codifica en la vista Index, para trabajar con ventanas modales en los procesos

```
Nuevo.cshtml      Index.cshtml * X
@model IEnumerable<MvcApplication1_Sesion04.Models.Producto>
@{ ViewBag.Title = "Index"; }

<script type="text/javascript">
$(document).ready(function () {
    $('#btnNuevo').click(function (ev) {
        $('#modal-content').load('/Home/Nuevo');
    });
})
</script>

<div id="ventana" class="modal fade" role="dialog" style="display:none">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal">x</button>
                <h4 class="modal-title">Mantenimiento de Productos</h4>
            </div>
            <div class="modal-body">
                <div id="modal-content">
                    Cargando datos...
                </div>
            </div>
            <div>
                <button type="button" class="btn btn-primary" data-dismiss="modal">Cancelar</button>
                <button type="button" class="btn btn-primary" data-dismiss="modal">Aceptar</button>
            </div>
        </div>
    </div>
</div>
<h2>Index</h2>
<div class="row">
    <a id="btnNuevo" data-toggle="modal" href="#ventana" class="btn btn-primary btn-lg">Nuevo</a>
</div>
```

Script para programar el evento Click de btnNuevo

Bloque donde se muestra la ventana modal al ejecutar un proceso

Etiqueta btnNuevo

## Trabajando con el ActionResult Nuevo

A continuación defina el ActionResult Nuevo, tal como se muestra

```

HomeController.cs
using ...;

namespace MvcApplication1_Sesion04.Controllers
{
    public class HomeController : Controller
    {
        static List<Producto> Productos = new List<Producto>();

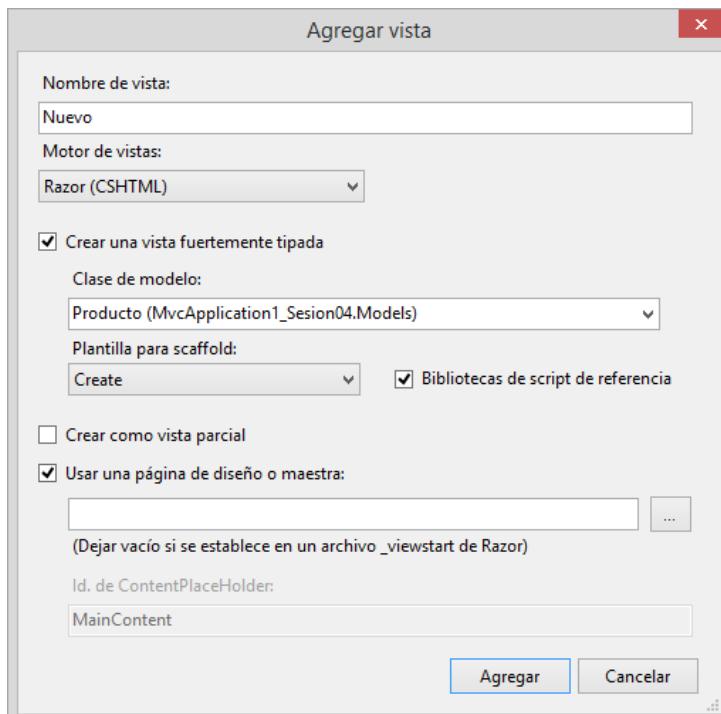
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult Nuevo(Producto reg)
        {
            if (!ModelState.IsValid)
                return View(reg);

            Productos.Add(reg);
            return RedirectToAction("Index");
        }
    }
}

```

A continuación agregamos la Vista a la acción Nuevo, donde seleccionamos la clase Producto y la plantilla será Create, tal como se muestra



En la vista Nuevo, realizar los cambios a la vista, tal como se muestra



```

@model MvcApplication1_Session04.Models.Producto
@{
    Layout = null;
}



## Nuevo


@using (Html.BeginForm()) {
    <fieldset>
        <legend>Producto</legend>
        <div class="editor-label">@Html.LabelFor(model => model.id)</div>
        <div class="editor-field">@Html.EditorFor(model => model.id) </div>

        <div class="editor-label">@Html.LabelFor(model => model.descripcion)</div>
        <div class="editor-field">@Html.EditorFor(model => model.descripcion) </div>

        <div class="editor-label">@Html.LabelFor(model => model.umedida)</div>
        <div class="editor-field">@Html.EditorFor(model => model.umedida)</div>

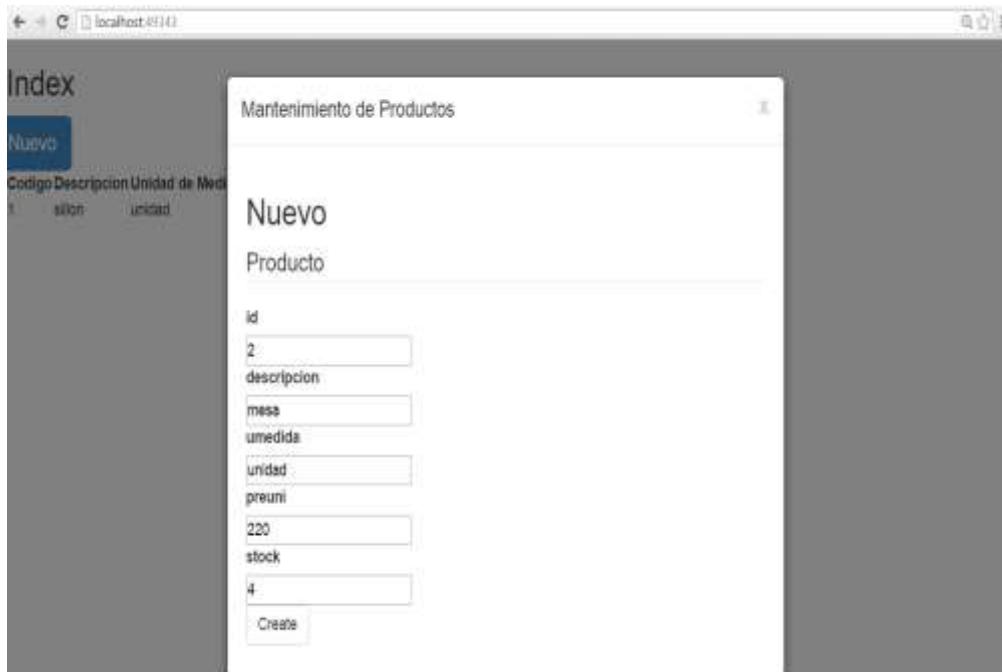
        <div class="editor-label">@Html.LabelFor(model => model.pneuni)</div>
        <div class="editor-field">@Html.EditorFor(model => model.pneuni)</div>

        <div class="editor-label">@Html.LabelFor(model => model.stock)</div>
        <div class="editor-field">@Html.EditorFor(model => model.stock)</div>

        <p><input type="submit" value="Create" class="btn btn-default" /></p>
        <div>@Html.ValidationSummary(true)</div>
    </fieldset>
}

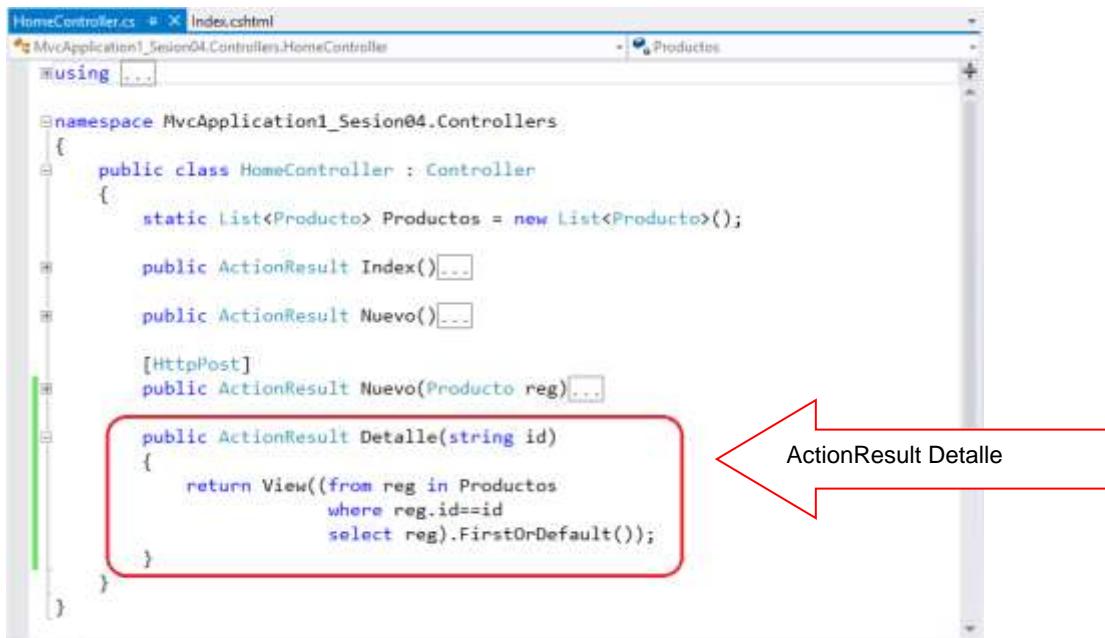
```

Para verificar el proceso, presiona la tecla F5, al presionar el botón Nuevo, se muestra una ventana modal para el ingreso de productos. Ingrese los datos, al presionar el botón Create, se cierra la ventana modal y se visualiza la ventana principal con el registro agregado



## Trabajando con el ActionResult Detalle

En el controlador Home, defina la acción Detalle, donde retorna el producto seleccionado por su campo id, tal como se muestra.



```

HomeController.cs * X Index.cshtml
MvcApplication1_Sesion04.Controllers.HomeController
Products

using ...

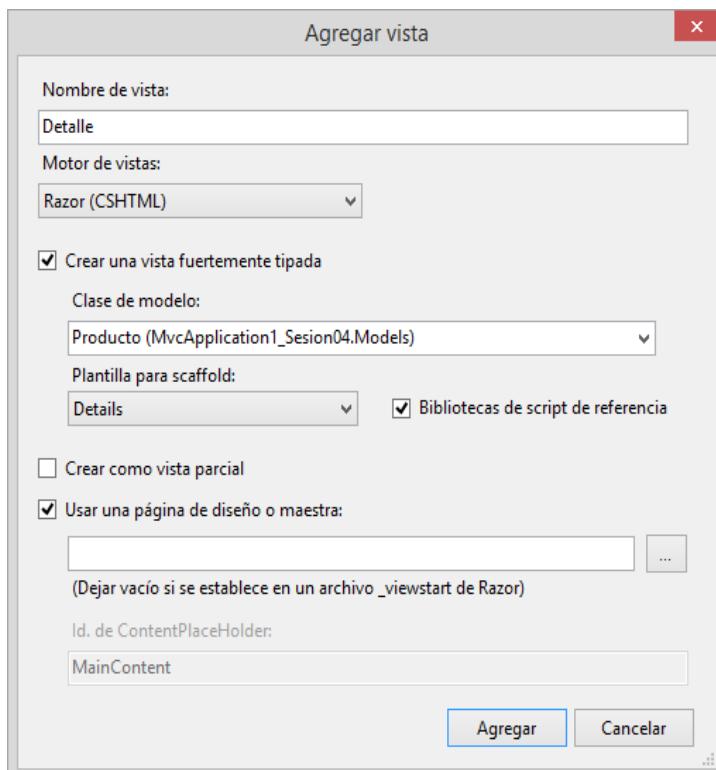
namespace MvcApplication1_Sesion04.Controllers
{
    public class HomeController : Controller
    {
        static List<Producto> Productos = new List<Producto>();

        public ActionResult Index()...
        public ActionResult Nuevo()...

        [HttpPost]
        public ActionResult Nuevo(Producto reg)...
        public ActionResult Detalle(string id)
        {
            return View((from reg in Productos
                        where reg.id==id
                        select reg).FirstOrDefault());
        }
    }
}

```

Defina la vista para la acción Detalle. Selecciona la clase de modelo Producto y la plantilla Details, tal como se muestra.



En la ventana Detalle.cshtml, realice los cambios tal como se muestra.



```

@model MvcApplication1_Sesion04.Models.Producto

@{ Layout = null; }

<h2>Detalle</h2>

<fieldset>
    <legend>Producto</legend>
    <div class="display-label">@Html.DisplayNameFor(model => model.descripcion)</div>
    <div class="display-field">@Html.DisplayFor(model => model.descripcion)</div>

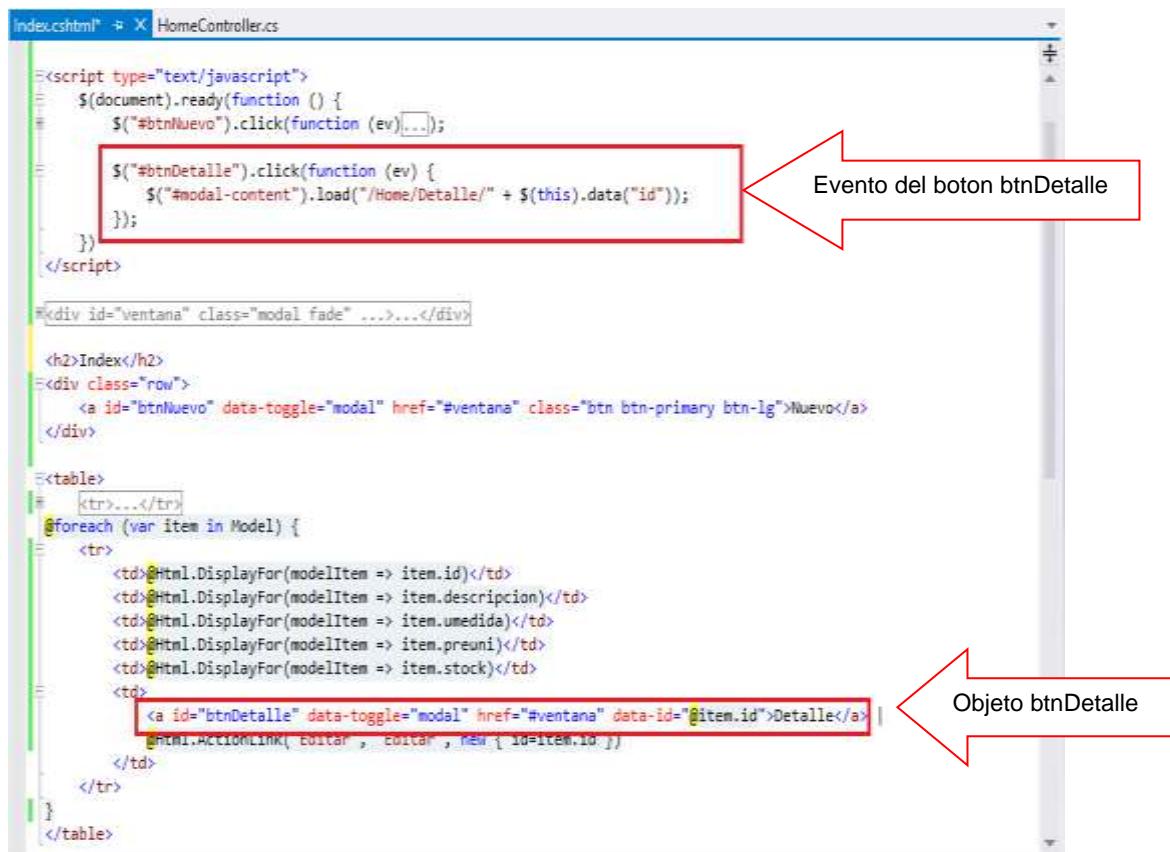
    <div class="display-label">@Html.DisplayNameFor(model => model.umedida)</div>
    <div class="display-field">@Html.DisplayFor(model => model.umedida)</div>

    <div class="display-label">@Html.DisplayNameFor(model => model.preuni)</div>
    <div class="display-field">@Html.DisplayFor(model => model.preuni)</div>

    <div class="display-label">@Html.DisplayNameFor(model => model.stock)</div>
    <div class="display-field">@Html.DisplayFor(model => model.stock)</div>
</fieldset>
<p>
    <button type="button" class="close" data-dismiss="modal">Retornar</button>
</p>

```

Regresamos a la vista Index para agregar el objeto btnDetalle y definimos su evento para mostrar en la ventana modal los datos del producto seleccionado



```

<script type="text/javascript">
$(document).ready(function () {
    $("#btnNuevo").click(function (ev) {
        $("#modal-content").load("/Home/Detalle/" + $(this).data("id"));
    });
})
</script>

<div id="ventana" class="modal fade" ...>...</div>

<h2>Index</h2>
<div class="row">
    <a id="btnNuevo" data-toggle="modal" href="#ventana" class="btn btn-primary btn-lg">Nuevo</a>
</div>

<table>
    <tr>...
    @foreach (var item in Model) {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.id)</td>
            <td>@Html.DisplayFor(modelItem => item.descripcion)</td>
            <td>@Html.DisplayFor(modelItem => item.umedida)</td>
            <td>@Html.DisplayFor(modelItem => item.preuni)</td>
            <td>@Html.DisplayFor(modelItem => item.stock)</td>
            <td>
                <a id="btnDetalle" data-toggle="modal" href="#ventana" data-id="@item.id">Detalle</a> | 
                @Html.ActionLink("Editar", "editar", "new", new { id=item.id })
            </td>
        </tr>
    }
</table>

```

Para comprobar su funcionalidad, presionar F5 para ejecutar el proyecto. Al presionar la opción Detalle, se visualiza los datos del producto seleccionado.

