Programación Orientada a Objetos II

ÍNDICE

PRESENTACIÓN

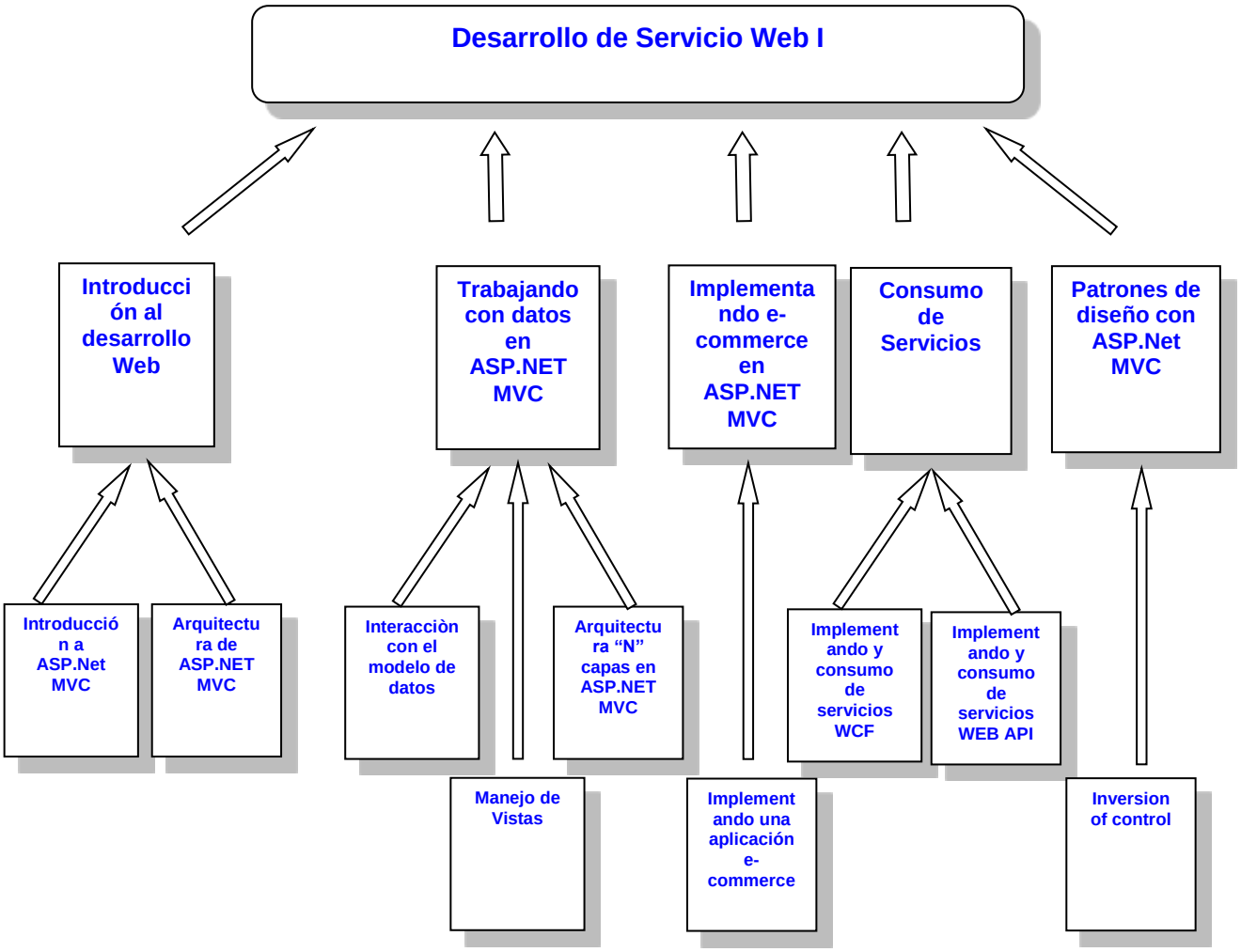
Visual Studio 2015 y su plataforma .NET FrameWork 4.5.2 permite implementar desarrollos de *software* de manera rápida y robusta. ASP .NET, tanto en Web Form como en MVC, admiten crear aplicaciones en tiempo mínimo bajo una plataforma de librerías del .NET Framework. De esta manera, la aplicación puede 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 ***Programación Orientada a Objetos II*** es un curso que pertenece a la línea de programación y desarrollo de aplicaciones con tecnología Microsoft, y se dicta en las carreras de TI de la institución. Brinda un conjunto de herramientas, plantillas y librerías de programación que permite a los alumnos desarrollar, en forma eficaz, soluciones a los problemas planteados en el curso.

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 un taller de programación con Visual Studio y el framework de MVC. En la primera parte del curso se desarrollan aplicaciones Web con Core ASP.NET MVC, acceso a datos utilizando las clases ADO.NET y el uso del motor de renderizado Razor para la vista de presentación. En la segunda parte del curso se profundiza el uso de la Arquitectura de Capas con de Dominio, implementando dicho proceso en una aplicación e-commerce. Por último, desarrollamos una aplicación Web para exportar los datos a un archivo de ReportViewer.
El curso es eminentemente práctico, consiste en un taller de programación con Visual Studio y el framework de MVC. En la primera parte del curso se desarrollan aplicaciones Web con Core ASP.NET MVC, acceso a datos utilizando las clases ADO.NET y el uso del motor de renderizado Razor para la vista de presentación. En la segunda parte del curso se profundiza el uso de la Arquitectura de Capas con de Dominio, implementando dicho proceso en una aplicación e-commerce. Por último, desarrollamos una aplicación Web para exportar los datos a un archivo de ReportViewer.

RED DE CONTENIDOS



UNIDAD DE APRENDIZAJE

1

INTRODUCCION AL ASP.NET CORE MVC

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 Core MVC (3 horas)

1. Introducción al patrón MVC
2. ASP.NET Core MVC
3. Plataforma ASP.NET Core MVC
4. Models
5. Vistas
6. Controllers
7. Razor y Scaffolding
8. URL de enrutamiento

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 practicas 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 prolijo 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:

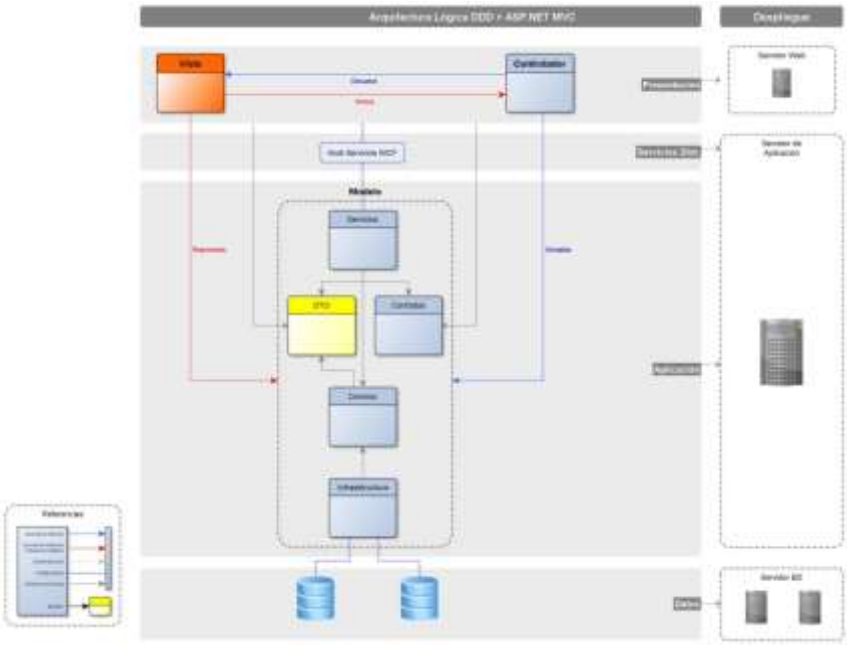


Figura: 1
Ref: https://jjestruch.wordpress.com/2012/02/21/arquitectura-ddd-domain-driven-design-asp-net-mvc/

¿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 ASP.NET Core MVC

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.

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 graficas 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.

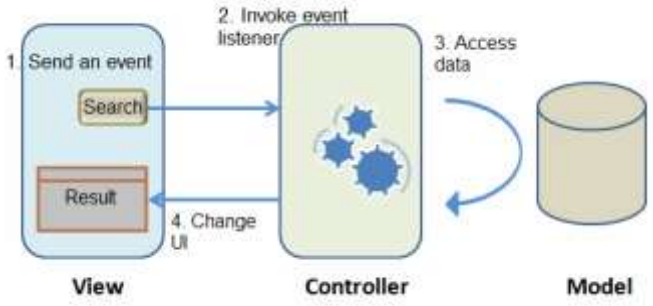


Figura: 2
Referencia: http://www.adictosaltrabajo.com/tutoriales/zk-mvc/

1.3 Plataforma ASP.NET MVC Core

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.

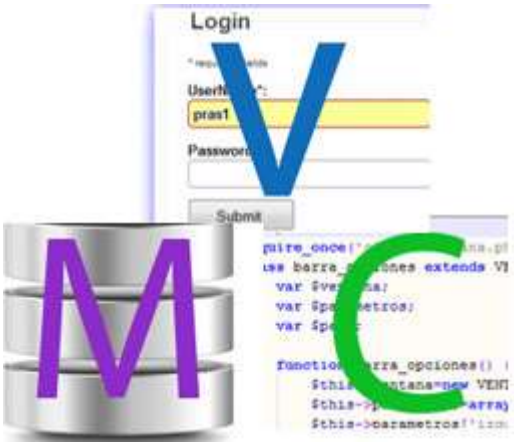


Figura: 3
Referencia: http://codigobase.com/el-porque-del-mvc-modelo-vista-controlador

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 sobrescritas 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.

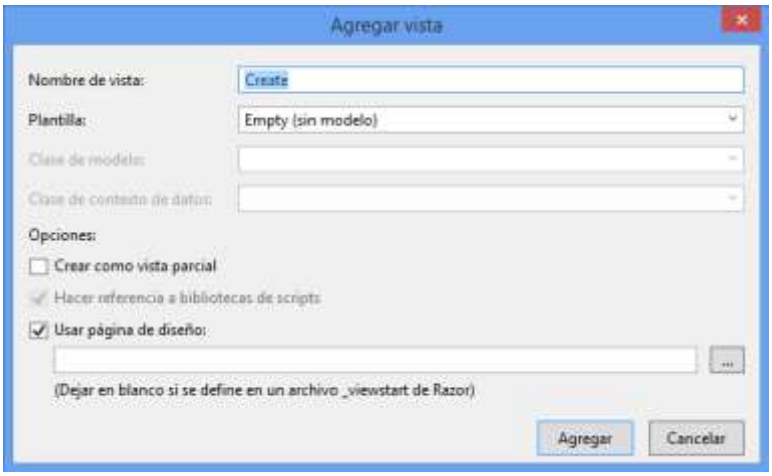
Mejoras para Razor: ASP.NET MVC 5 incluye la última view engine de Razor, la cual incluye mejor soporte para la resolución de referencias URL utilizando la sintaxis basada en tilde (~/), asi como también provee soporte para atributos HTML condicionales.

1.3.1 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 HTLM 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 seria parcial y solo se crearía una sola vez. Para crea 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 archivo 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().



1.3.2 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");
    }

}
```

1.1.3 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 scafold 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.

1.1.4 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:

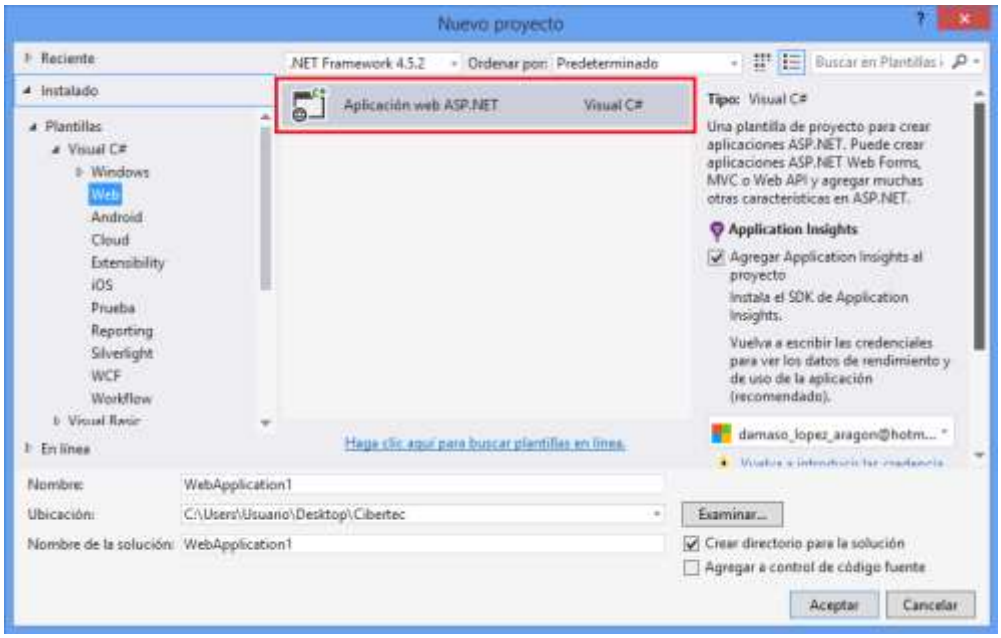
1. En cada petición URL no se asigna un archivo físico del disco como una página .aspx, sino que se asigna una acción de un controlador (más un parámetro), que nos mostrará una vista específica.
2. Las rutas son lógicas, es decir, siguen la estructura definida en la tabla de enrutamiento, lo que favorece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

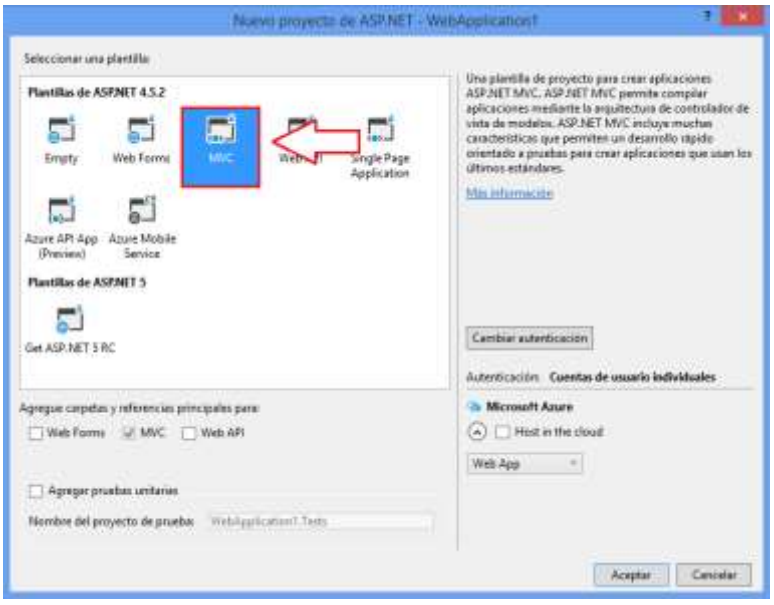
```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

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



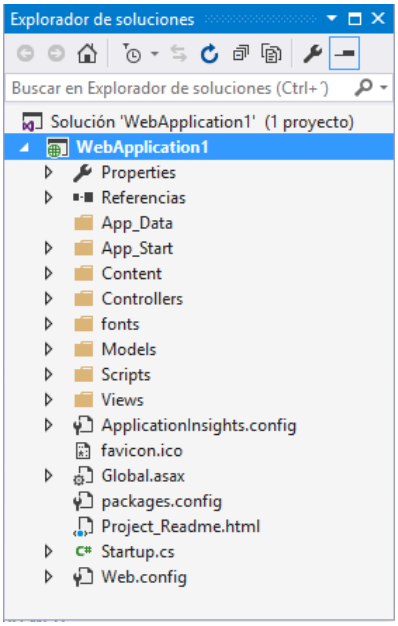
A continuación seleccionamos la plantilla del proyecto, el cual será de tipo MVC



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.

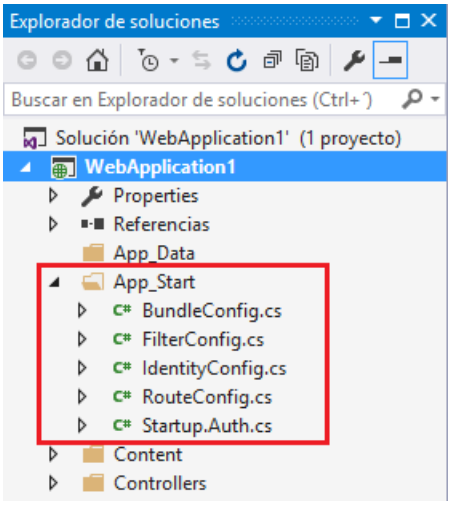


Directorio App_Data.

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 vacio.

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.

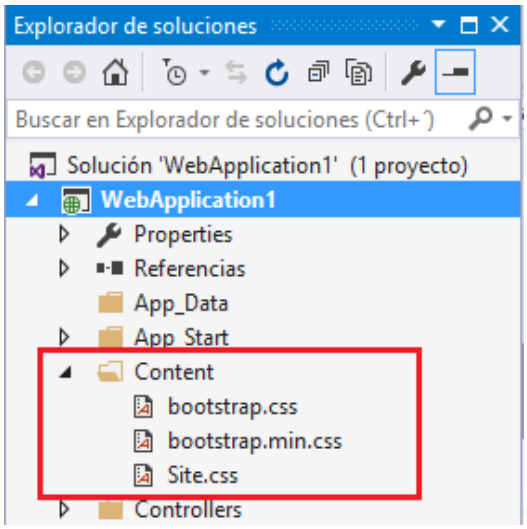


Por defecto contiene los siguientes archivos (clases):

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

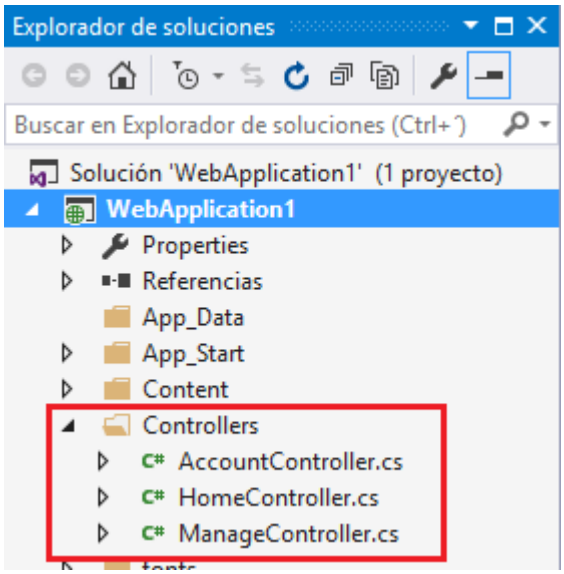
Directorio Content.

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 permita personalizar el aspecto visual de nuestra aplicación de forma fácil y rápida.



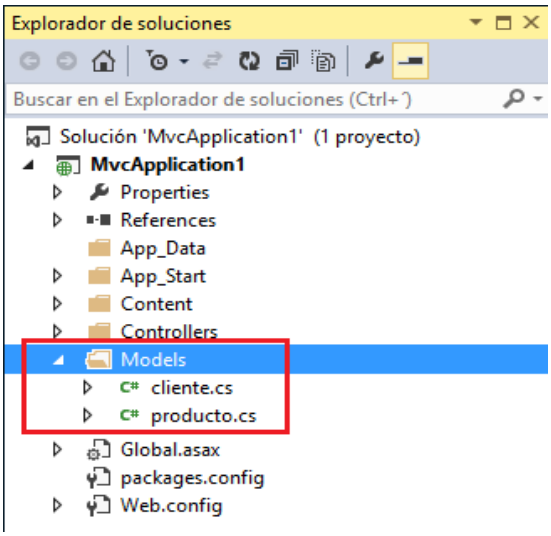
Directorio Controllers.

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.



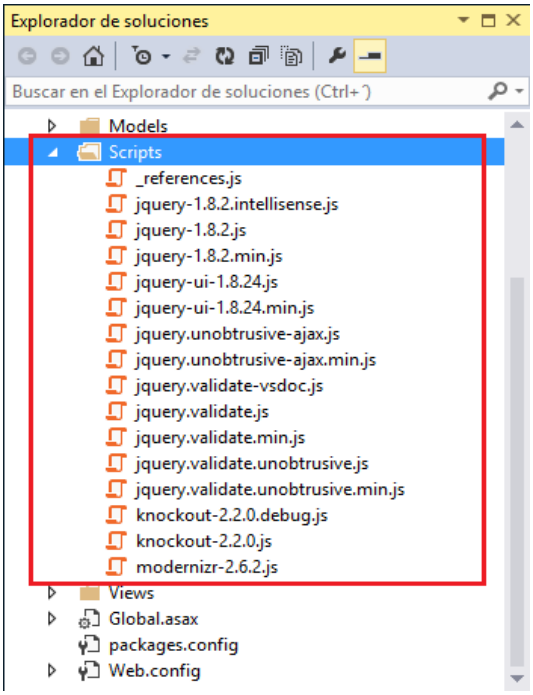
Directorio Models

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.



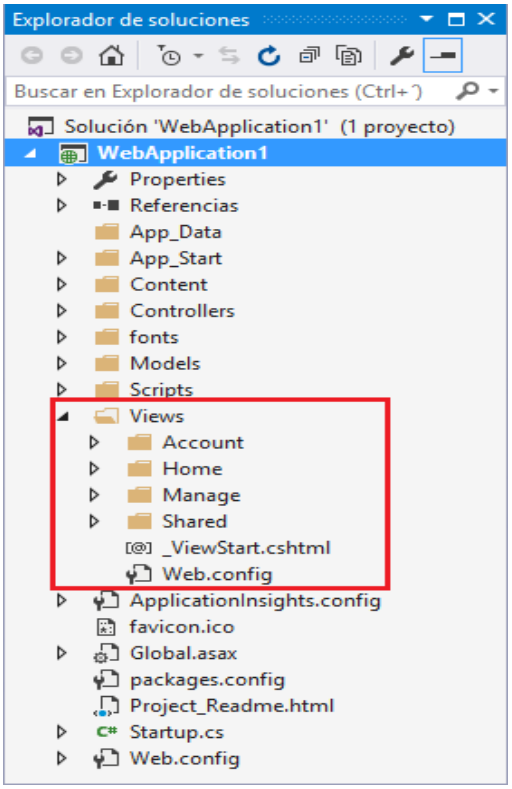
Directorio Scripts

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.



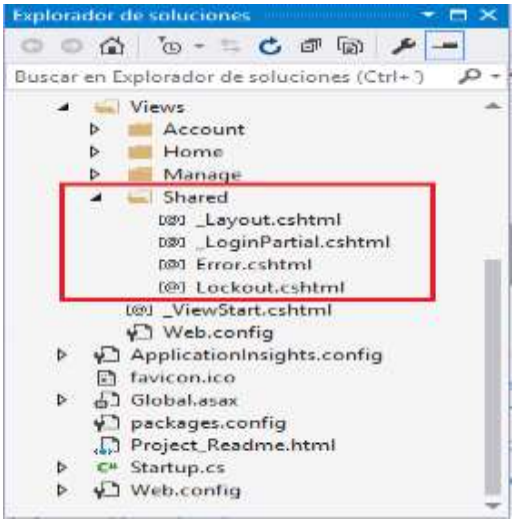
Directorio Views

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

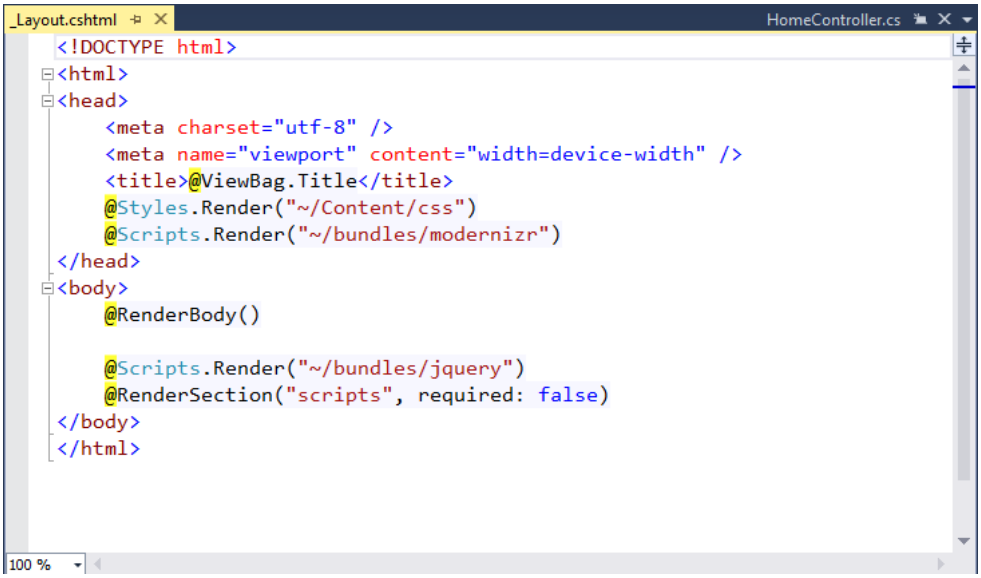
Este archivo establece el layout por defecto de las páginas. El contenido del archivo es sencillo y únicamente especifica el archivo de layout.

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

El layout es un archivo con extension .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

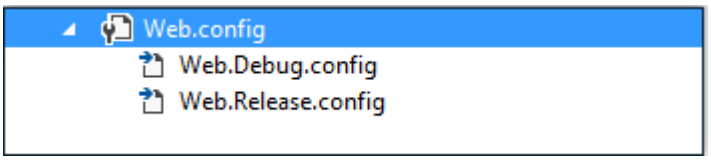
El archivo _Layout.cshtml definela capa de la aplicación, que contiene la estructura general de documento, que es reutilizada en el resto de vistas.
El archivo _Layout.cshtml se encuentra dentro del directorio Views/Shared. El contenido del archivo layout por defecto se muestra a continuación:



Fijemonos 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.

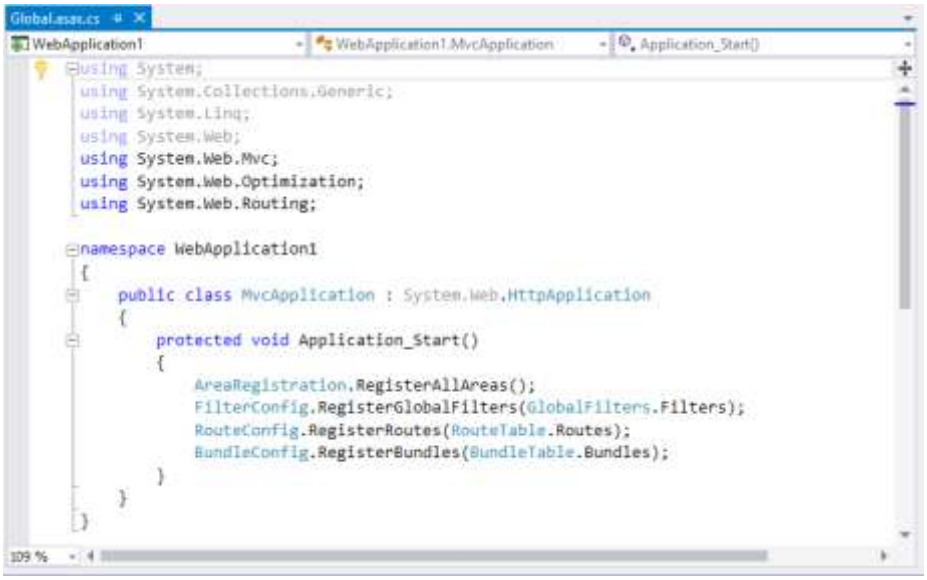


El archivo global.asax

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.

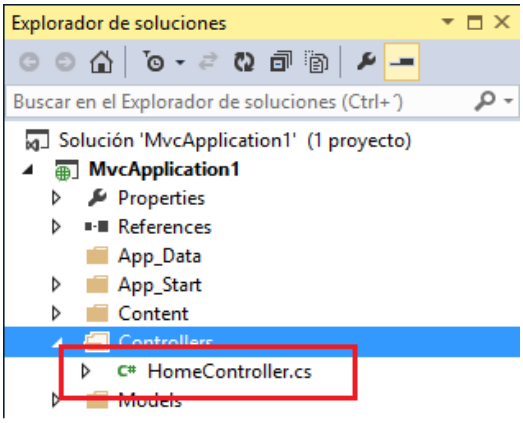
Desde este archivo podemos manejar eventos a nivel de aplicación, sesión, cache, autenticacion, etc.

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.



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.



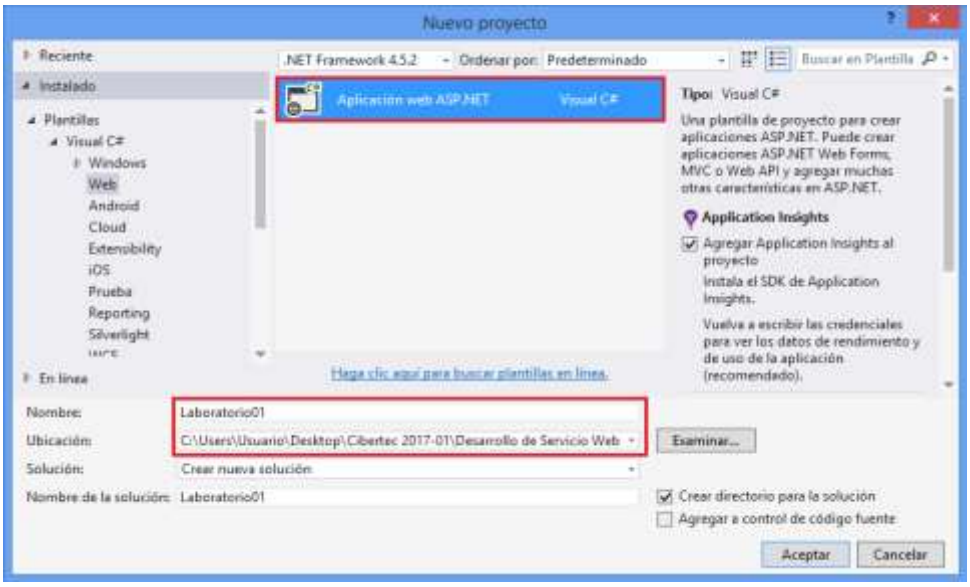
Laboratorio 1.1
Creando una aplicación ASP.NET MVC

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

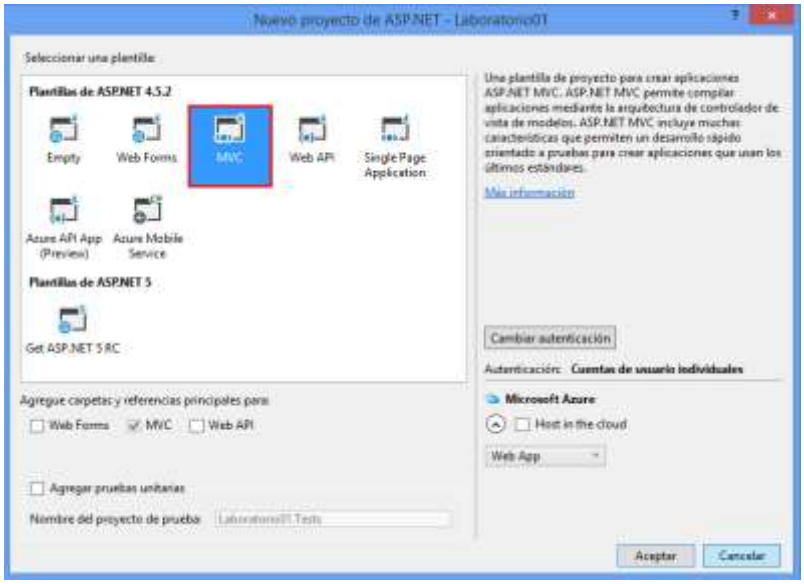
Creando el proyecto

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

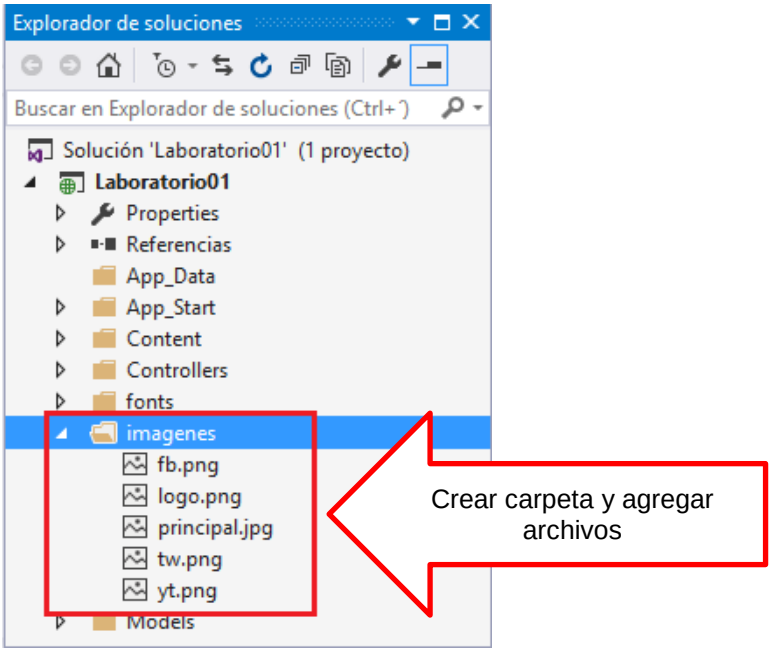


A continuación, seleccionar la plantilla del proyecto MVC. Presiona el botón ACEPTAR



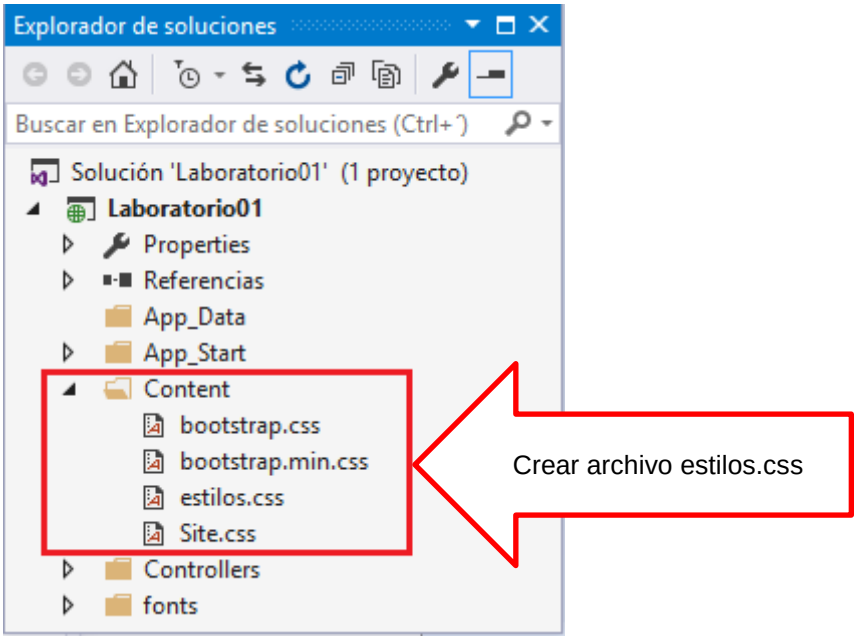
Agregando la carpeta imágenes

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.

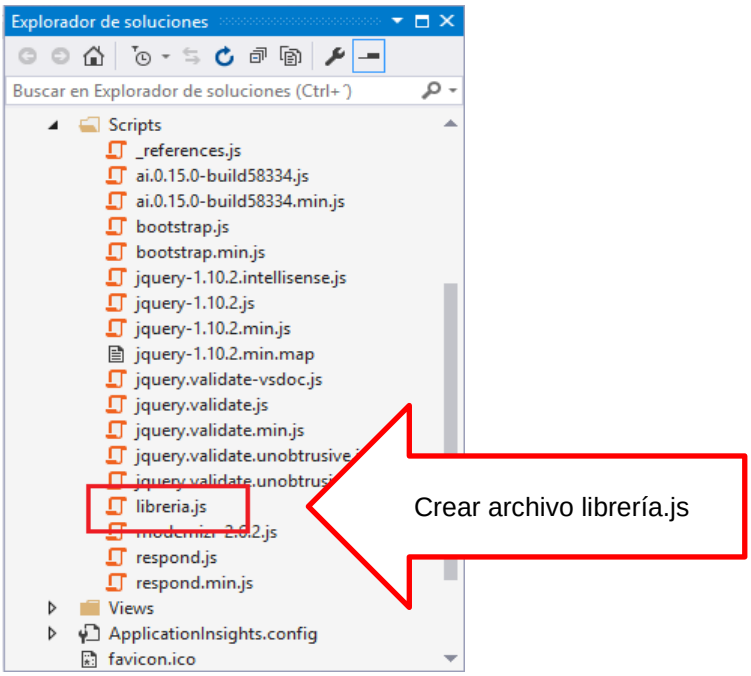


A continuación definimos estilos a las etiquetas que utilizará las páginas cshtml.

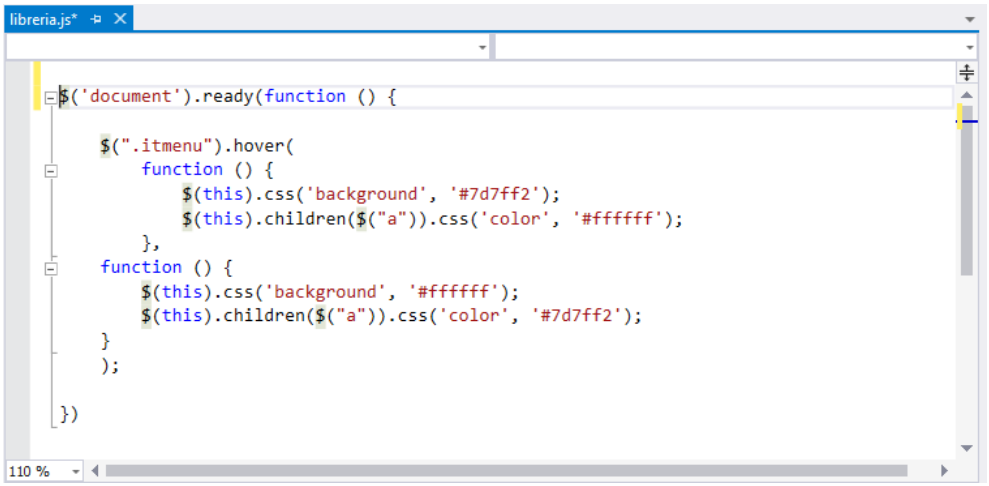
```
body {
    background:#7d7ff2;
}
header,nav,footer{
    width:100%; height:auto; float:left;
}
#logo{
    width:250px; height:130px; margin:0 auto;
    padding:20px;
}
#menu{
    width:100%; height:auto; float:left;
}

#menu tr td{
    width:19%; height:auto; background:#ffffff; 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:#ffffff;
}
```

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



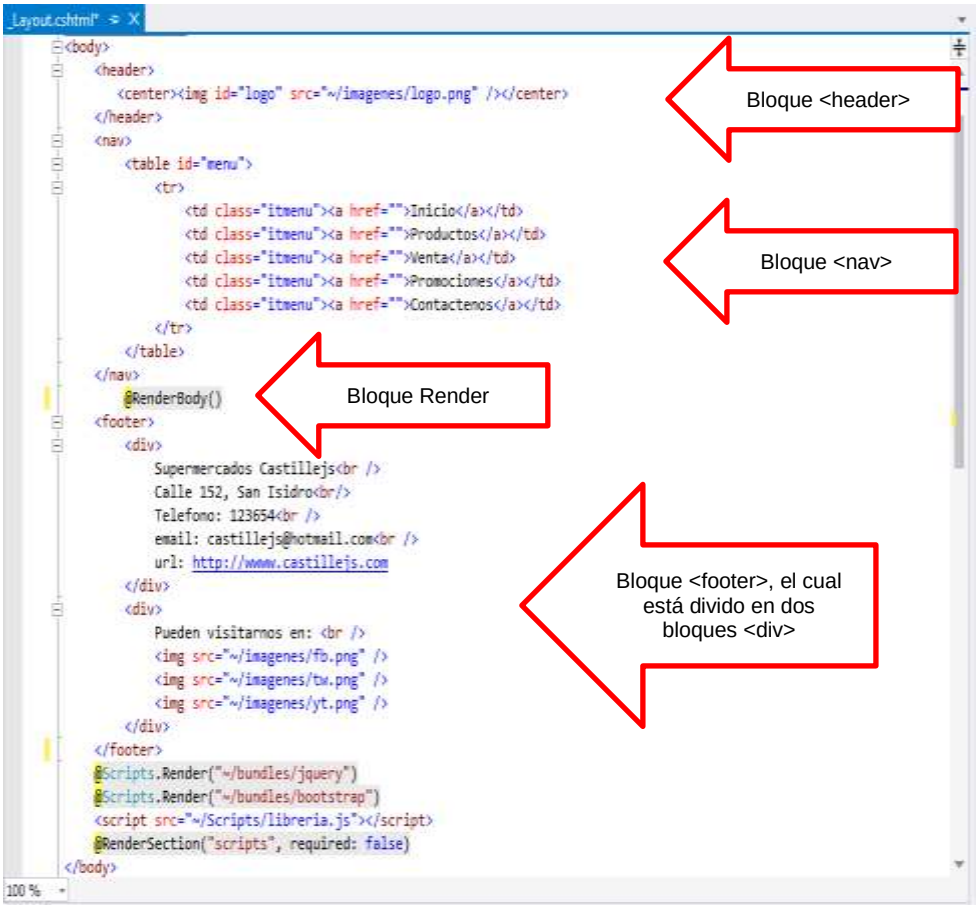
A continuación visualizamos el contenido del archivo js, donde programamos la clase .itmenu en el evento hover.



Trabajando con el _Layout

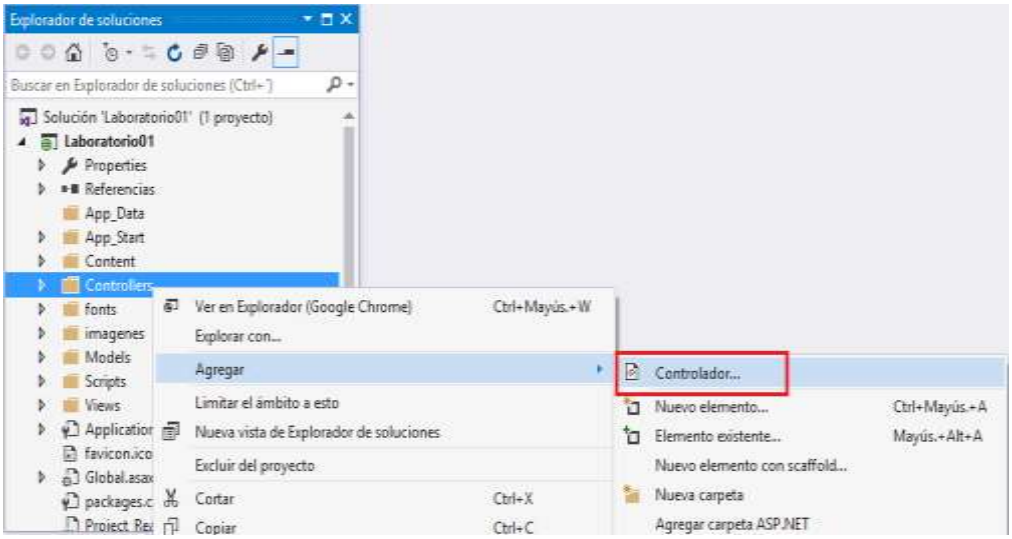
Abrir la pagina _Layout, para realizar el diseño de la página maestra.
Primero agregamos un link para enlazarnos al archivo estilo.css, y agregamos el link script para librería.js, tal como se muestra

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



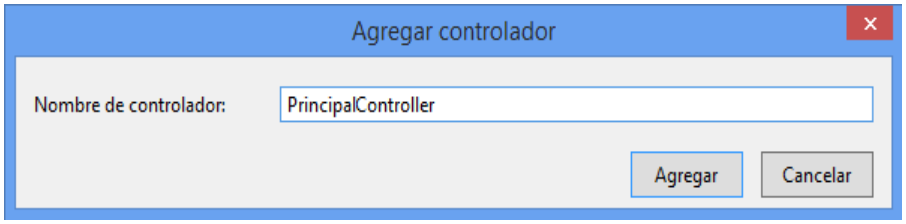
Agregando un Controlador al Proyecto

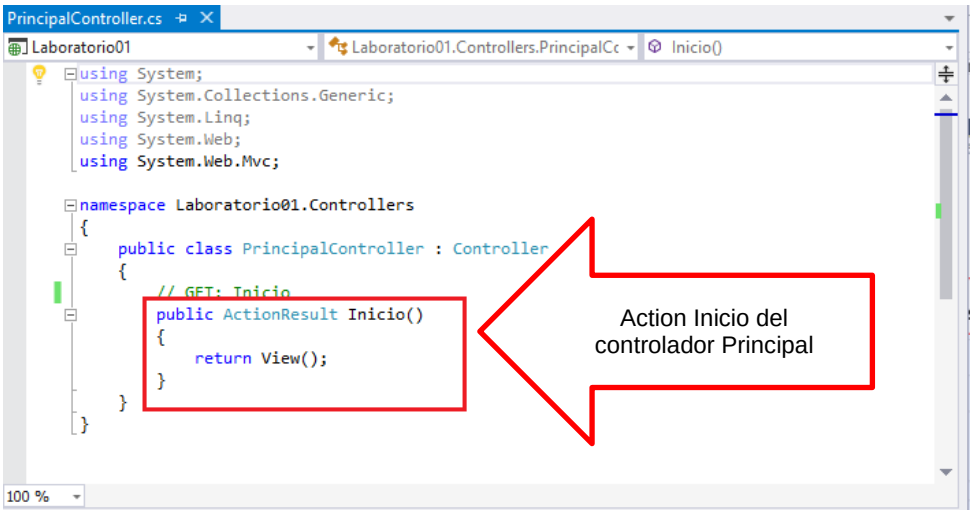
A continuación agregamos el controlador: En la carpeta Controllers, selecciona la opción **Agregar → Controlador**…, tal como se muestra en la figura.



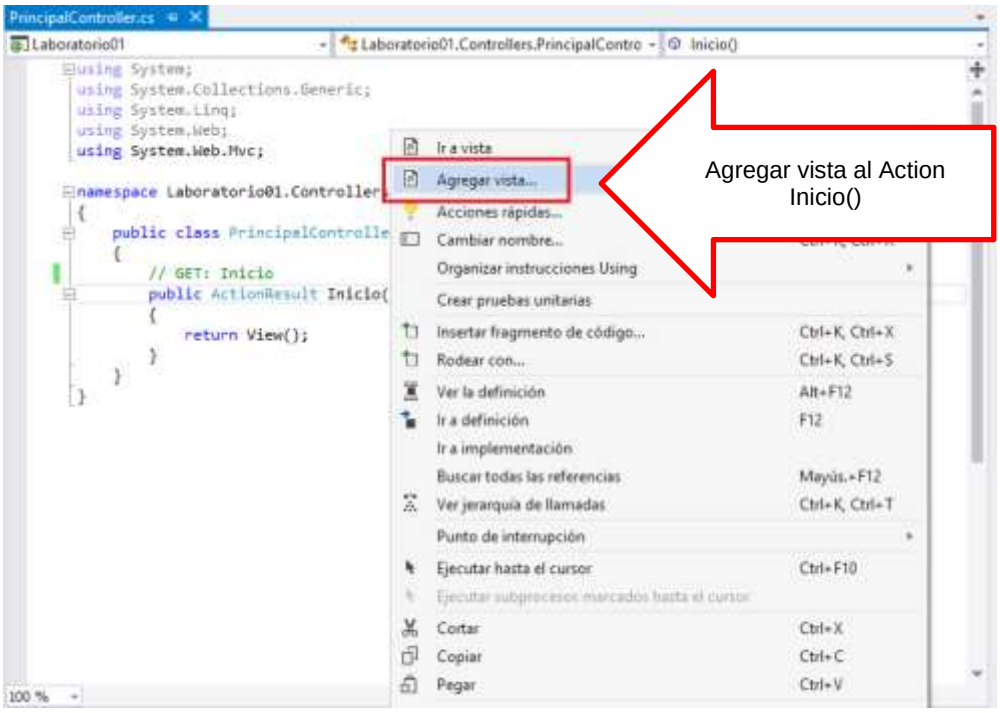
En la ventana Scafold, selecciona el tipo de controlador. Para nuestra aplicación seleccionamos el controlador en blanco, tal como se muestra.
A continuación presionar el botón Agregar

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

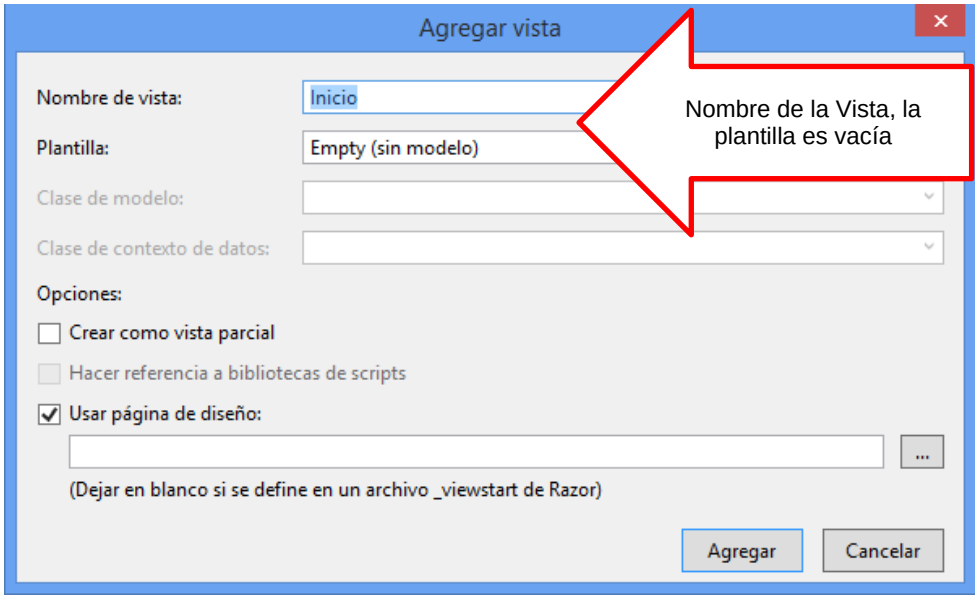




En el controlador, se muestra el ActionResult Inicio(). A continuación vamos a agregar una vista al Action.

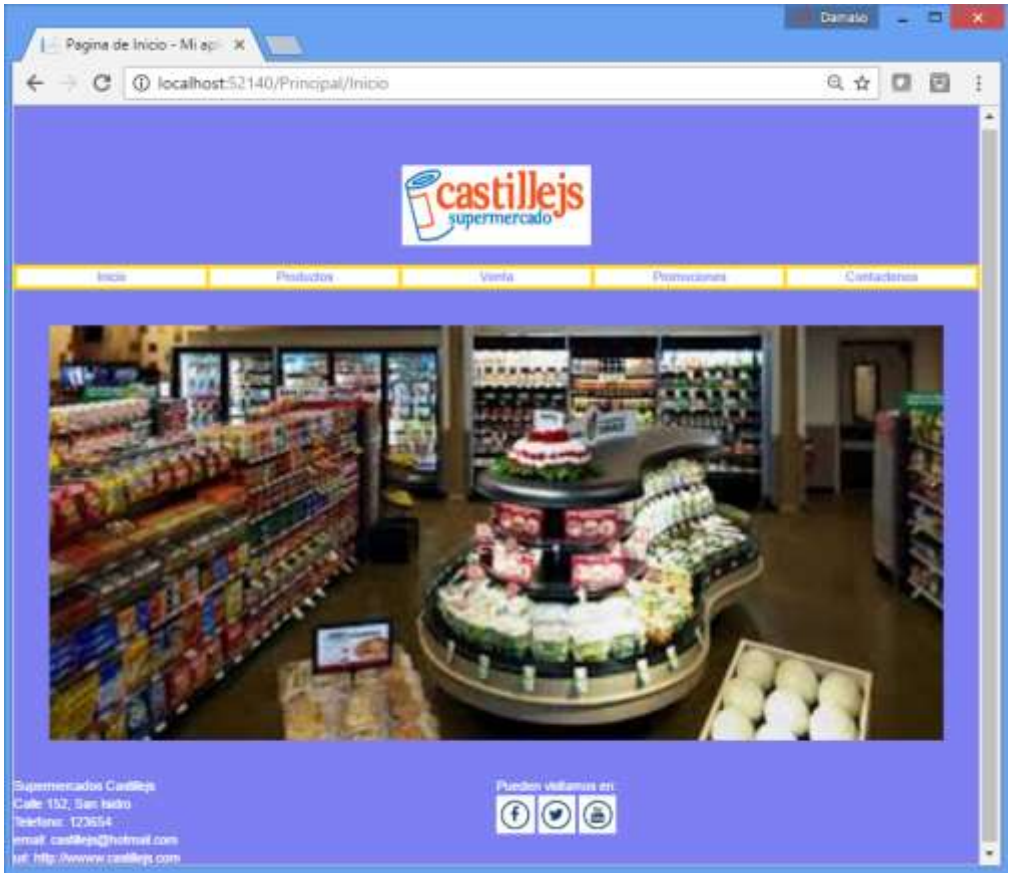


En la ventana Agregar vista, se visualiza el nombre de la vista. No hacer cambios, por ahora. Presiona el botón AGREGAR



A continuación se muestra la Vista inicio.cshtml.
Realice los cambios a la vista, tal como se muestra en la figura

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
  - http://es.slideshare.net/ogensollen/desarrollo-de-aplicaciones-web-con-aspnet-mvc5
  - http://nakedcode.net/?p=193

UNIDAD DE APRENDIZAJE

1

INTRODUCCION AL ASP.NET CORE MVC

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: ASP.NET Core MVC (6 horas)

2.1 Plataforma ASP.NET MVC
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. ASP.NET CORE 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

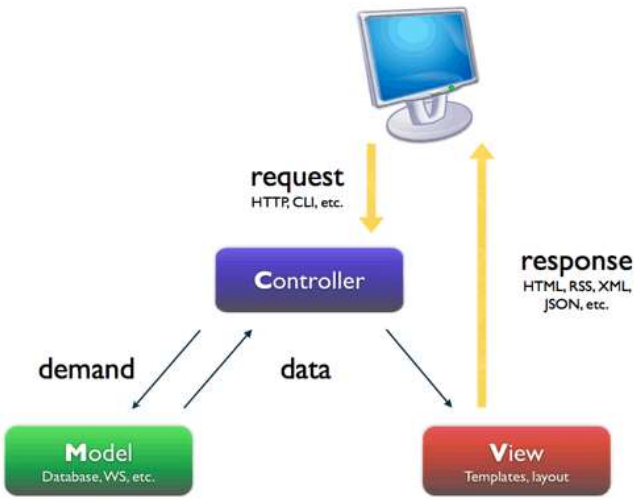


Figura: 1
Referencia: http://librosweb.es/libro/jobeet_1_4/capitulo_4/la_arquitectura_mvc.html

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 retornado 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.

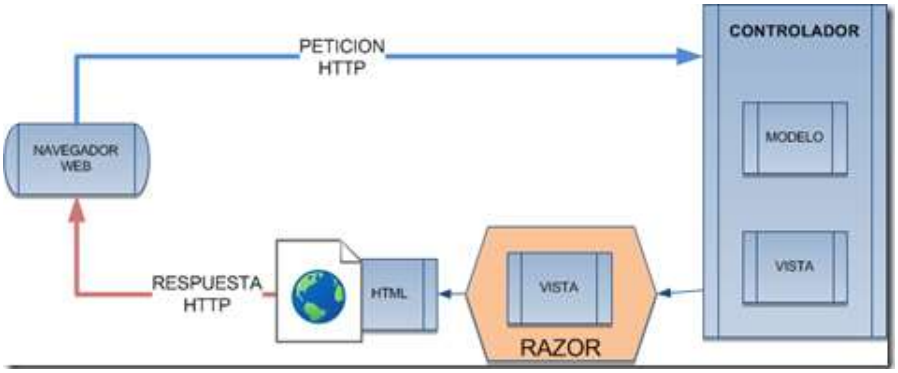


Figura: 2
Referencia: http://www.devjoker.com/contenidos/articulos/525/Patron-MVC-Modelo-Vista-Controlador.aspx

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.

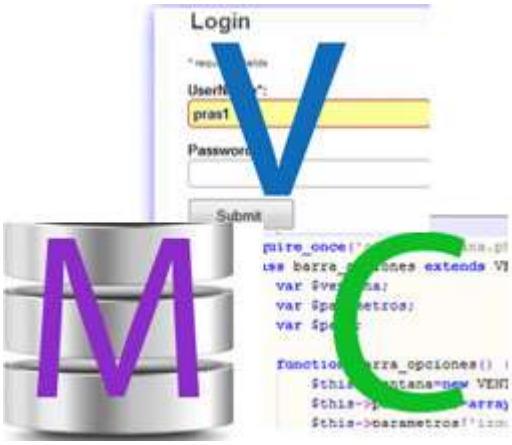


Figura: 3
Referencia: http://codigobase.com/el-porque-del-mvc-modelo-vista-controlador

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

2.1.1 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 unívoca 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 subruta 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	Fie	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();
}
```

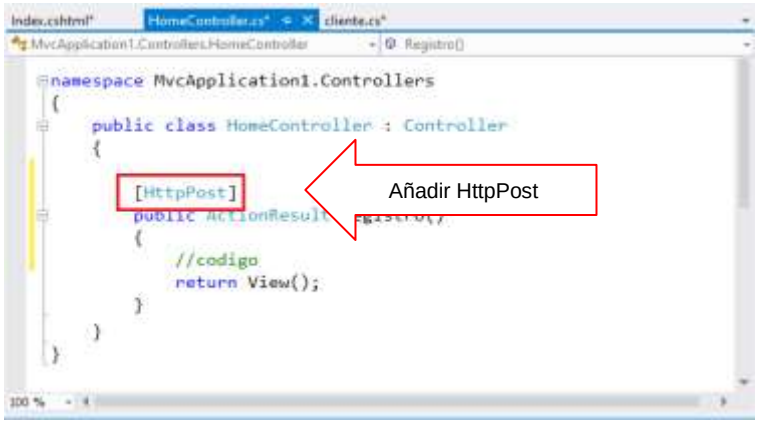
El marco de MVC también admite argumentos opcionales para los métodos de acción. Los parámetros opcionales en el marco de MVC se administran utilizando argumentos de tipo que acepta valores NULL para los métodos de acción de controlador. Por ejemplo, si un método puede tomar una fecha como parte de la 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.

2.1.2 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.

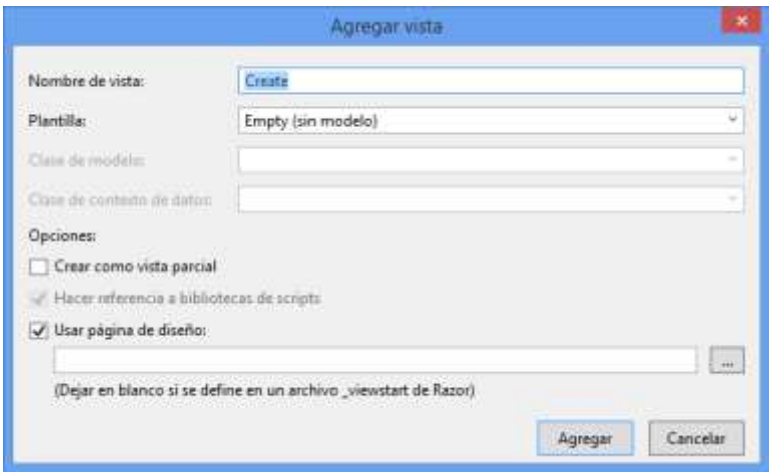
2.2 Vistas

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 HTLM 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 seria parcial y solo se crearía una sola vez. Para crea 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 archivo 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().



2.2.1 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 scafold 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.

2.2.2 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

2.2.3 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.

2.2.4 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.
- RequlatExpression - 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.

MaxLenghAttribute y MinLenghAttribute
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 ***MaxLenghtAttribute*** 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; }
```

RequieredAttribute
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; }
```

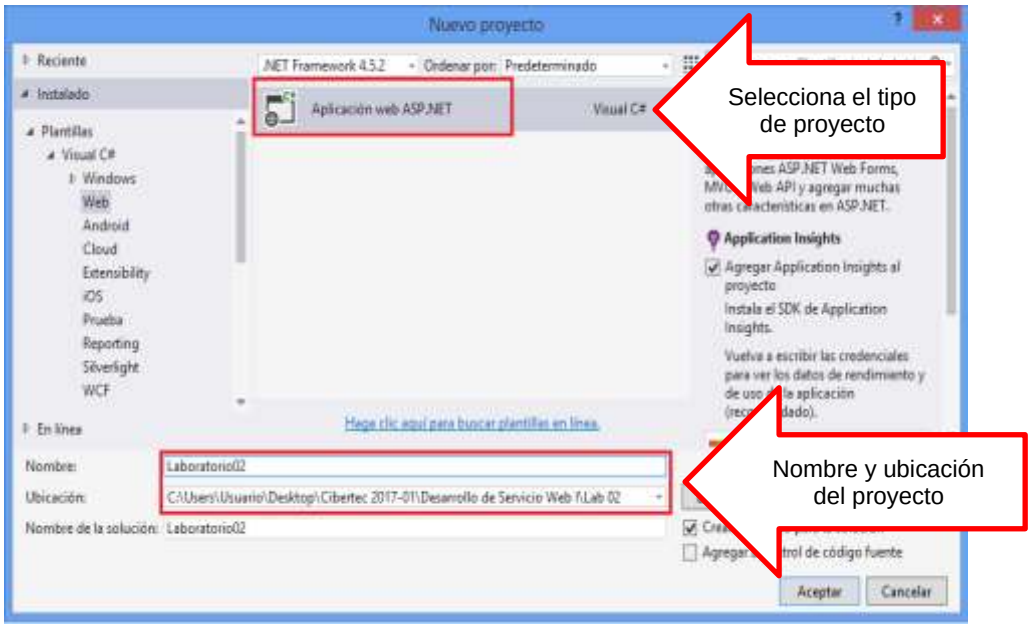
Laboratorio 2.1
Creando una aplicación ASP.NET MVC

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

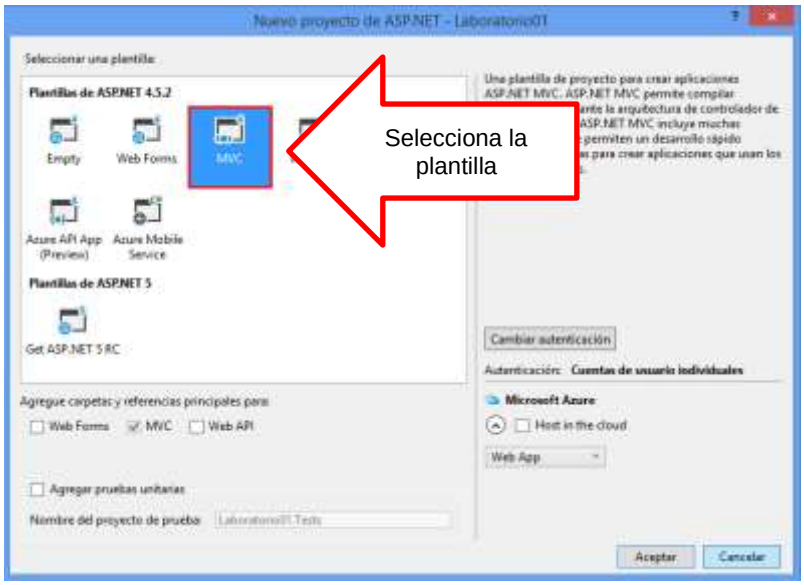
Creando el proyecto

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

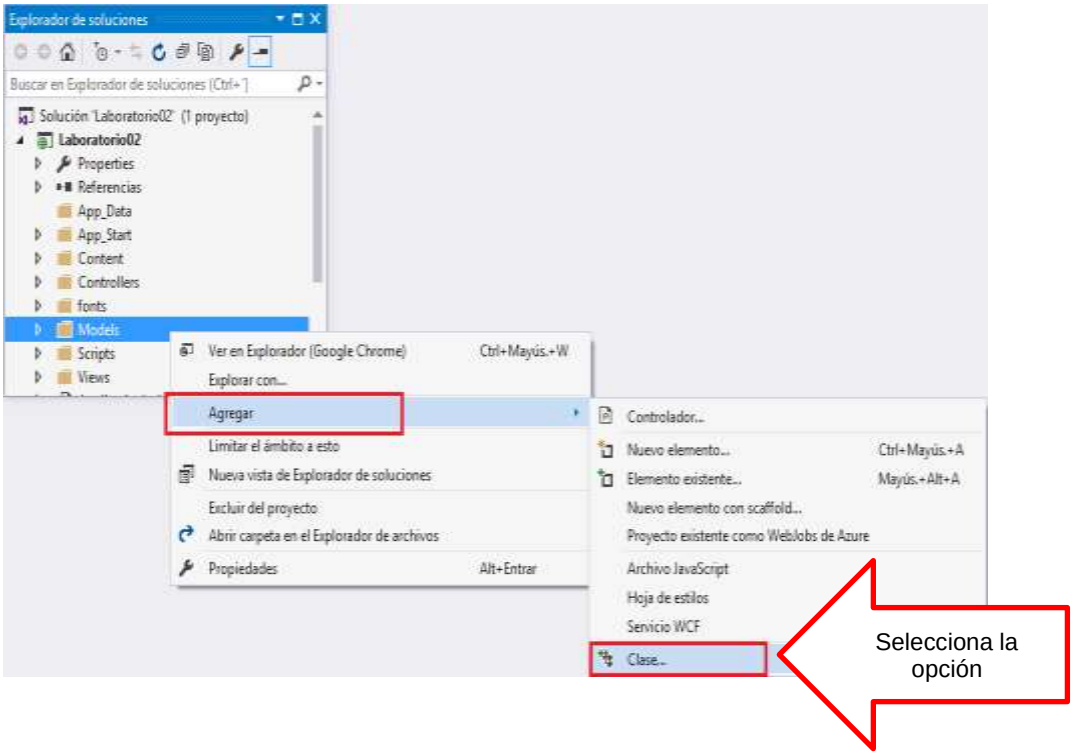


A continuación, seleccionar la plantilla del proyecto MVC. Presiona el botón ACEPTAR

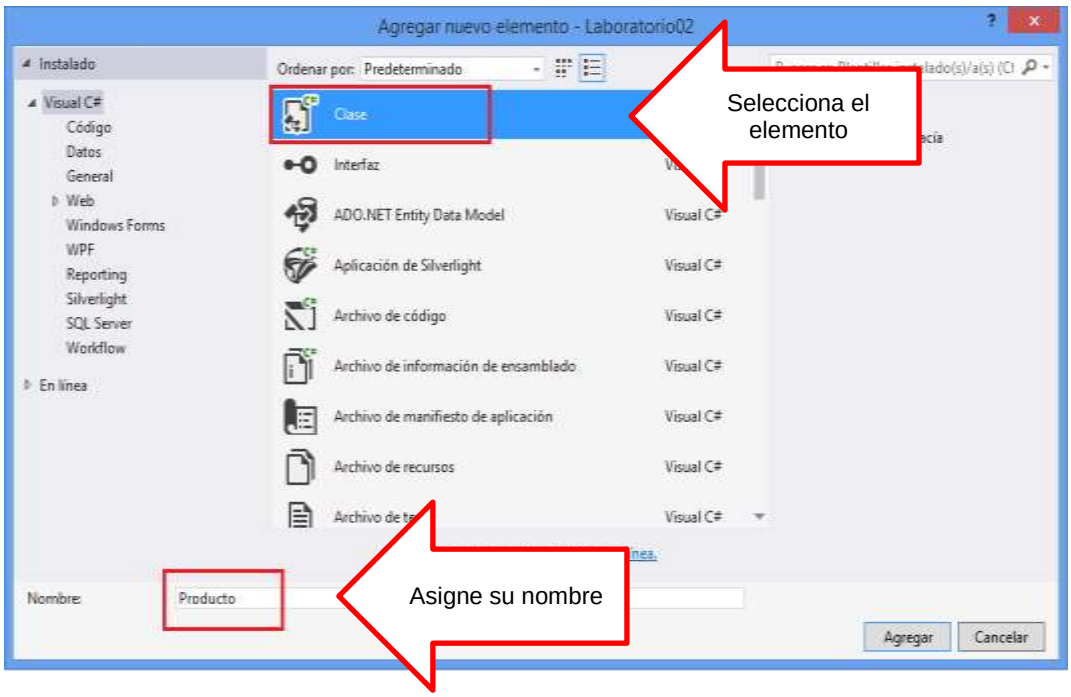


Trabajando con el Modelo

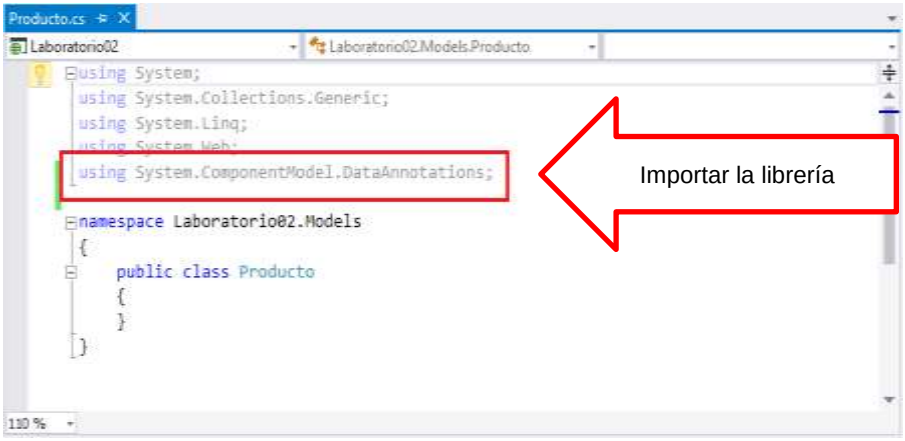
Primero, creamos una clase en la carpeta Models: Agregar una clase llamada Producto, tal como se muestra



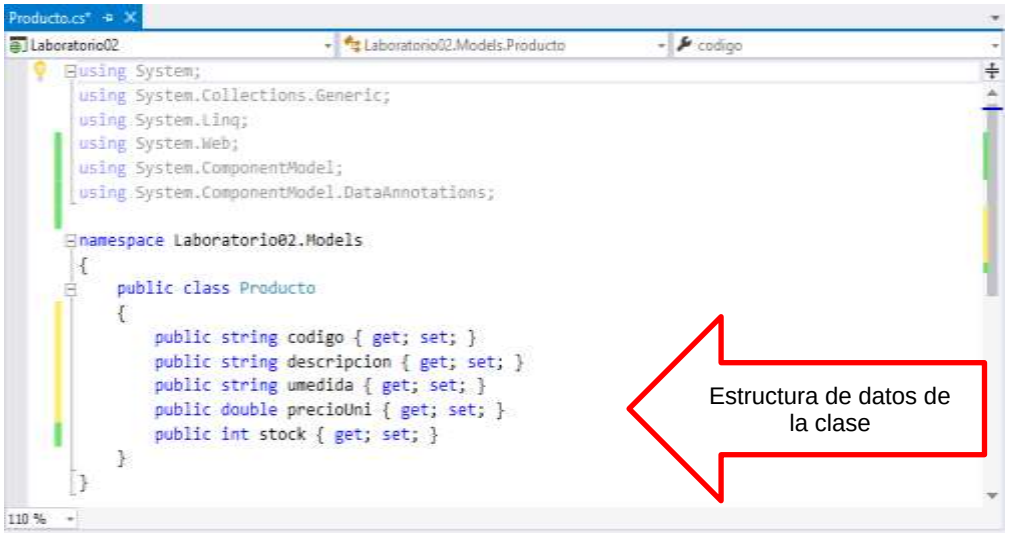
En la ventana **Agregar nuevo elemento**, selecciona el elemento **Clase** y asigne el nombre: Producto, presiona el botón Agregar



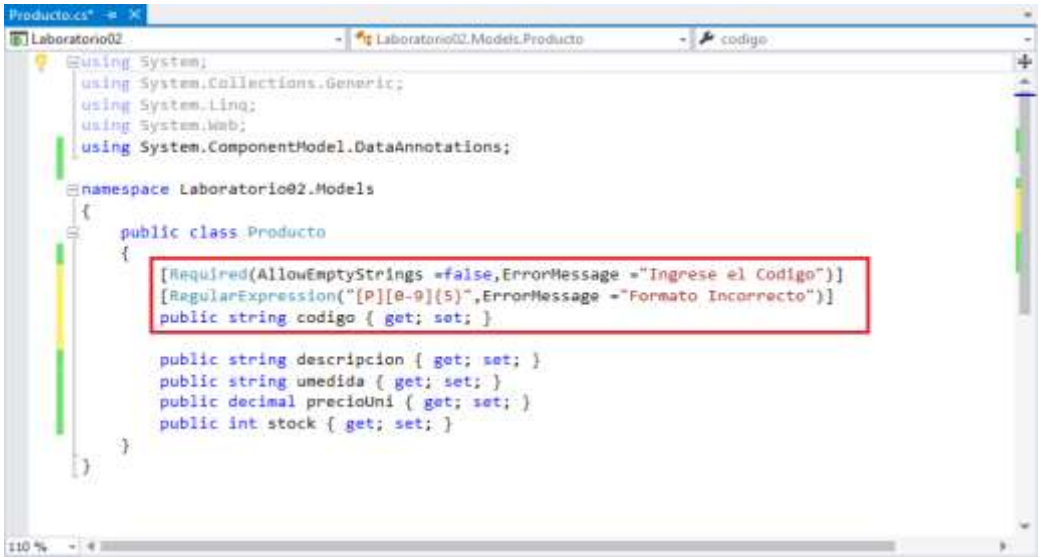
En la clase importar la librería de Anotaciones y Validaciones



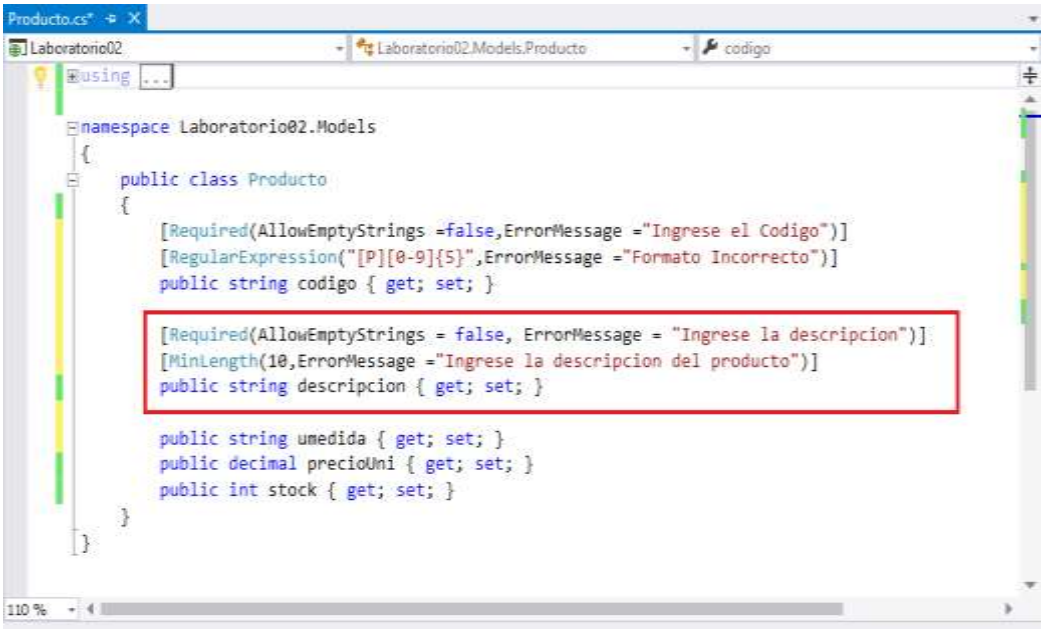
En la clase Producto, primero, defina la estructura de datos de la clase, tal como se muestra.



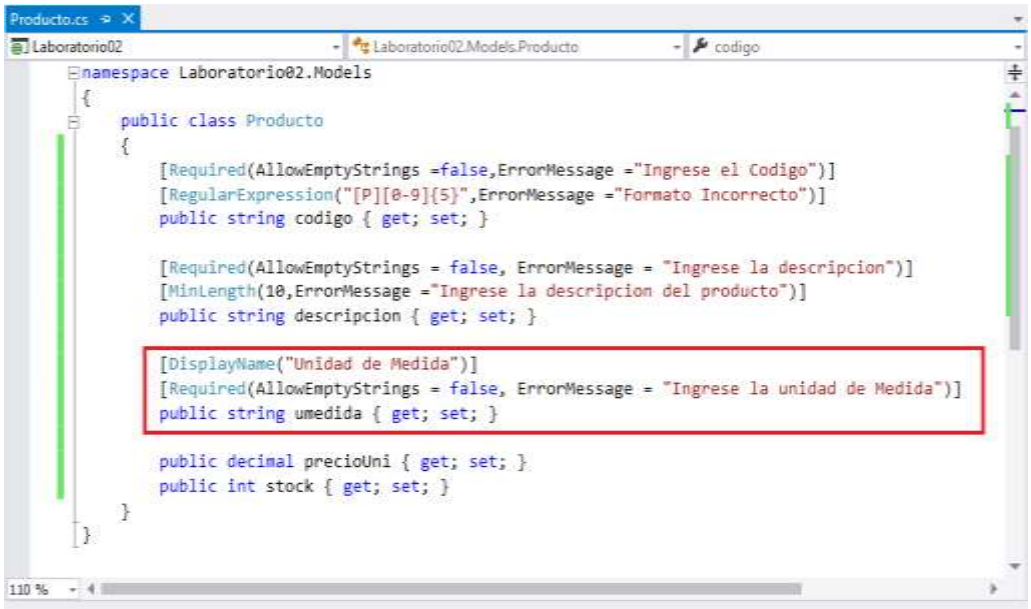
A continuación validamos el campo código: no debe estar vacío y su formato es P99999, tal como se muestra



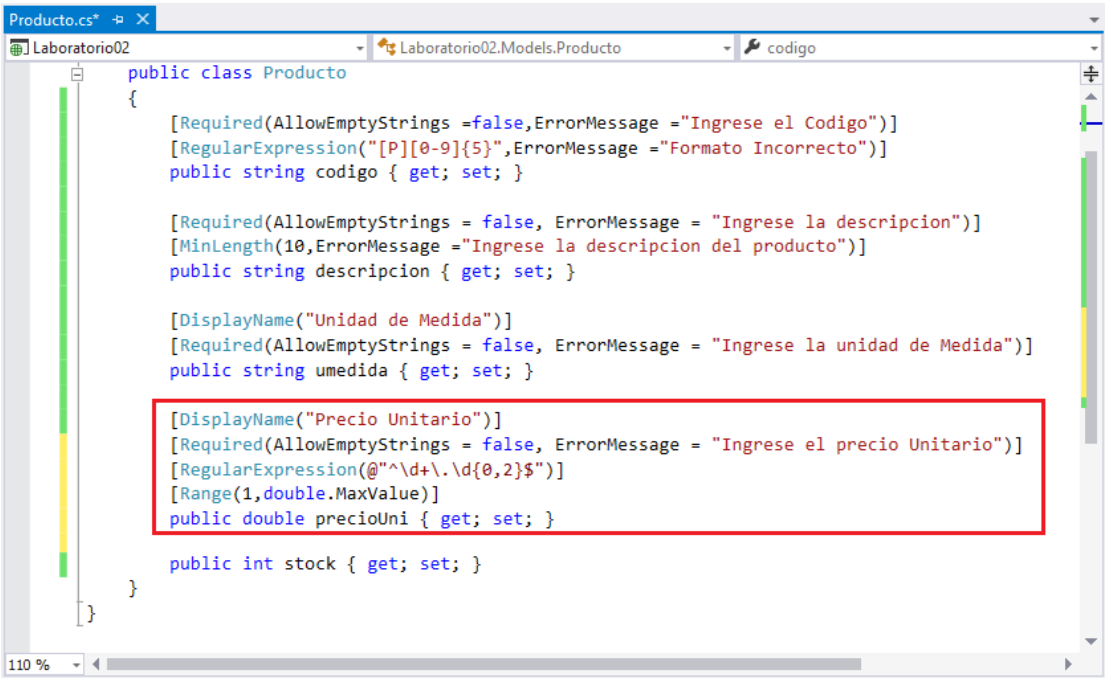
Luego validamos el campo descripción: no debe estar vacío y la longitud minima de caracteres es 10, tal como se muestra



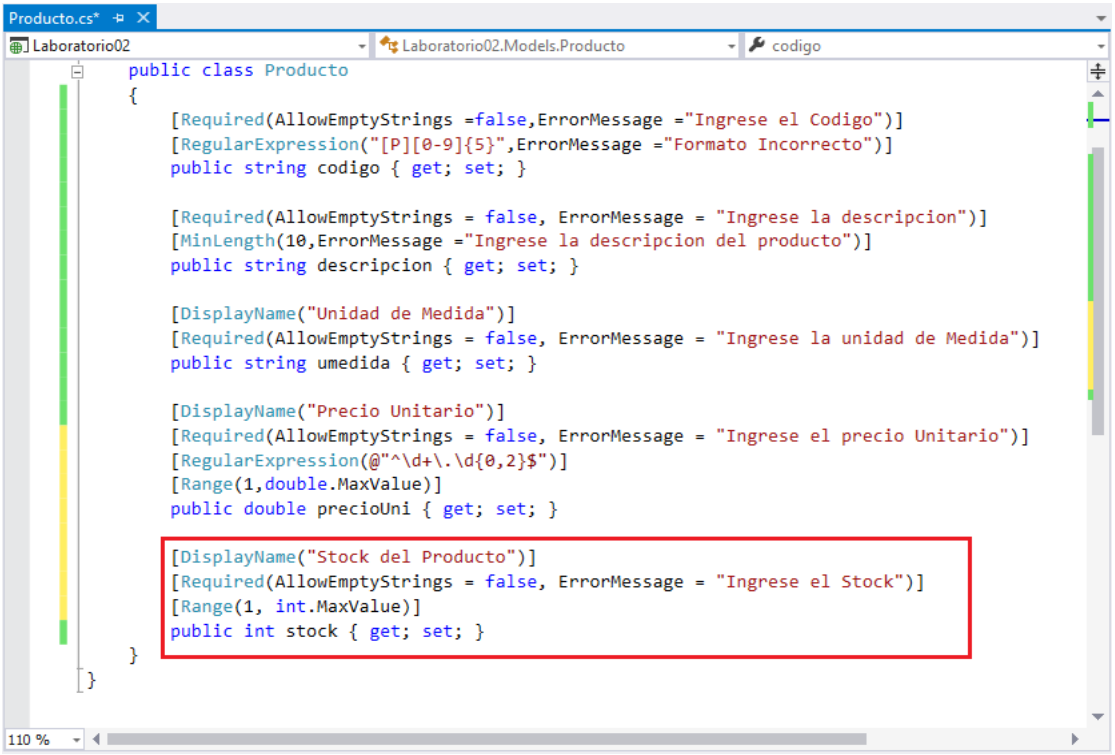
Continuamos con la validación del campo umedida, el cual será obligatorio, y asignamos el nombre a visualizar (DisplayName), importar la librería System.ComponentModel



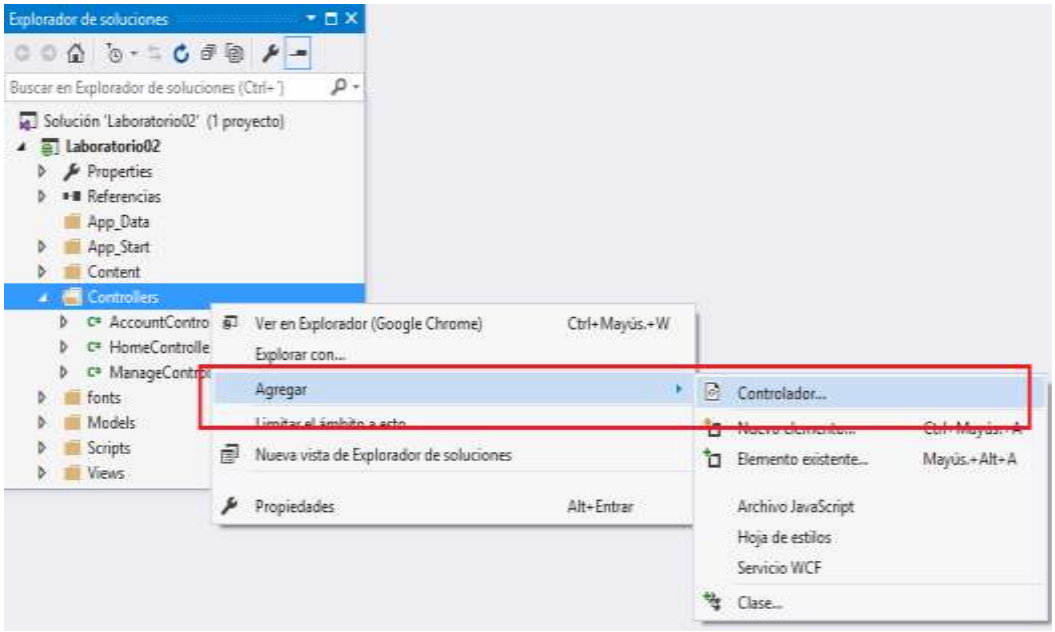
Luego validamos el campo preUni: asignar un Nombre para visualizarlo, indicar que el campo obligatorio y solo debe contener dos decimales (RegularExpression) y su rango de valores se encuentra entre 1 al valor máximo del tipo de dato.



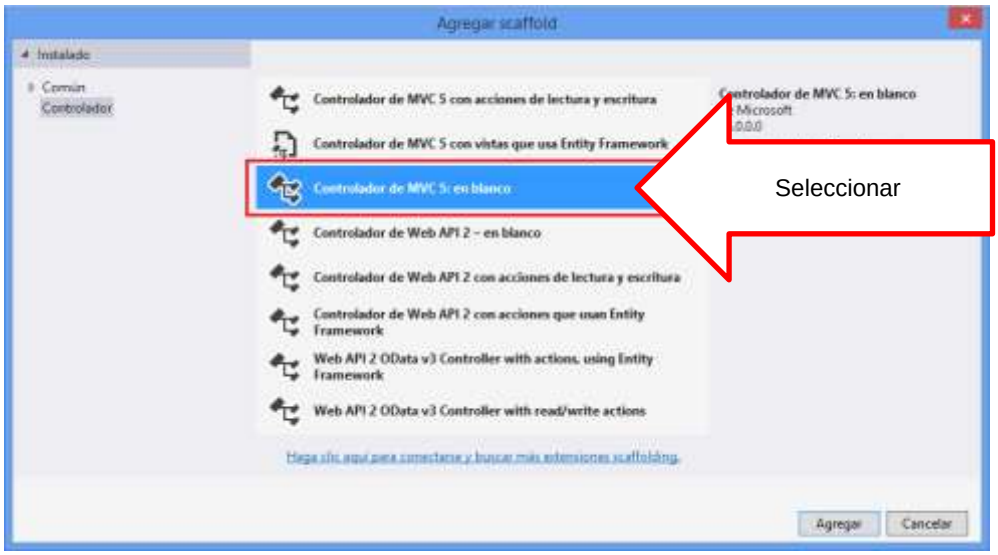
Y por ultimo validamos el campo stock: asignamos un nombre, campo obligatorio y cuyo rango de valores es 1 hasta el valor máximo del tipo de dato



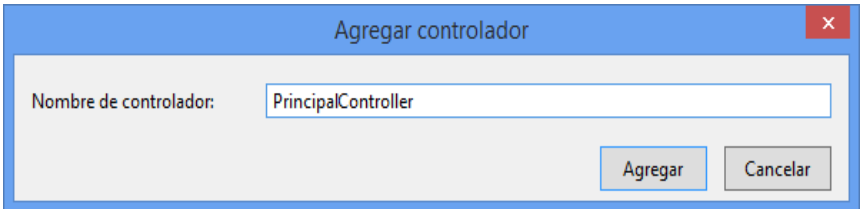
A continuación agregar en la carpeta Controller un controlador, tal como se muestra



En la ventana Scafold, selecciona el controlar MVC 5 en blanco, tal como se muestra

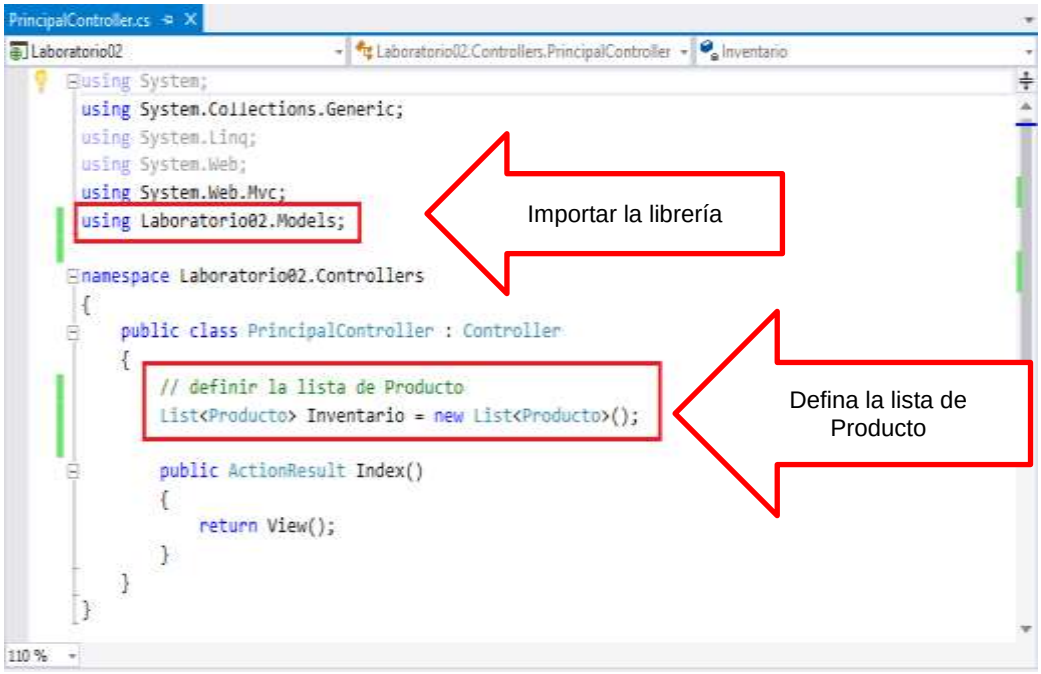


A continuación ingrese el nombre del Controlador, presiona al botón Agregar



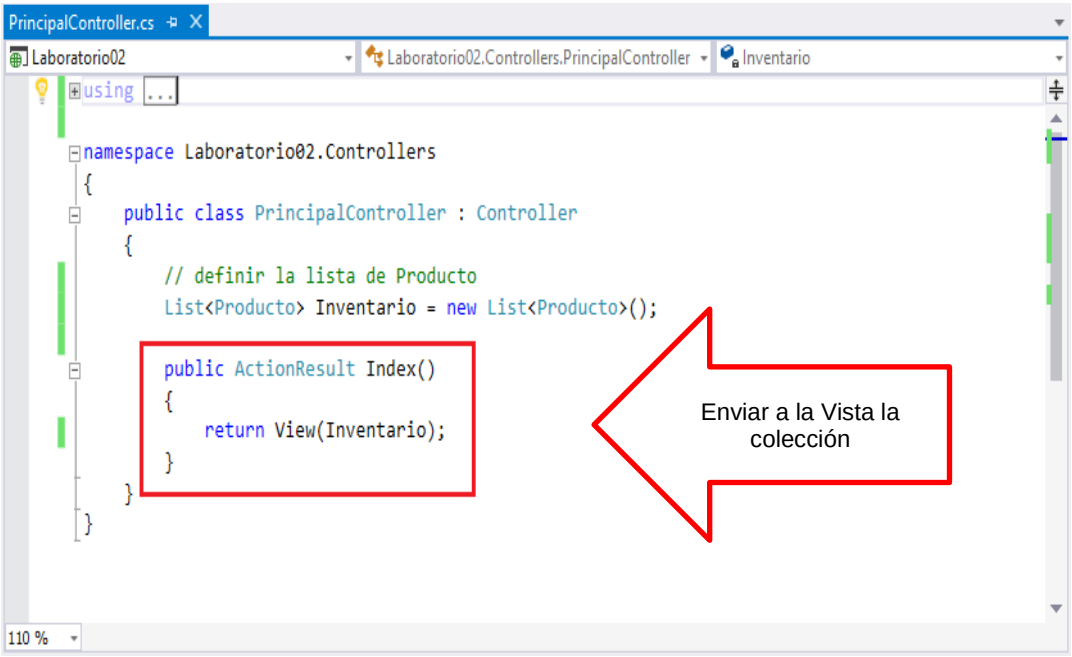
En la ventana de código PrincipalController:

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

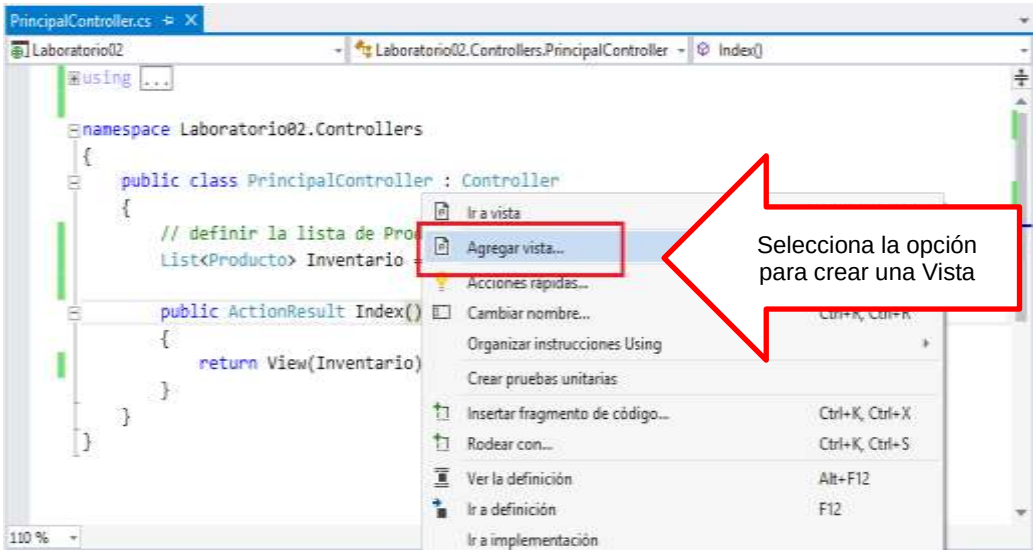


En el ActionResult Index(), enviar a la vista la lista de Producto, llamado Inventario. La sintaxis es:

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

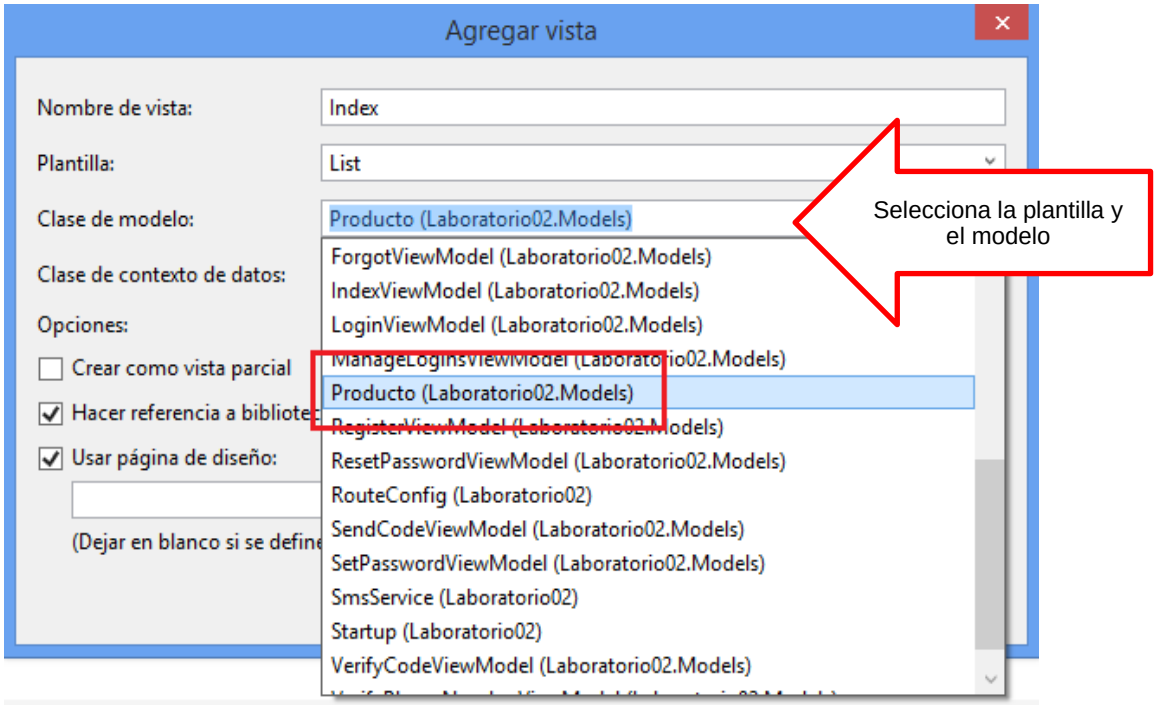


A continuación agregar una Vista a la acción Index: hacer click derecho a la acción y selecciona Agregar Vista, tal como se muestra



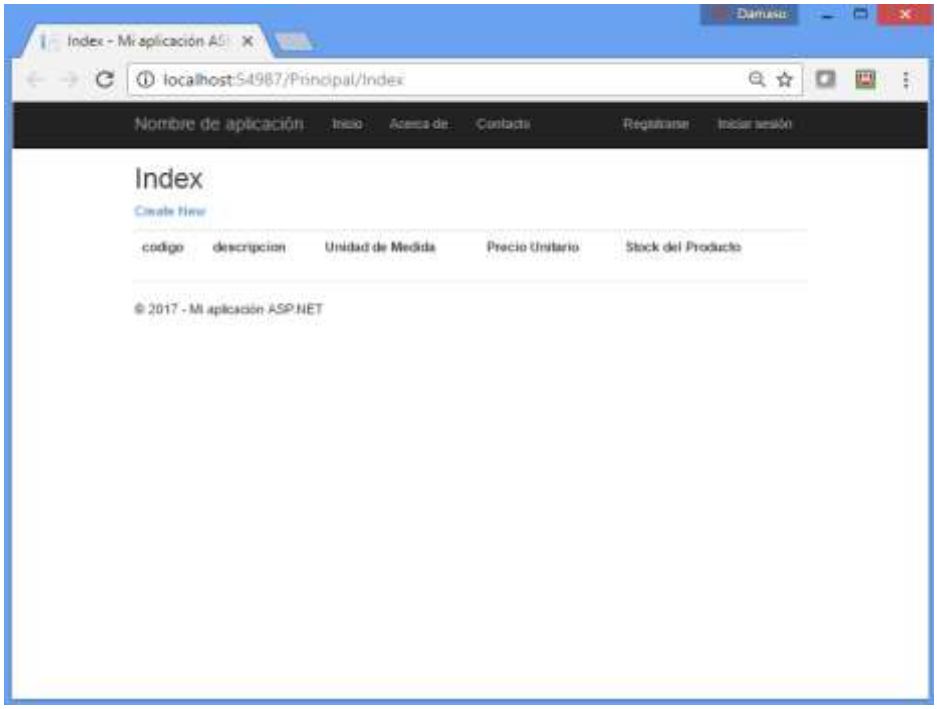
En la ventana Agregar Vista, aparece el nombre de la Vista: Index. No cambiar el nombre de la vista.

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

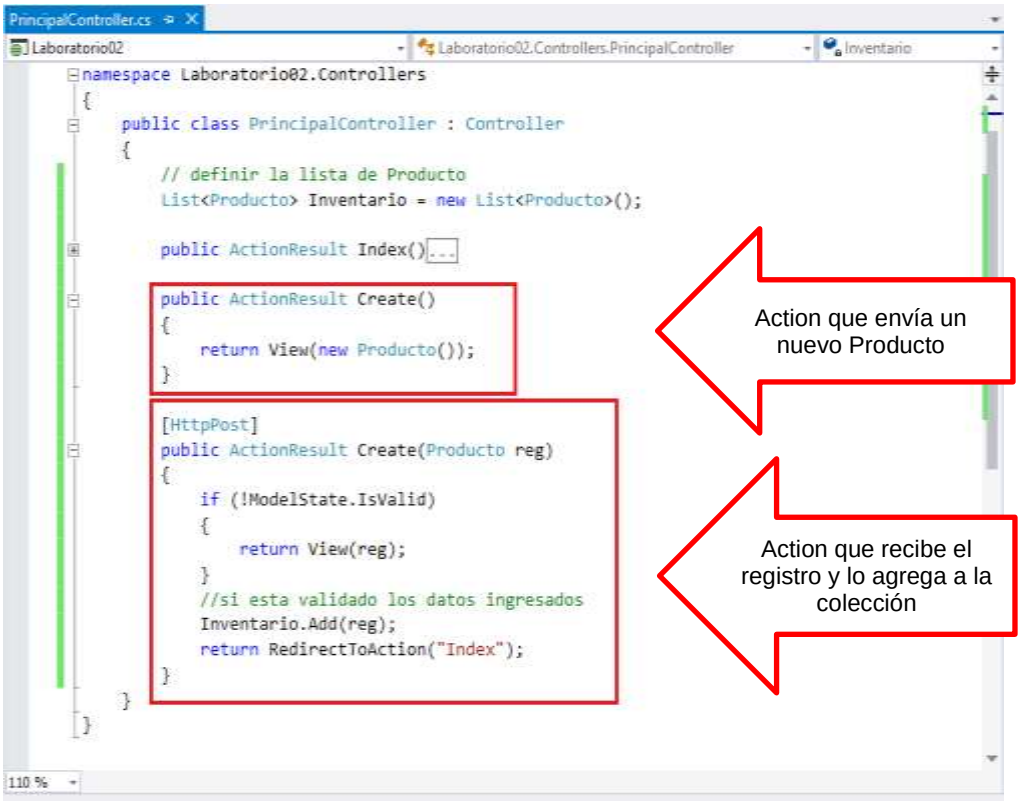


Generada la Vista, se visualiza tal como se muestra.

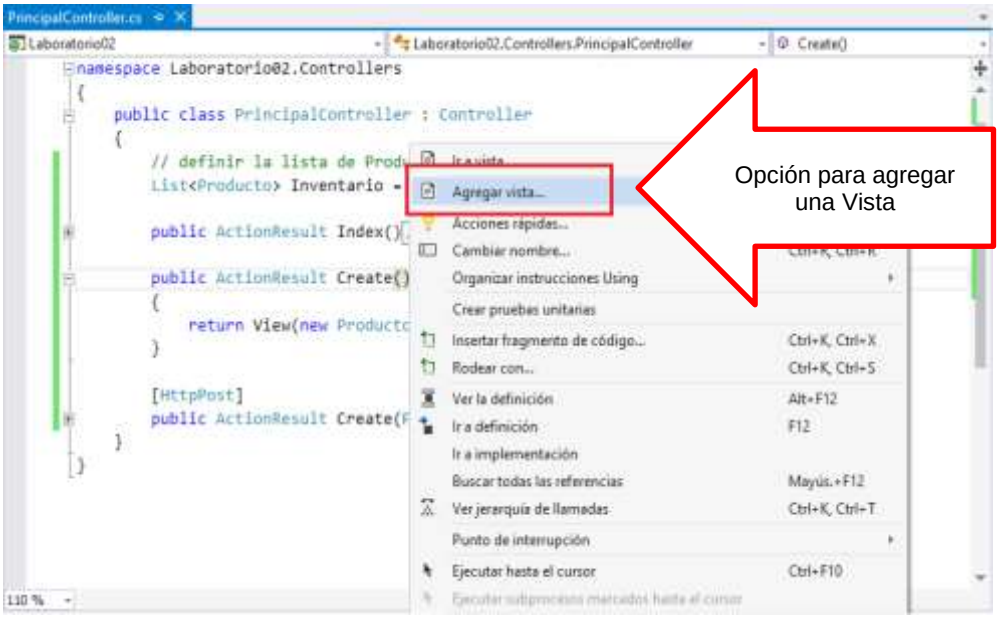
Ejecute la Vista Ctrl + F5, visualizando la ventana Index.



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.

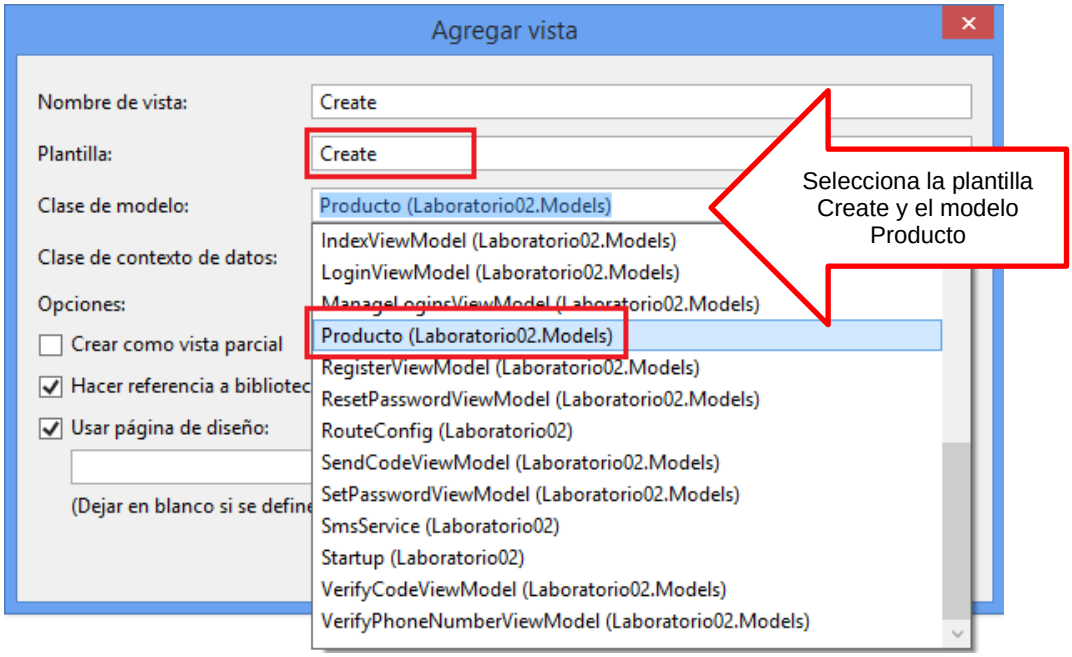


Definida la acción Create, agregar una vista: hacer click derecho a la acción y seleccionar la opción Agregar Vista, tal como se muestra

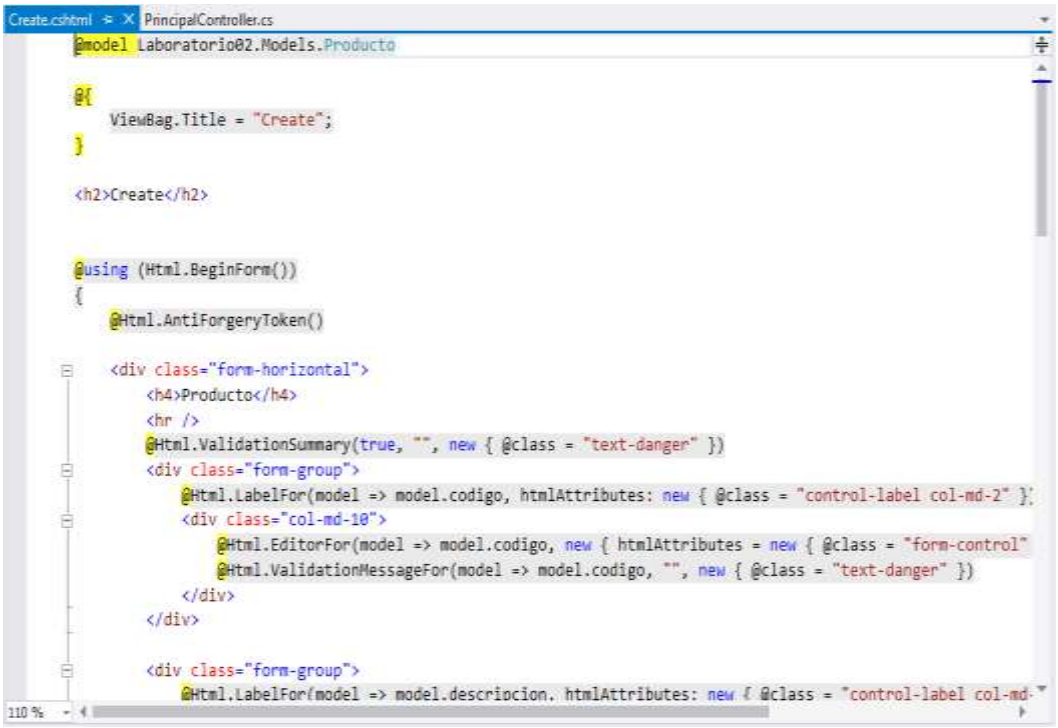


En la ventana Agregar Vista, aparece el nombre de la Vista: Create. No cambiar el nombre de la vista.

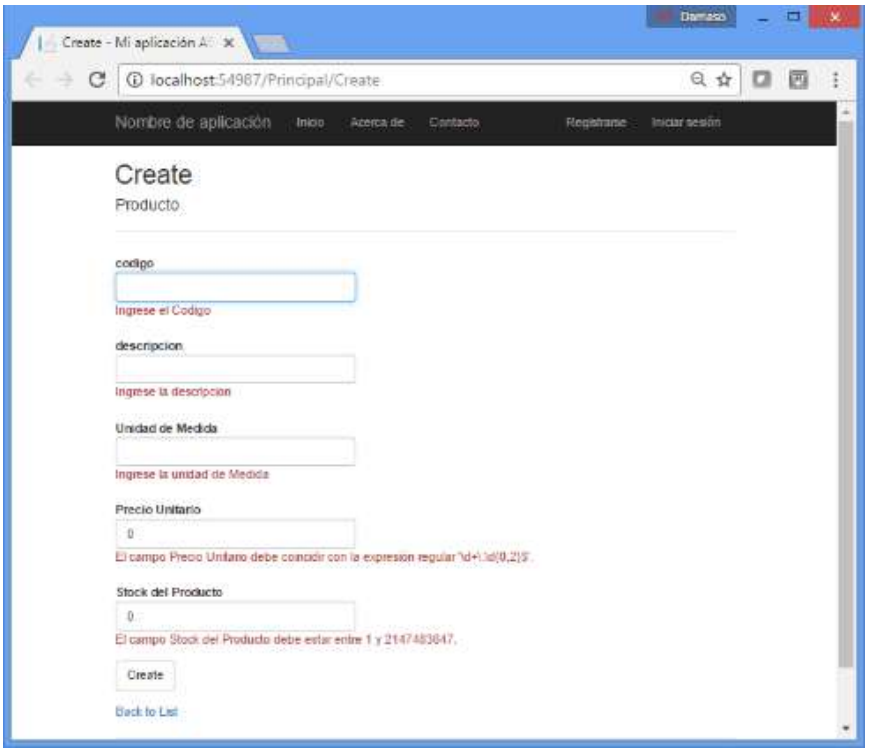
- Selecciona la plantilla: Create
- Selecciona la clase de modelo: Producto, el cual ingresamos los datos de un producto a la página.
- Presiona el botón Agregar



Página Create creada por la plantilla Create.



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.

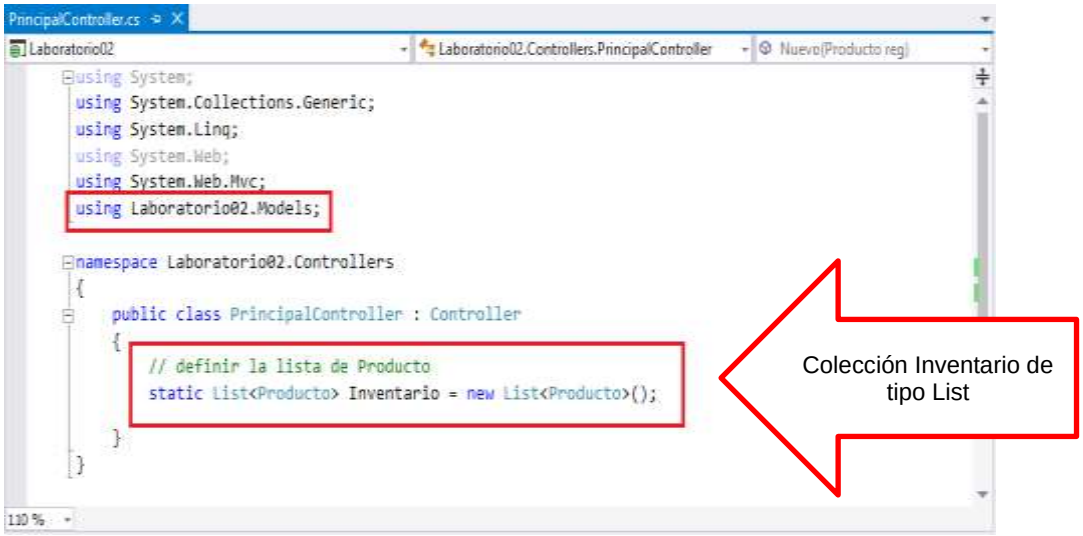


Laboratorio 2.2
Creando una aplicación ASP.NET MVC - RAZOR

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

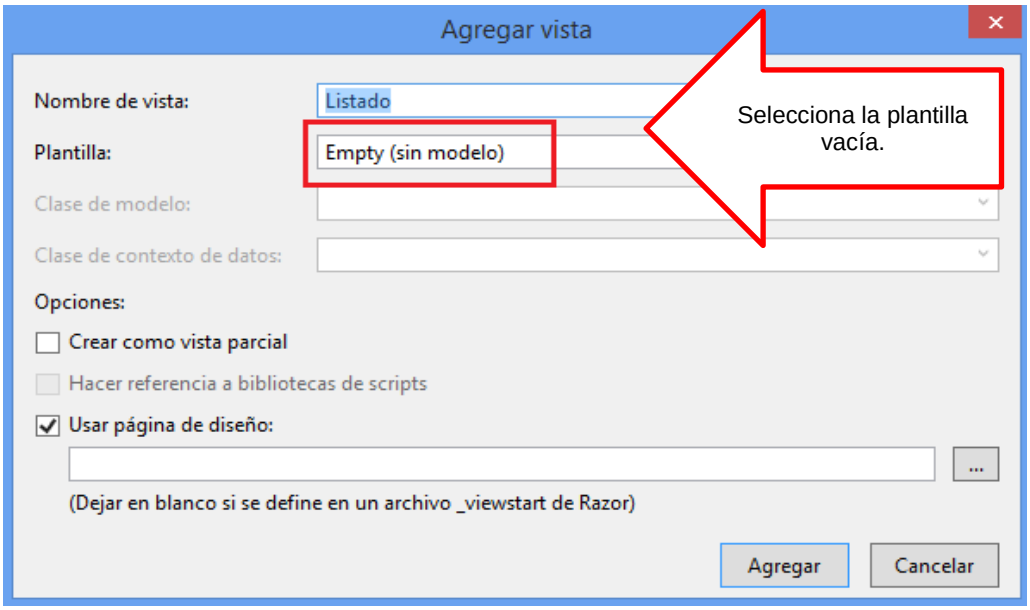
Trabajando con el Controlador

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.



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.

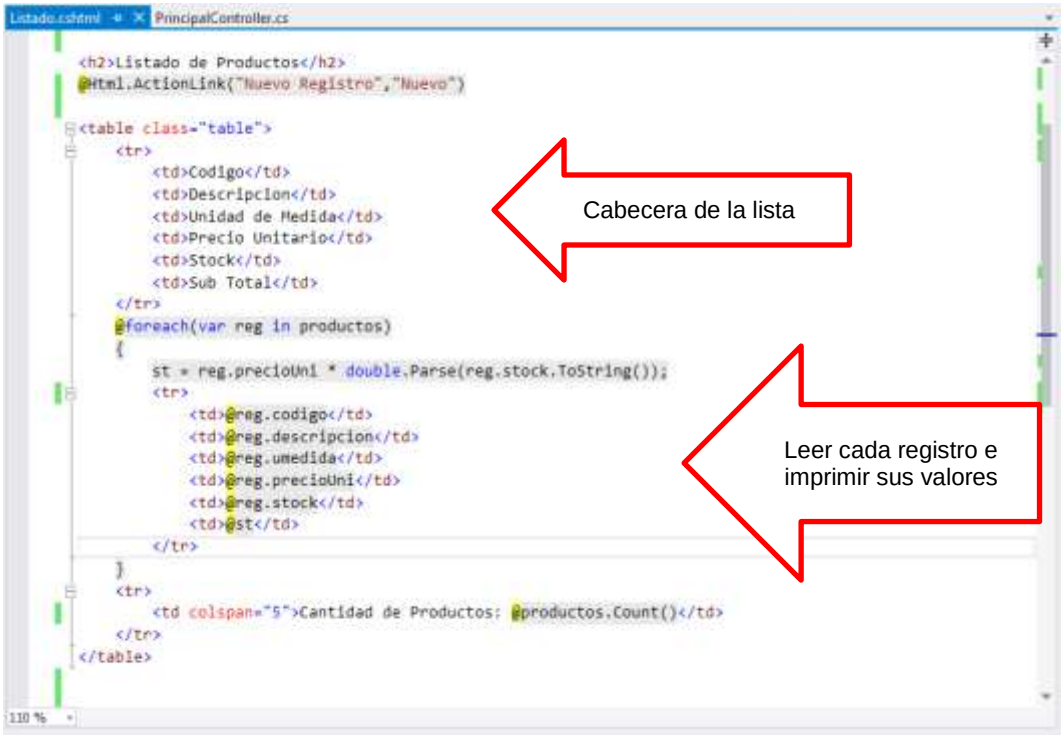
A continuación agregamos la Vista al Action Listado. Observe que la plantilla debe ser vacía (Empty). Presiona el botón Agregar



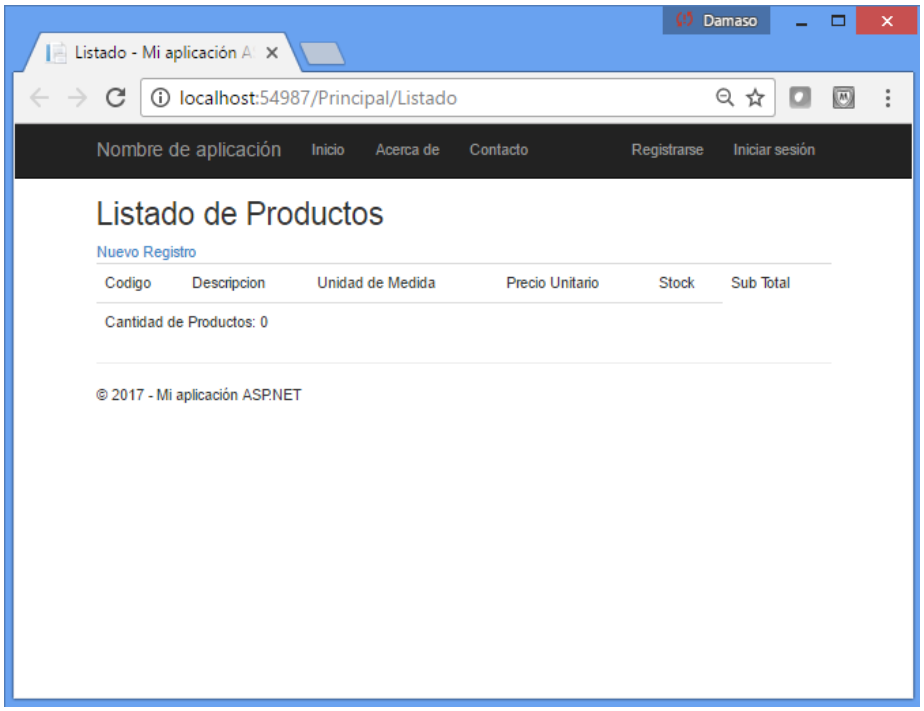
En la vista, primero importamos la carpeta Models, porque utilizamos la clase Producto en el proceso del Listado.

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

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.

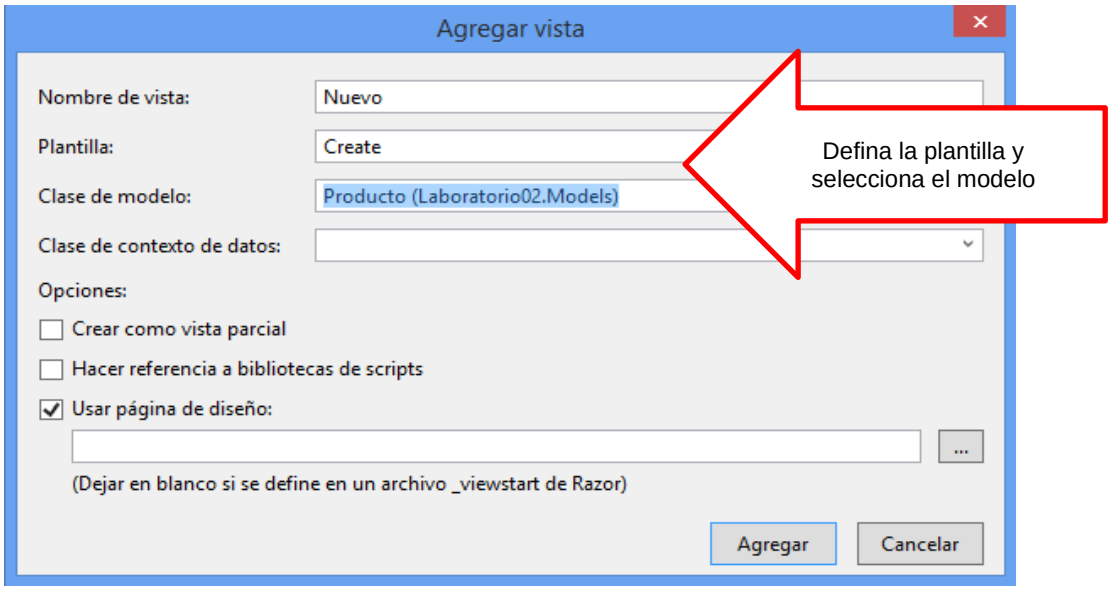


Guarde el proyecto y ejecuta la pagina Ctrl + F5, donde se visualiza la pagina en blanco

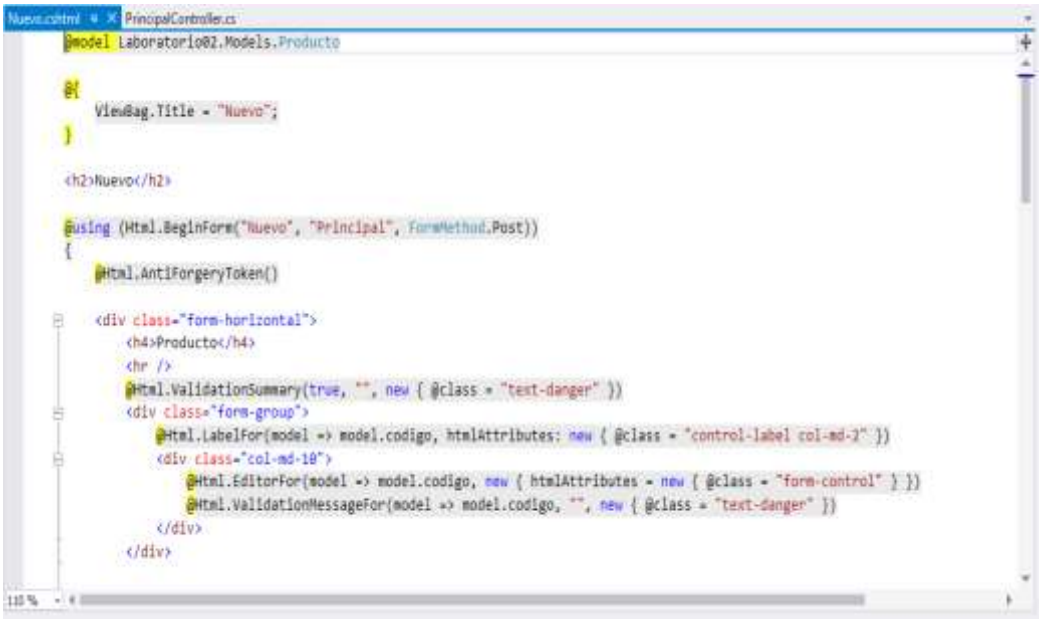


A continuación defina el ActionResult Nuevo, para enviar un Registro y recibir el registro para almacenar en la colección, tal como se muestra

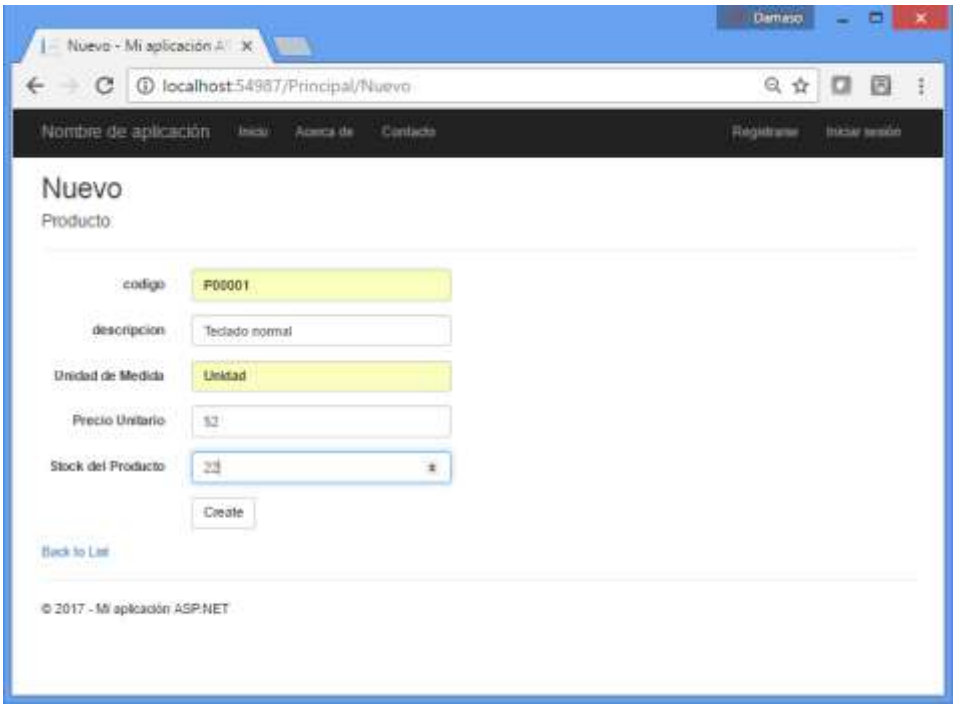
A continuación defina su vista: su plantilla es Create y el modelo es Producto



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



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://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://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: Introducción a ADO.NET (1 horas)

1. Arquitectura del ADO.NET
2. Manejando una cadena de conexión a un origen de datos utilizando ConfigurationManager

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. INTRODUCCION ADO.NET

Cuando desarrolle aplicaciones con ADO.NET contará con diferentes requisitos para trabajar con datos. En algunos casos, puede que simplemente desee realizar una consulta de datos en un formulario; en otros casos necesita actualizar los datos.

Independientemente de lo que haga con los datos, hay ciertos conceptos fundamentales que debe de comprender acerca del enfoque de los datos en ADO.NET, los cuales los trataremos en este primer capítulo del manual.

3.1 ARQUITECTURA Y PROVEEDORES DE DATOS

Tradicionalmente, el procesamiento de datos ha dependido principalmente de un modelo de dos niveles basado en una conexión. A medida que aumenta el uso que hace el procesamiento de datos de arquitecturas de varios niveles, los programadores están pasando a un enfoque sin conexión con el fin de proporcionar una mejor escalabilidad a sus aplicaciones.

Arquitectura de ADO.NET

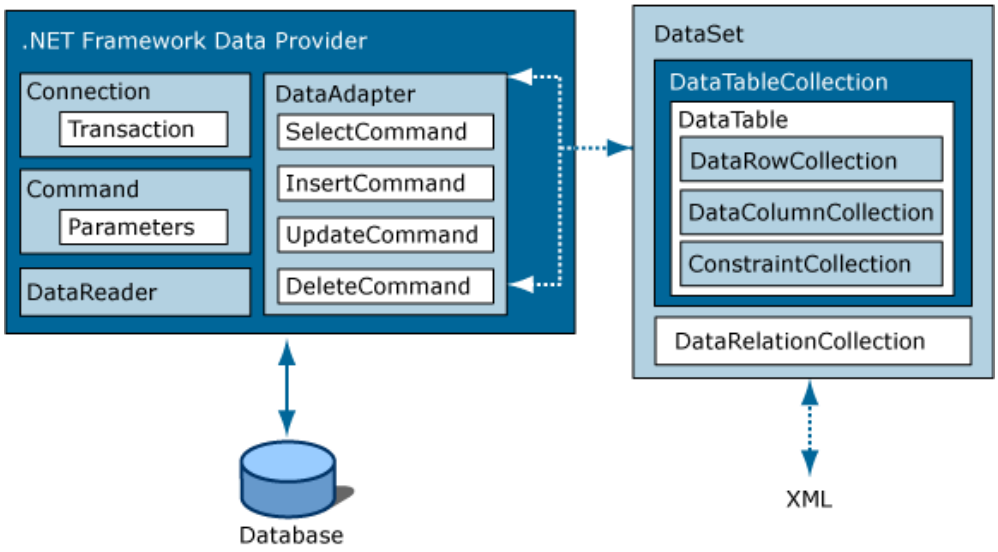


Figura 1: Arquitectura de ADO.NET
Referencia: hampprogramandoando.blogspot.com

Componentes de ADO.NET

Los dos componente principales de ADO.NET para el acceso a datos y su manipulación son los proveedores de datos .NET Fraework y DataSet.

LINQ to SQL

LINQ to SQL admite consultas en un modelo de objetos asignado a las estructuras de datos de una base de datos relacional sin utilizar un modelo conceptual intermedio. Cada tabla se representa mediante una clase distinta, acoplando de manera precisa el modelo de objetos al esquema de la base de datos relacional. LINQ to SQL traduce Language-integrated queries del modelo de objetos a Transact-SQL y lo envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados, LINQ to SQL los vuelve a traducir a objetos.

ADO.NET Entity Framework

ADO.NET Entity Framework está diseñado para permitir que los desarrolladores creen aplicaciones de acceso a los datos programando en un modelo de aplicación conceptual en lugar de programar directamente en un esquema de almacenamiento relacional. El

objetivo es reducir la cantidad de código y mantenimiento que se necesita para las aplicaciones orientadas a datos

WCF Data Services

Describe cómo se usa Servicios de datos de WCF para implementar servicios de datos en web o en una intranet. Los datos se estructuran como entidades y relaciones de acuerdo a las especificaciones de Entity Data Model. Los datos implementados en este modelo se pueden direccionar mediante el protocolo HTTP estándar

XML Y ADO.NET

ADO.NET aprovecha la eficacia de XML para proporcionar acceso a datos sin conexión. ADO.NET fue diseñado teniendo en cuenta las clases de XML incluidas en .NET Framework; ambos son componentes de una única arquitectura.

PROVEEDORES DE DATOS ADO.NET

Los proveedores de datos .NET Framework sirven para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente, se colocan en un DataSet con el fin de que el usuario pueda verlos cuando los necesite, se combinan con datos de varios orígenes o se utilizan de forma remota entre niveles. Los proveedores de datos .NET Framework son ligeros, de manera que crean un nivel mínimo entre el origen de datos y el código, con lo que aumenta el rendimiento sin sacrificar funcionalidad.

En la tabla siguiente se muestran los proveedores de datos .NET que se incluyen en el Framework .NET

Proveedor de Datos .NET	Descripción
.NET Framework Proveedor de datos para SQL Server	Proporciona acceso a datos para Microsoft SQL Server. Utiliza la librería System.Data.SqlClient.
Proveedor de datos .NET Framework para OLE DB	Para orígenes de datos que se exponen mediante OLE DB. Utiliza la librería System.Data.OleDb.
Proveedor de datos .NET Framework para ODBC	Para orígenes de datos que se exponen mediante ODBC. Utiliza la librería System.Data.Odbc.
Proveedor de datos .NET Framework para Oracle	Para orígenes de datos de Oracle. El proveedor de datos .NET Framework para Oracle es compatible con la versión 8.1.7 y posteriores del software de cliente de Oracle y utiliza la librería System.Data.OracleClient.
Proveedor EntityClient	Proporciona acceso a datos para las aplicaciones de Entity Data Model. Utiliza la librería System.Data.EntityClient.

Los cuatro objetos centrales que constityen un proveedor de datos de .NET Framework son:

Objeto	Descripción
Connection	Establece una conexión a una fuente de datos. La clase base para todos los objetos Connection es **DbConnection**.

Command	Ejecuta un comando en una fuente de datos. Expone Parameters y puede ejecutarse en el ámbito de un objeto Transaction desde Connection. La clase base para todos los objetos Command es **DbCommand**.
DataReader	Lee un flujo de datos de solo avance y solo lectura desde una fuente de datos. La clase base para todos los objetos DataReader es **DbDataReader**.
DataAdapter	Llena un DataSet y realiza las actualizaciones necesarias en una fuente de datos. La clase base para todos los objetos DataAdapter es **DbDataAdapter**.

Objeto	**Descripción**
Transaction	Incluye operaciones de actualización en las transacciones que se realizan en el origen de datos. ADO.NET es también compatible con las transacciones que usan clases en el espacio de nombres System.Transactions.
CommandBuilder	Un objeto auxiliar que genera automáticamente las propiedades de comando de un **DataAdapter** o que obtiene de un procedimiento almacenado información acerca de parámetros con la que puede rellenar la colección **Parameters** de un objeto **Command**.
Parameter	Define los parámetros de entrada y salida para los comandos y procedimientos almacenados
ConnectionStringBuilder	Un objeto auxiliar que proporciona un modo sencillo de crear y administrar el contenido de las cadenas de conexión utilizadas por los objetos **Connection**.
Exception	Se devuelve cuando se detecta un error en el origen de dato
ClientPermission	Se proporciona para los atributos de seguridad de acceso del código de los proveedores de datos

DATASET EN ADO.NET

El objeto DataSet es esencial para la compatibilidad con escenarios de datos distribuidos desconectados con ADO.NET.

El objeto DataSet es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente independientemente del origen de datos. Se puede utilizar con muchos y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación.

El DataSet representa un conjunto completo de datos que incluye tablas relacionadas y restricciones, así como relaciones entre las tablas. En la siguiente ilustración se muestra el modelo de objetos DataSet.

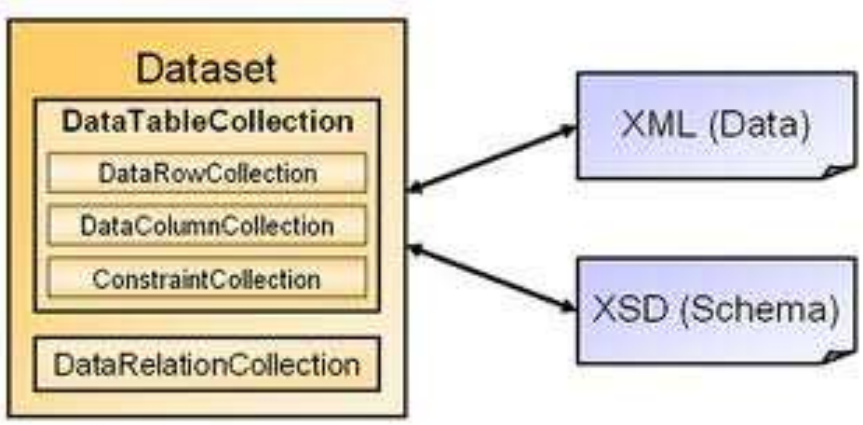


Figura 2: DataSet
Referencia: edn.embarcadero.com

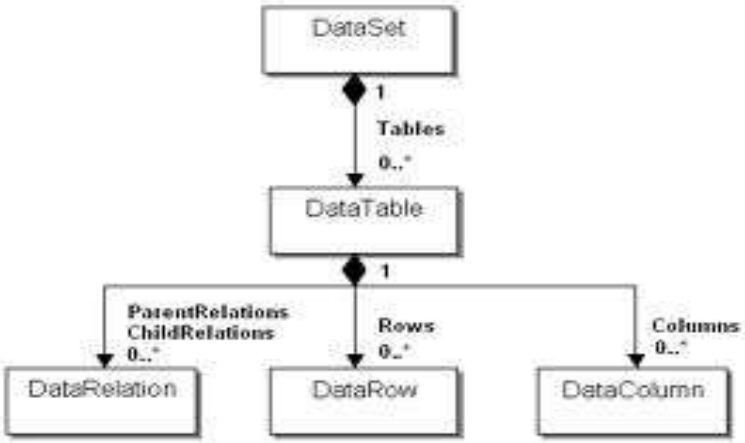


Figura 3: DataSet y Componenets
Referencia: www.c-sharpcorner.com

3.2 MANEJANDO UNA CADENA DE CONEXIÓN A UN ORIGEN DE DATOS

En ADO.NET se utiliza un objeto Connection para conectar con un determinado origen de datos mediante una cadena de conexión en la que se proporciona la información de autenticación necesaria. El objeto Connection utilizado depende del tipo de origen de datos.

Cada proveedor de datos .NET Framework incluye un objeto Connection:

- Proveedor de datos para OLE DB incluye un objeto OleDbConnection,
- Proveedor de datos para SQL Server incluye un objeto SqlConnection,
- Proveedor de datos para ODBC incluye un objeto OdbcConnection y
- Proveedor de datos para Oracle incluye un objeto OracleConnection.

Conectar a SQL Server mediante ADO.NET

Para conectarse a Microsoft SQL Server 7.0 o posterior, utilice el objeto SqlConnection del proveedor de datos .NET Framework para SQL Server. El proveedor de datos .NET

Framework para SQL Server admite un formato de cadena de conexión que es similar al de OLE DB (ADO).

En el ejemplo siguiente se muestra la forma de crear un abrir una conexión a un origen de datos en SQL Server

```
SqlConnection cn = new SqlConnection(“ Data Source=(local); Initial Catalog=NorthWind;
 user id=sa; pwd=sql”);

cn.Open()
```

Donde:

Data Source: Origen de datos
Initial Catalog=Nombre de la base de datos
User id=usuario
Pwd=clave

Cerrar una Conexión

Debe cerrar siempre el objeto Connection cuando deje de usarlo. Esta operación se puede realizar mediante los métodos Close o Dispose del objeto Connection.

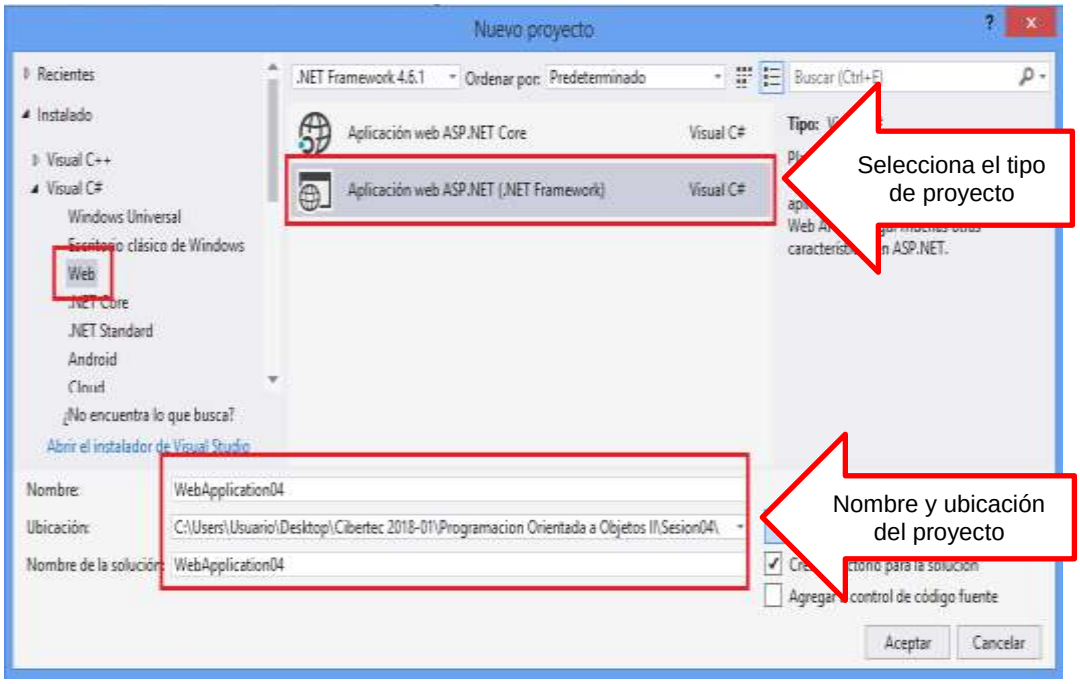
Las conexiones no se liberan automáticamente cuando el objeto Connection queda fuera de ámbito o es reclamado por el garbageCollector.

Laboratorio 01
Consulta de Datos

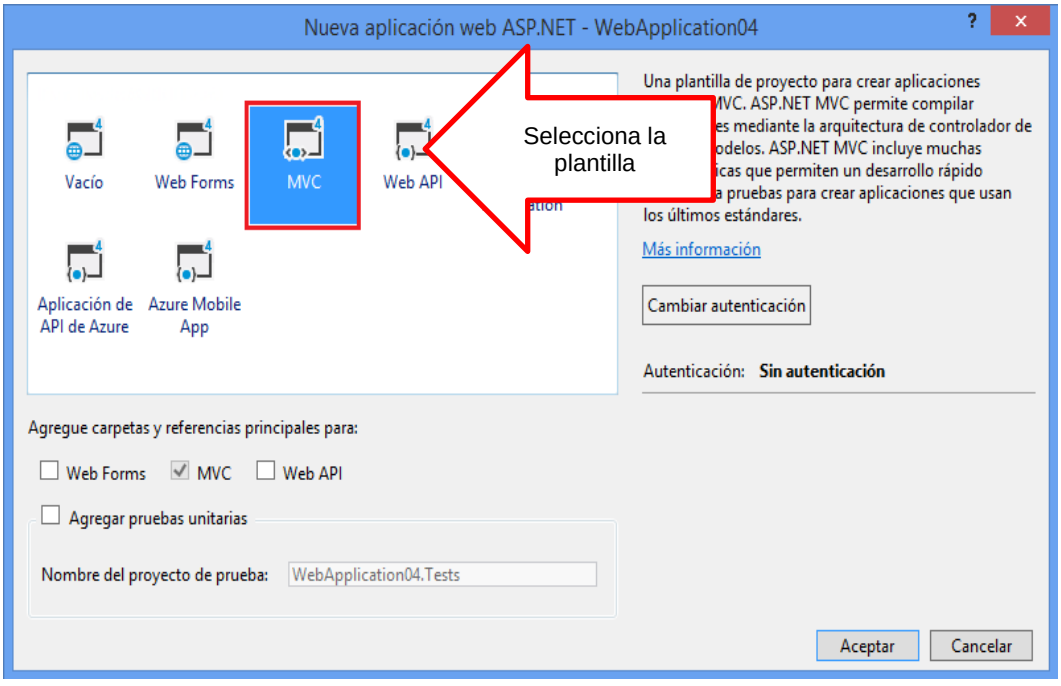
Implemente un proyecto ASP.NET MVC, diseñe una página que permita listar los registros de la tabla tb_clientes de la base de datos Negocios2018

Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR

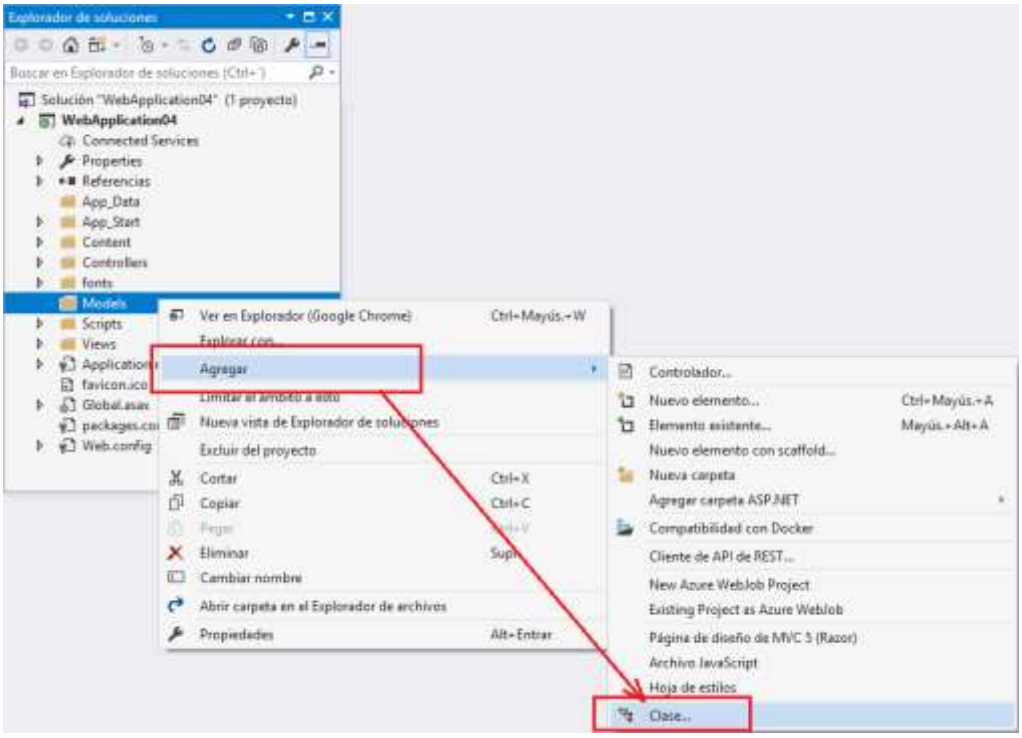


A continuación, seleccionar la plantilla del proyecto MVC.

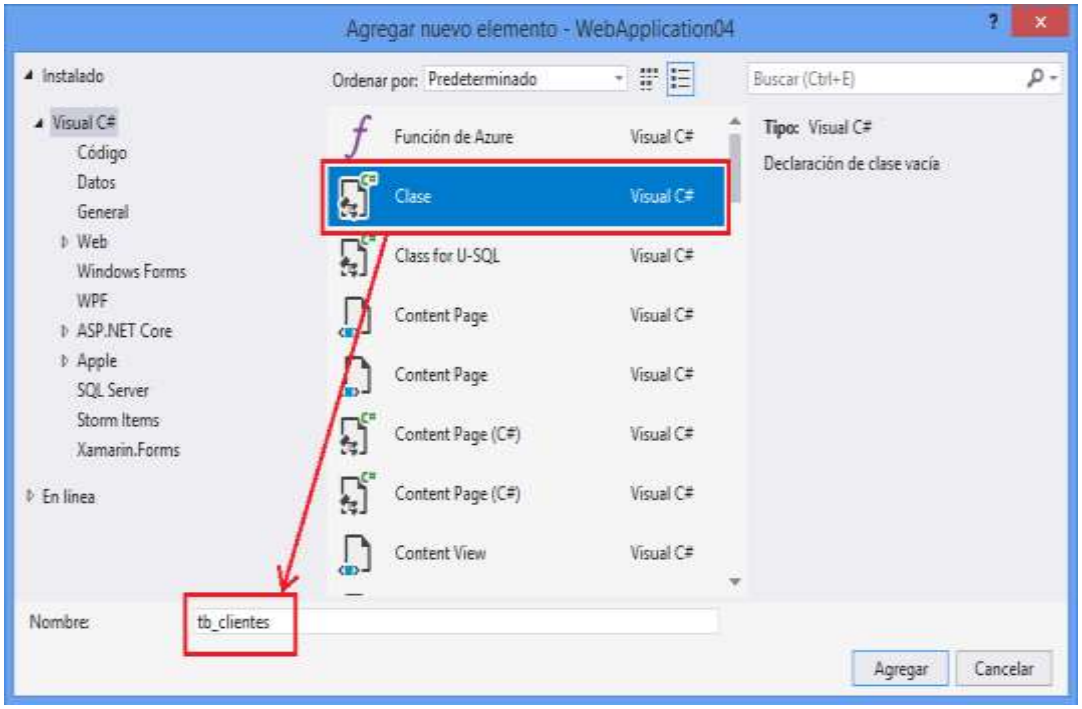


Creando la Clase del Modelo

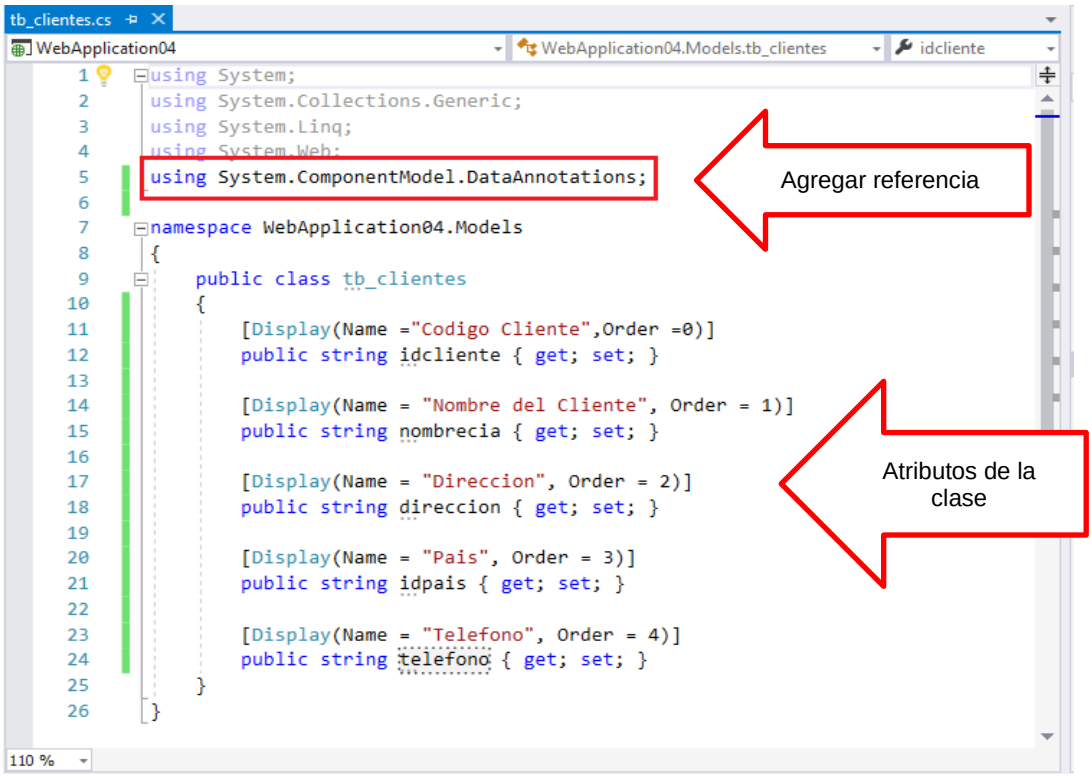
Para efectuar y realizar la consulta de los datos de la tabla tb_clientes, agregamos, en la carpeta Models, una clase, tal como se muestra.



Asigne el nombre de la clase, la cual se llamará tb_clientes, tal como se muestra. Presiona el botón AGREGAR



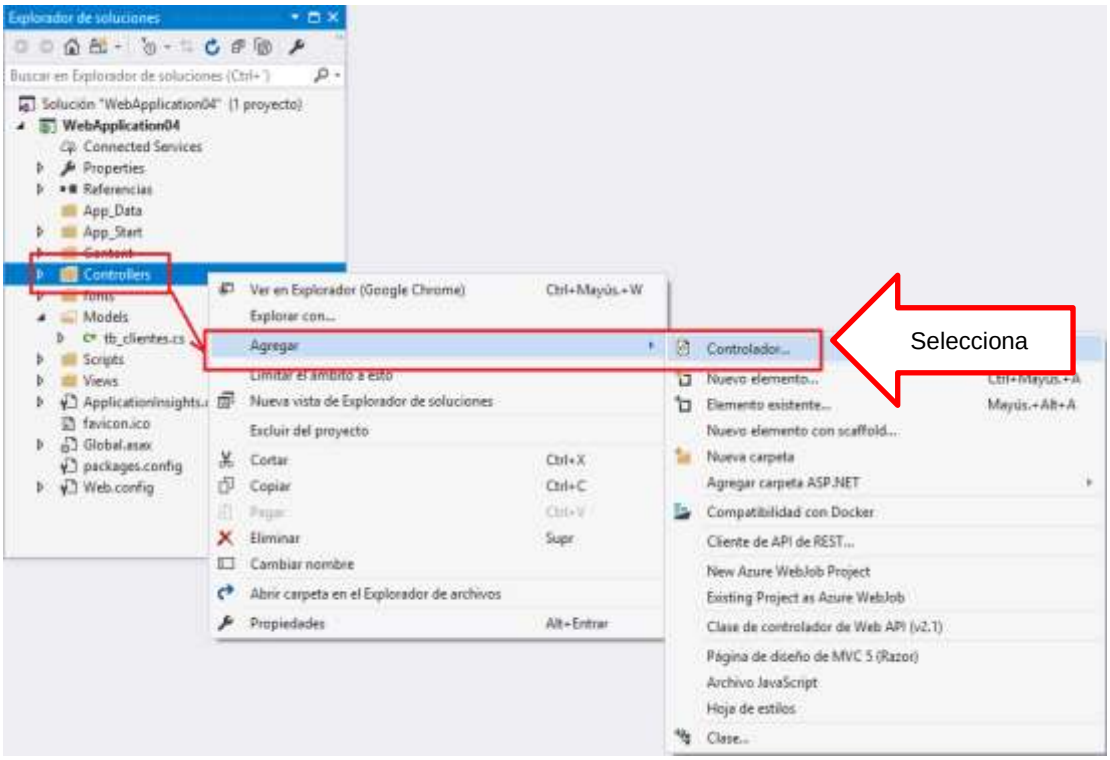
Para iniciar, agregamos la referencia DataAnnotations, luego defina los atributos de la clase, visualizando el texto y orden de cada uno de ellos [Display (Name=”texto”, Order=”n”)], tal como se muestra.



Terminado con el proceso de la definición de la clase, el siguiente paso es el desarrollo del controlador y su vista

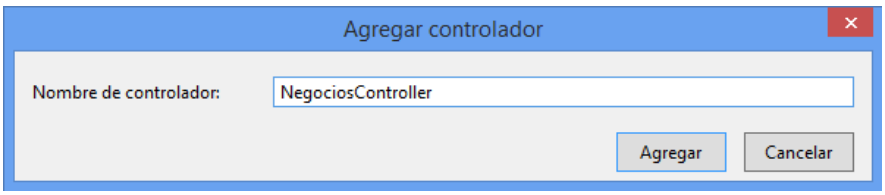
Creando un Controlador

A continuación, creamos un controlador: Desde la carpeta Controllers, selecciona la opción Agregar, Controlador…, tal como se muestra.

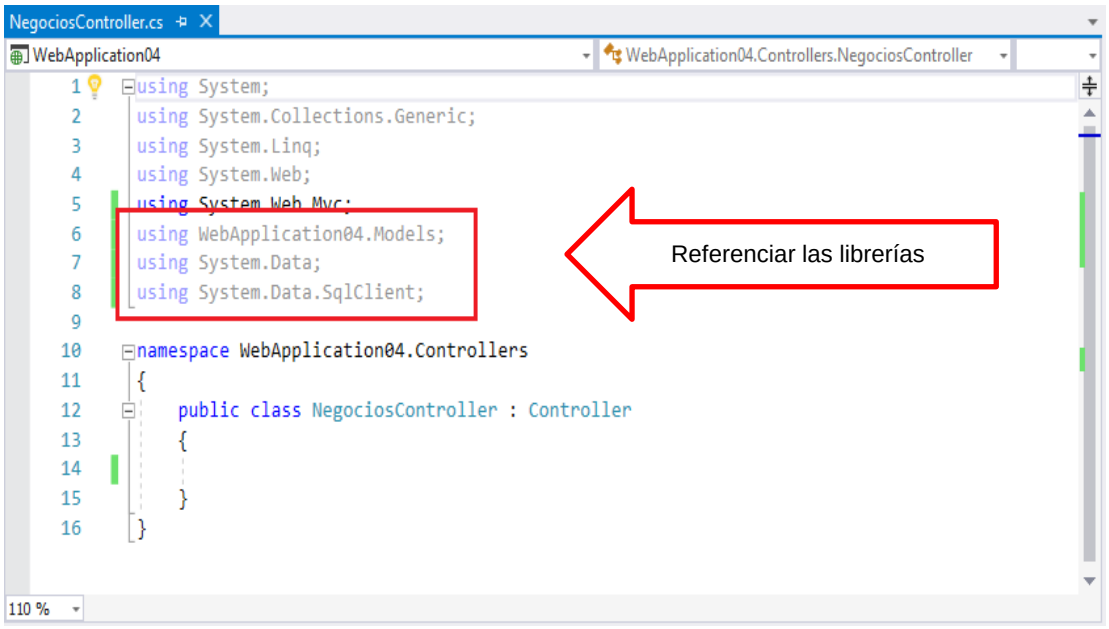


Selecciona el scaffold, Controlador de MVC5: en blanco, presiona el botón AGREGAR

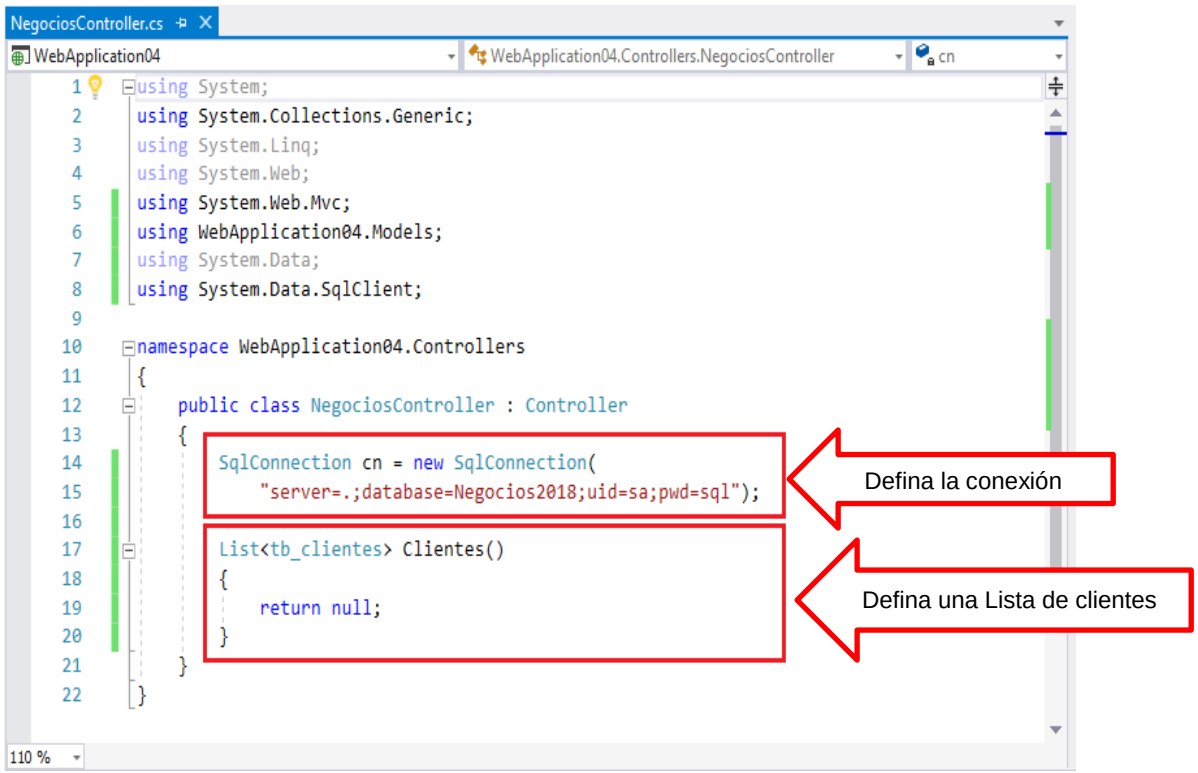
Asigne el nombre del controlador: NegociosController, tal como se muestra



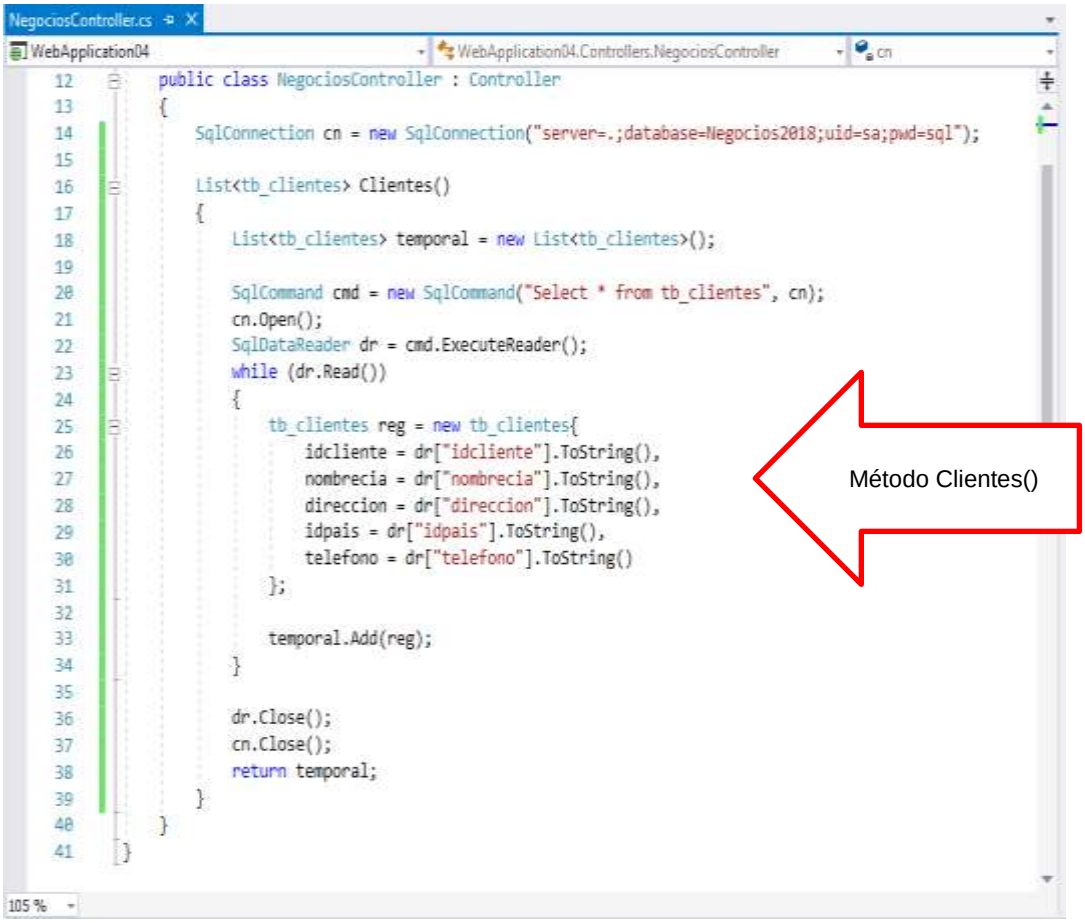
Referenciar la carpeta **Models**, y la librería **Data**.**SqlClient** tal como se muestra.



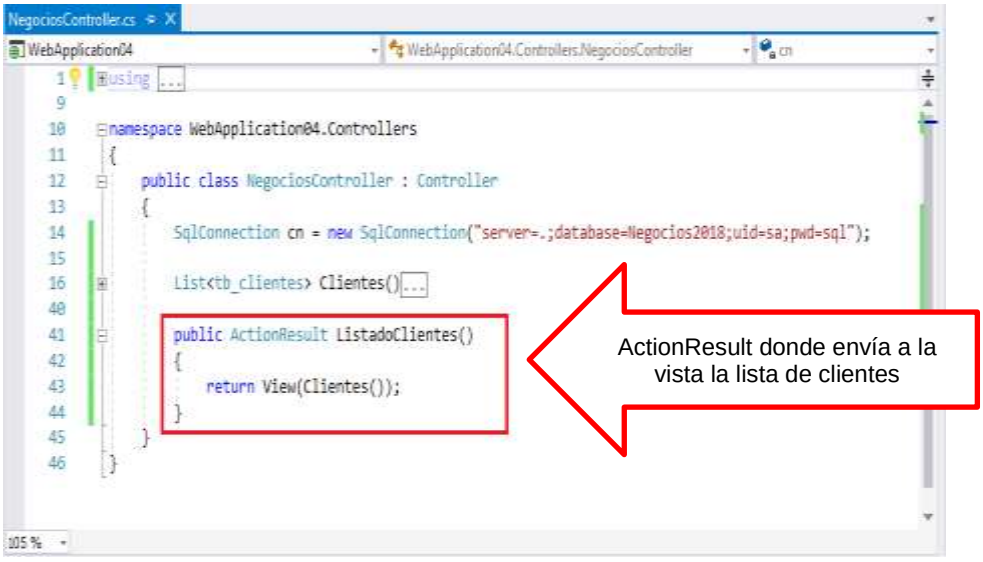
Dentro del controlador, defina la conexión a la base de datos Negocios2018 instanciando el **SqlConnection**; luego defina una lista donde retornará los registros de tb_clientes



A continuación codifique la lista llamada Clientes:
1. Ejecuta la sentencia SQL a través del SqlCommand, almacenando los resultados en el SqlDataReader.
2. Para guardar los resultados en el temporal se realiza a través de un bucle (mientras se pueda leer: dr.Read()), vamos almacenando cada registro en el temporal.
3. Cerrar los objetos y enviar el objeto lista llamada temporal.

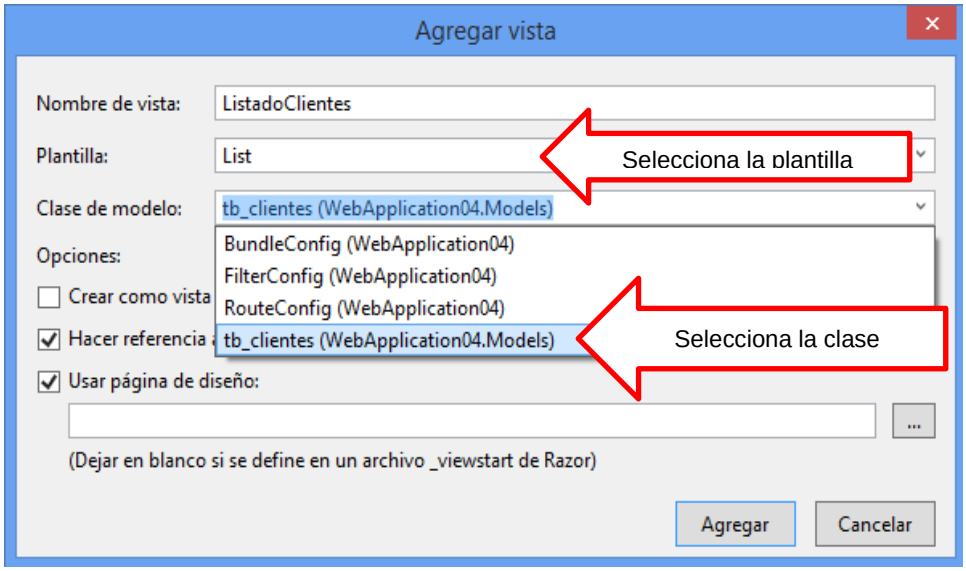


Defina el ActionResult ListadoClientes(), el cual enviará a la Vista el resultado del método Clientes(), tal como se muestra.



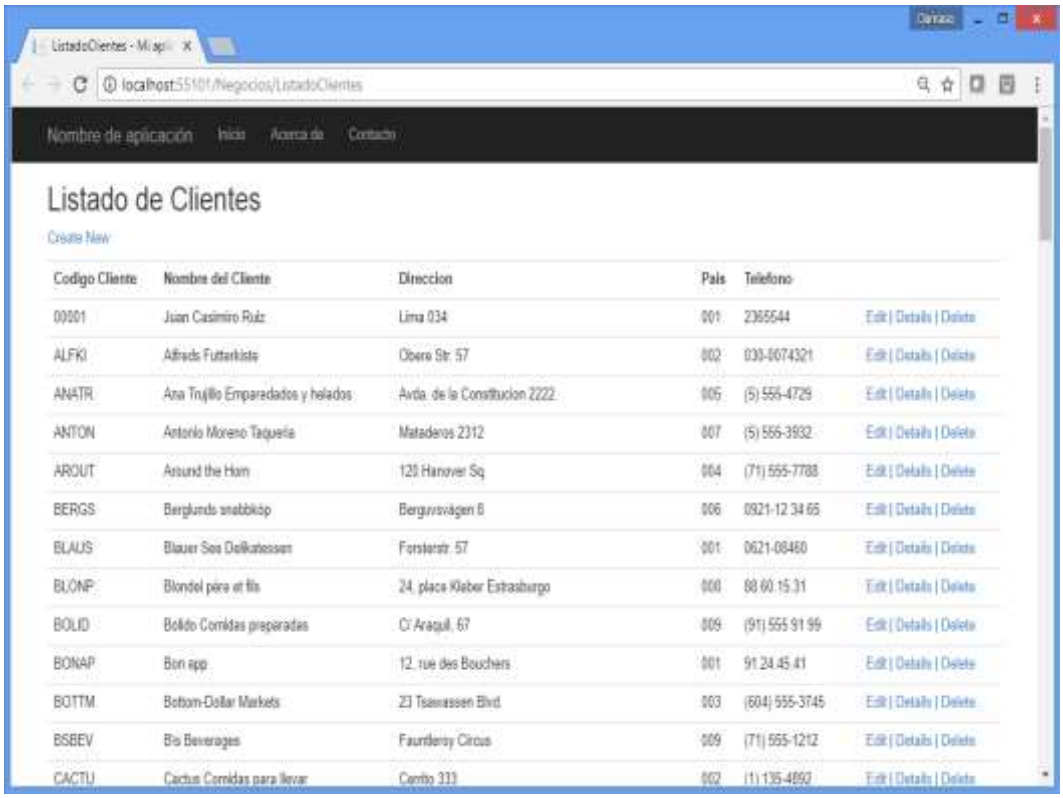
Agregando la Vista.

En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **List**; y la **clase de modelo** la cual es **tb_clientes**, tal como se muestra



Vista generada por la plantilla **List**, tal como se muestra

Ejecutamos la Vista, visualizando los registros de la tabla tb_clientes



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 rellenar 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
  - 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/

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 4: Recuperación de datos (7 horas)

4.1 La clase DataReader: realizando consulta de datos.
4.2 Consulta de datos utilizando parámetros con DataReader.
4.3 Trabajando con Vistas

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. RECUPERACIÓN DE DATOS

La función principal de cualquier aplicación que trabaje con una fuente de datos es conectarse a dicha fuente de datos y recuperar los datos que se almacenan.

Los proveedores de .NET Framework para ADO.NET sirven como puente entre una aplicación y un origen de datos, permitiendo ejecutar instrucciones SQL para recuperar datos mediante el DataAdapter o el DataReader.

4.1 REALIZANDO UNA CONSULTA CON DATAADAPTER

La clase DataAdapter se encarga de las operaciones entre la capa de datos y la capa intermedia donde los datos son transferidos. Se puede decir que sirve como puesnte entre un objeto DataSet y un origen de datos asociados para recuperar y guardar datos.

Básicamente, permite rellenar (Fill) el objeto DataSet para que sus datos coincidan con los del origen de datos y permite actualizar (Update) el origen de datos para que sus datos coincidan con los del DataSet.

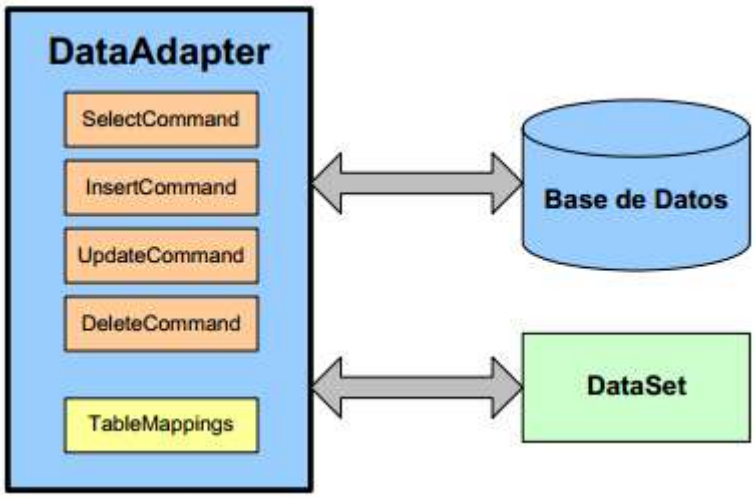


Figura 1: Diagrama del objeto DataAdapter
Referencia: http://ehu.es/mrodriguez/archivos/csharppdf/ADONET/ADONET.pdf

Los constructores de la clase son:

- DataAdapter (Command selectCommand). Utiliza un comando de selección como parámetro.
- DataAdapter (string selectText, string selectConnectionString). Se utiliza una sentencia SQL de selección y una cadena de conexión como parámetros
- DataAdapter (string selectText, Connection selectConnection). Se utiliza los parámetros sentencia SQL de selección y un objeto de tipo conexión.

En la interface IDataAdapter se declaran los siguientes métodos (toda clase que implemente esta interface está obligata a implementarlos)

- Fill (DataSet). Rellena un objeto de la clase DataSet
- FillSchema (Dataset, Tipo de schema). Rellena un esquema con un DataSet indicando el tipo de esquema a rellenar
- Update (DataSet). Actualiza el DataSet correspondiente
- GetFillParameters(). Recoge el conjunto de parámetros cuando se ejecuta una consulta de selección.

Cada proveedor de datos de .NET FrameWork cuenta con un objeto DataAdapter:

Proveedor	Descripcion
OledbDataAdapter	Proveedor de datos para OLEDB
SqlDataAdapter	Proveedor de datos para SQL Server
OdbcDataAdapter	Proveedor de datos para ODBC
OracleDataAdapter	Proveedor de datos para Oracle

REALIZANDO UNA CONSULTA CON PARÀMETROS

Cuando el DataAdapter ejecuta instrucciones de consulta con parámetros, se deben definen qué parámetros de entrada y de salida se deben crear. Para crear un parámetro en un DataAdapter, se utiliza el método: Parameters.Add() o Parameteres.AddWithValue().

El método **Parameters.Add()** se debe especificar el nombre de columna, tipo de datos y tamaño, asignando el valor del parámetro definido. El método Add de la colección Parameters toma el nombre del parámetro, el tipo de datos, el tamaño (si corresponde al tipo) y el nombre de la propiedad SourceColumn de DataTable

El método **Parameters.AddWithValue()** se debe especificar el nombre del parámetro y su valor.

4.2 TRABAJANDO CON DATAREADER

La recuperación de datos mediante DataReader implica crear una instancia del objeto Command y de un DataReader a continuación, para lo cual se llama a **Command**.**ExecuteReader** a fin de recuperar filas de un origen de datos.

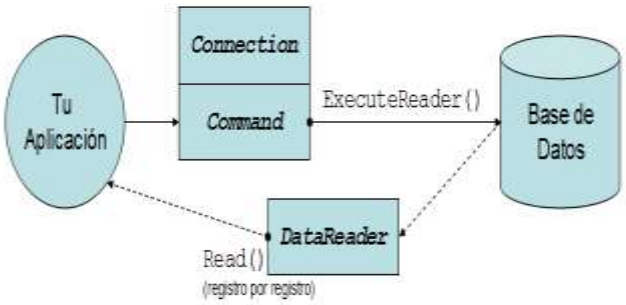


Figura 1: Diagrama del objeto DataReader
Referencia: http://www.dotnetero.com/2006/08/adonet-para-novatos.html

Puede utilizar el método **Read** del objeto DataReader para obtener una fila a partir de los resultados de una consulta.Para tener acceso a cada columna de la fila devuelta, puede pasar a DataReader el nombre o referencia numérica de la columna en cuestión.Sin embargo, el mejor rendimiento se logra con los métodos que ofrece DataReader y que permiten tener acceso a los valores de las columnas en sus tipos de datos nativos (GetDateTime, GetDouble, GetGuid, GetInt32, etc.).

Para obtener una lista de métodos de descriptor de acceso con tipo para **DataReaders** de proveedores de datos:

Proveedor	Descripcion
OledbDataReader	Proveedor de datos para OLEDB
SqlDataReader	Proveedor de datos para SQL Server
OdbcDataReader	Proveedor de datos para ODBC
OracleDataReader	Proveedor de datos para Oracle

Cerrar el DataReader

Siempre debe llamar al método **Close** cuando haya terminado de utilizar el objeto DataReader.

Si Command contiene parámetros de salida o valores devueltos, éstos no estarán disponibles hasta que se cierre el **DataReader**.

Tenga en cuenta que mientras está abierto un **DataReader**, ese **DataReader** utiliza de forma exclusiva el objeto **Connection**. No se podrá ejecutar ningún comando para el objeto **Connection** hasta que se cierre el **DataReader** original, incluida la creación de otro **DataReader**.

Recuperar varios conjuntos de resultados con NextResult

En el caso en que se devuelvan varios resultados, el DataReader proporciona el método **NextResult** para recorrer los conjuntos de resultados en orden.

Obtener información del esquema a partir del DataReader

Mientras hay abierto un **DataReader**, puede utilizar el método **GetSchemaTable** para recuperar información del esquema acerca del conjunto actual de resultados. **GetSchemaTable** devuelve un objeto **DataTable** rellenado con filas y columnas que contienen la información del esquema del conjunto actual de resultados.

DataTable contiene una fila por cada una de las columnas del conjunto de resultados. Cada columna de una fila de la tabla de esquema está asociada a una propiedad de la columna que se devuelve en el conjunto de resultados. **ColumnName** es el nombre de la propiedad y el valor de la columna es el de la propiedad. En el ejemplo de código siguiente se muestra la información del esquema de **DataReader**.

4.3 TRABAJANDO CON VISTAS Y CONSULTA DE DATOS

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.

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 programacion 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 paginas web Razor puede ser descrito como paginas HTML con dos clases de contenido: contenido HTML y código Razor.
- Las paginas 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 csthml (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"})
```

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

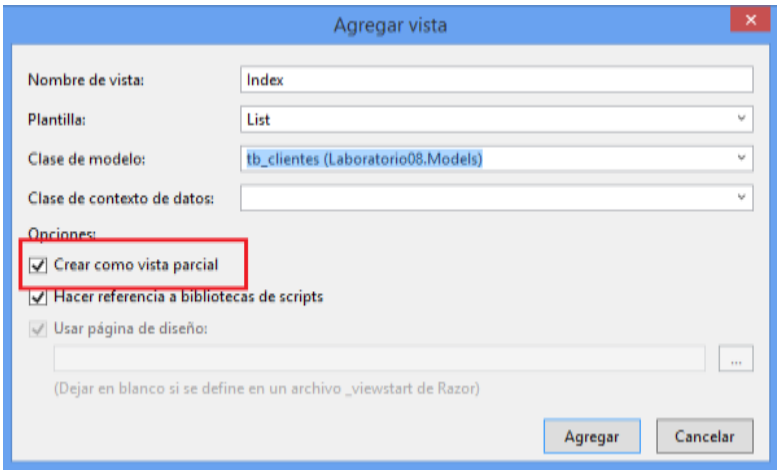
4.2 Vistas parciales

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 crea 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 parcila, 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.

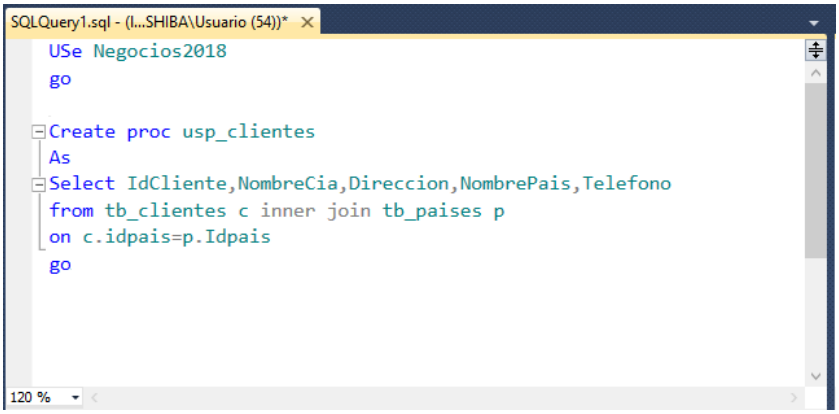
Por el contrario, una aplicación con una estructura más convencional utiliza el enlace de datos únicamente en controles de lista y texto, y responde a la entrada del usuario mediante el control de eventos expuestos por los controles. Los controladores de eventos están estrechamente acoplados a los controles 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.

Laboratorio 4.1
Consulta de Datos - publicando la conexión

Implemente un proyecto ASP.NET MVC, diseñe una página que permita listar los registros de la tabla tb_clientes de la base de datos Negocios2018, ejecutando el procedimiento almacenado usp_clientes

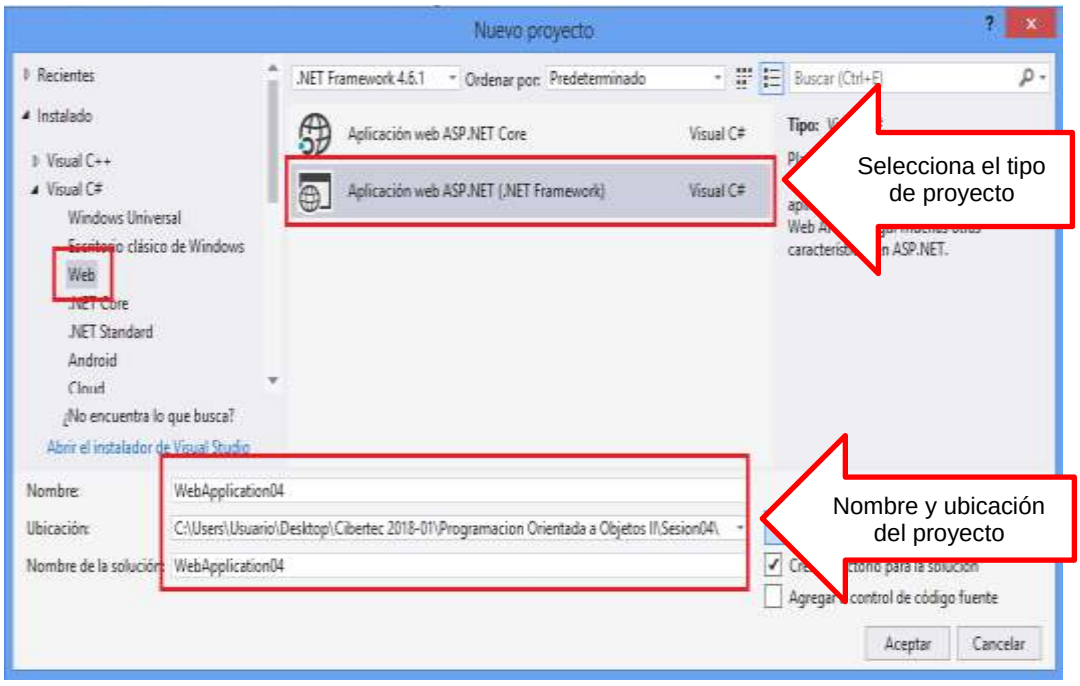
Creando el procedimiento almacenado

En el administrador del SQL Server, defina el procedimiento almacenado usp_clientes, tal como se muestra.

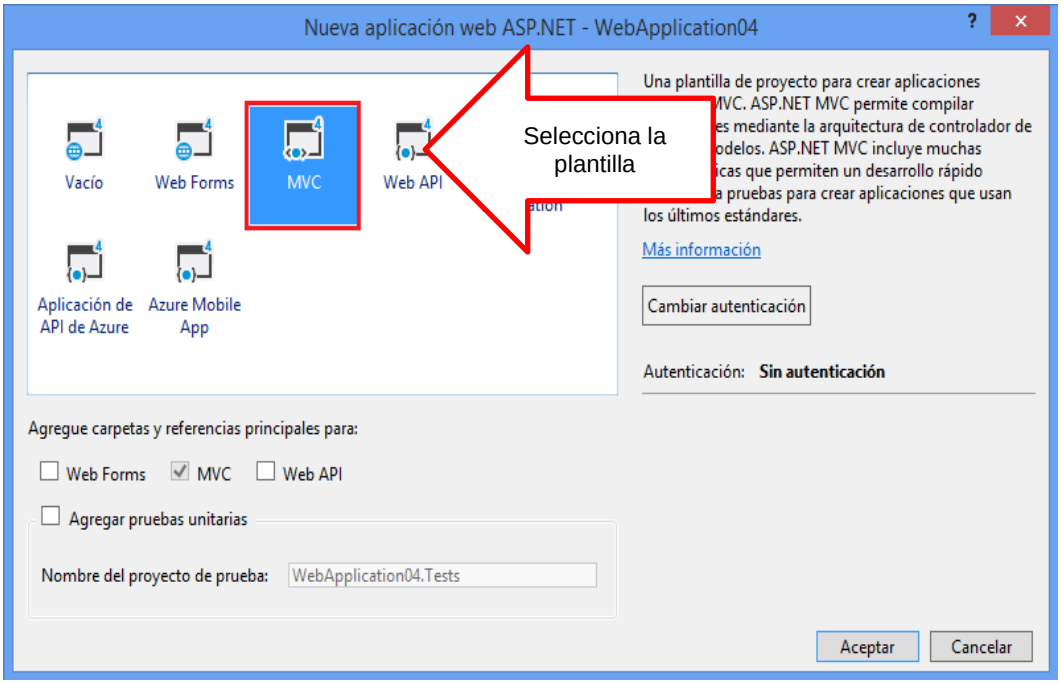


Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR

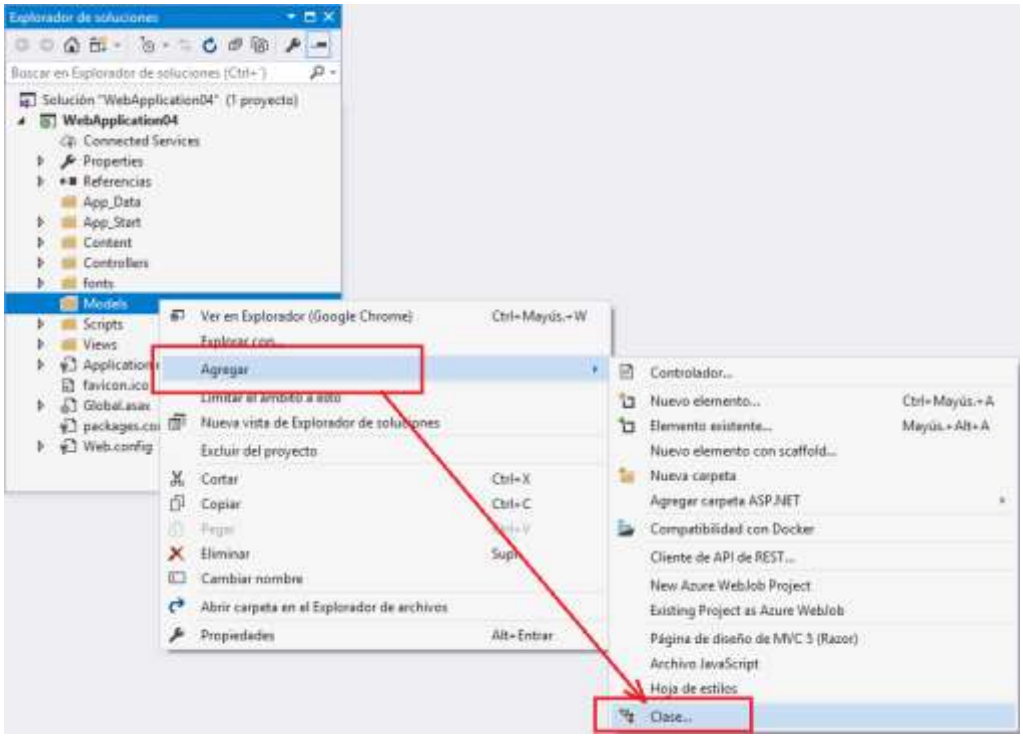


A continuación, seleccionar la plantilla del proyecto MVC.

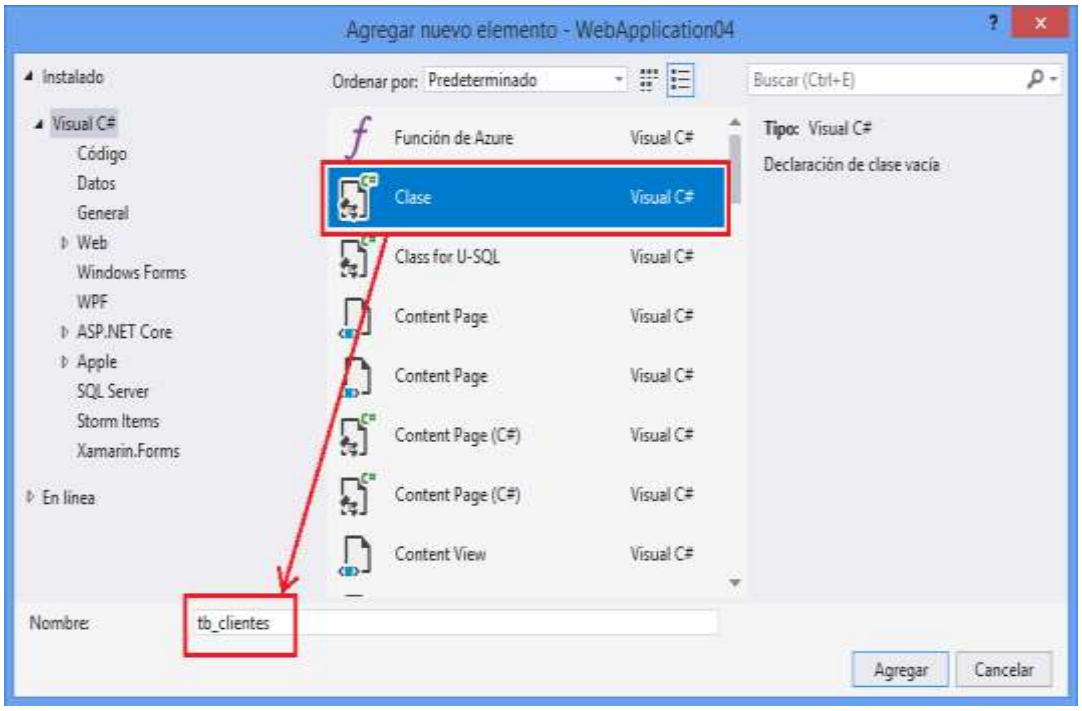


Creando la Clase del Modelo

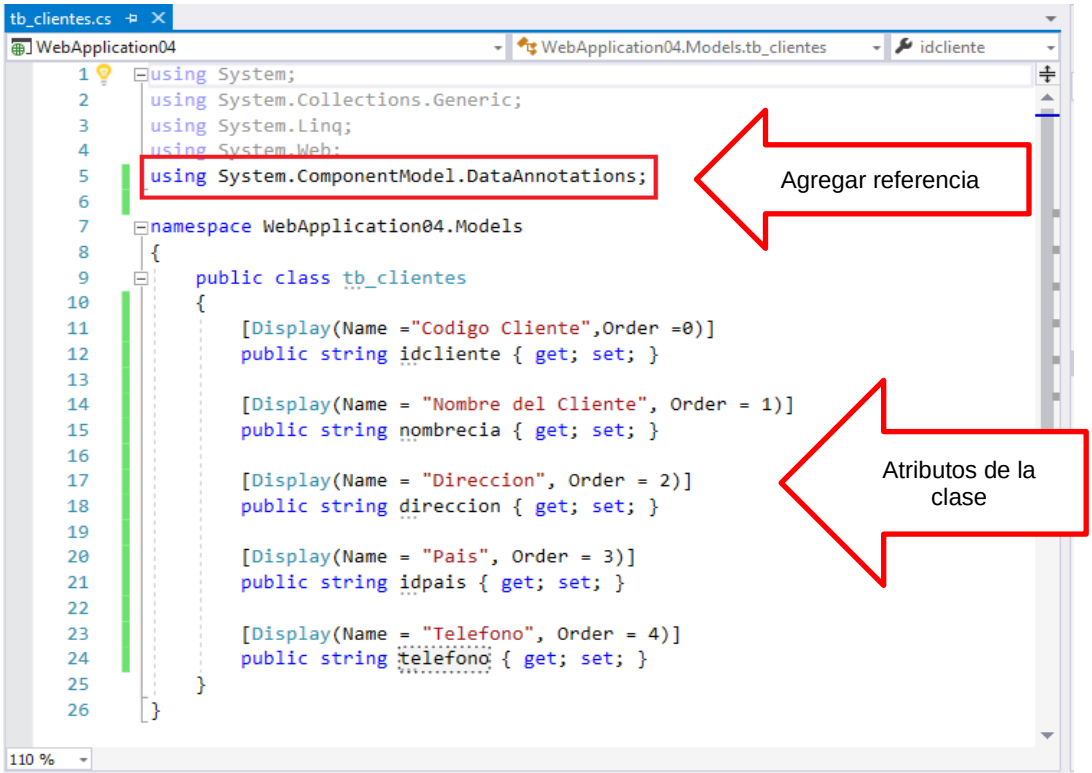
Para efectuar y realizar la consulta de los datos de la tabla tb_clientes, agregamos, en la carpeta Models, una clase, tal como se muestra.



Asigne el nombre de la clase, la cual se llamará tb_clientes, tal como se muestra. Presiona el botón AGREGAR

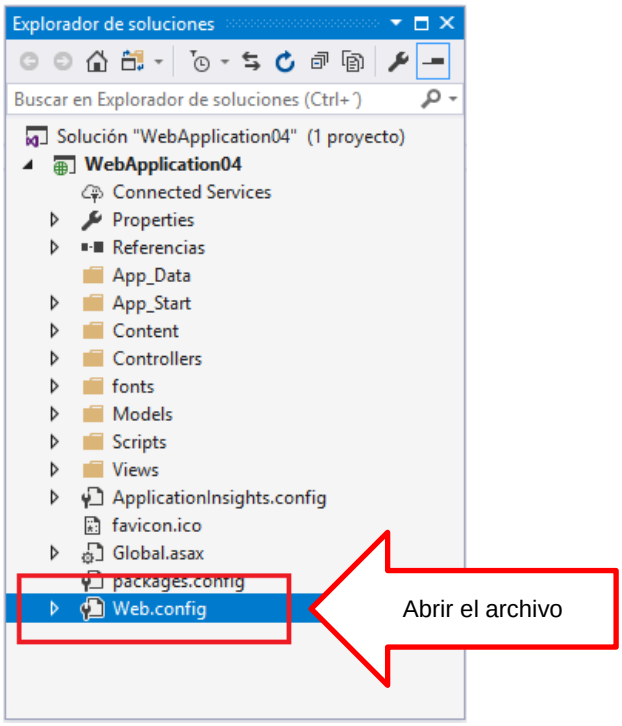


Para iniciar, agregamos la referencia DataAnnotations, luego defina los atributos de la clase, visualizando el texto y orden de cada uno de ellos [Display (Name="texto", Order="n")], tal como se muestra.

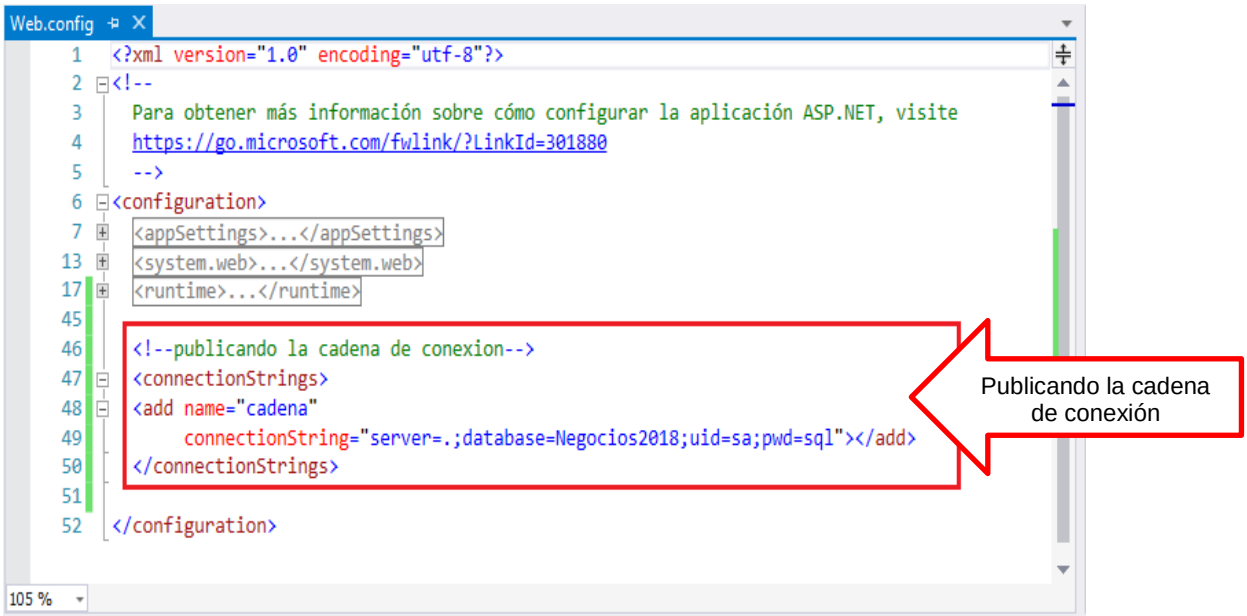


Publicando la cadena de conexión

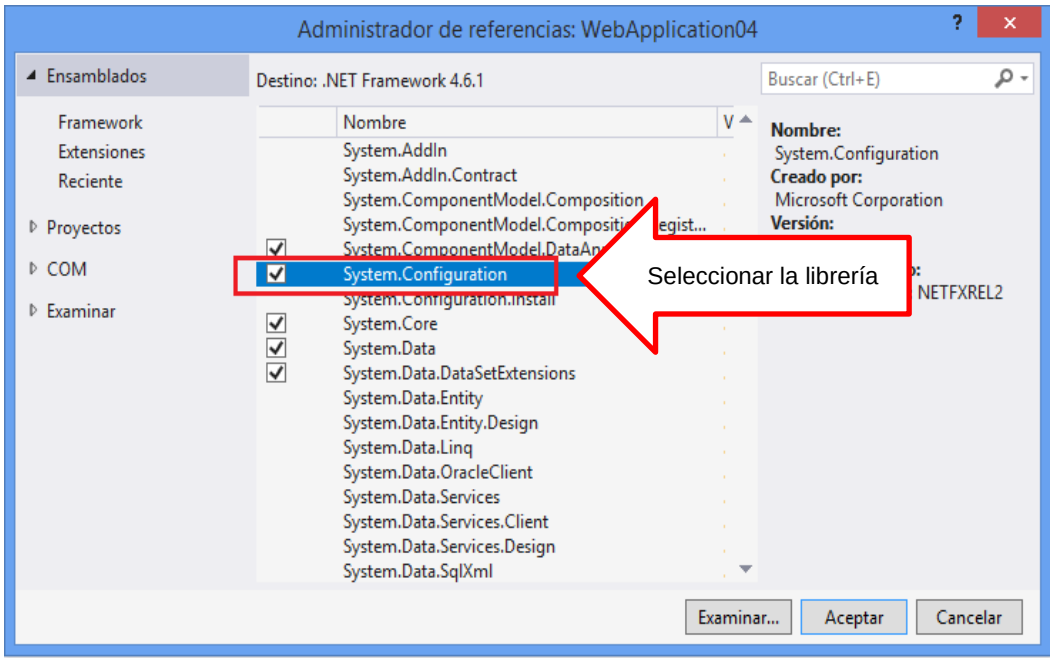
Abrir el archivo Web.config, tal como se muestra



Defina la etiqueta <connectionStrings> para agregar una cadena de conexión a la base de datos Negocios2018, tal como se muestra.

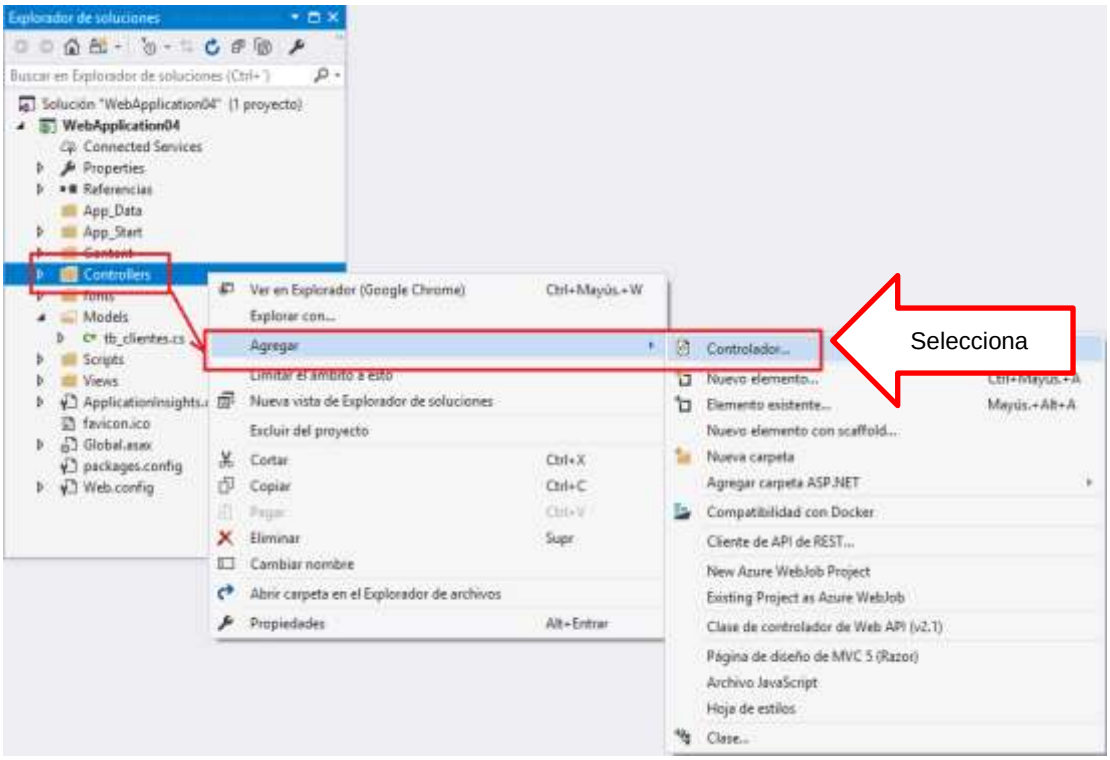


A continuación agregar la referencia al proyecto:
Selecciona desde la opción **Proyectos → Agregar Referencia**, selecciona la librería System.Configuration, tal como se muestra. Presiona el botón ACEPTAR.



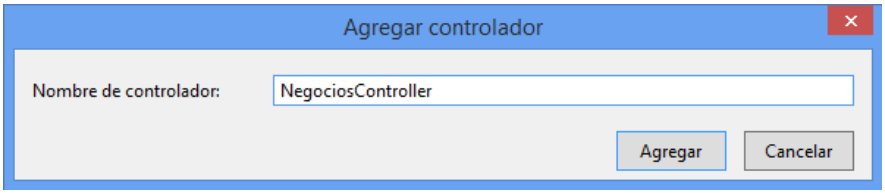
Creando un Controlador

A continuación, creamos un controlador: Desde la carpeta Controllers, selecciona la opción Agregar, Controlador…, tal como se muestra.

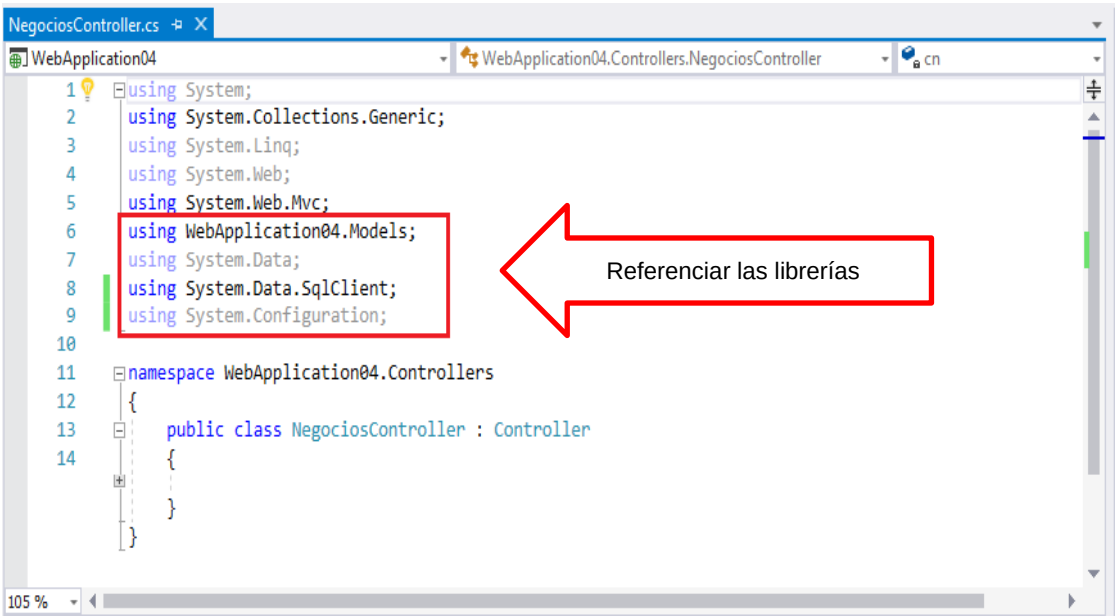


Selecciona el scaffold, Controlador de MVC5: en blanco, presiona el botón AGREGAR

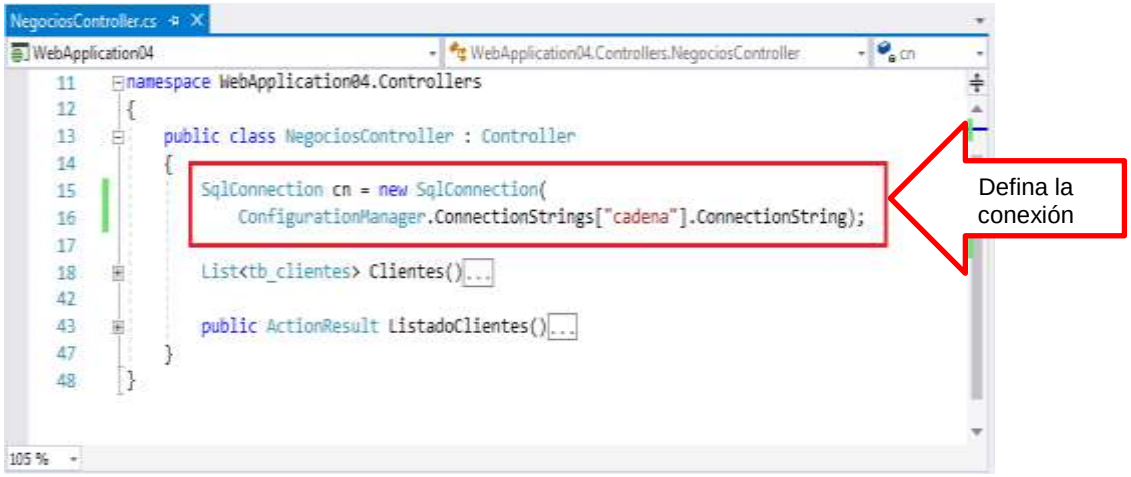
Asigne el nombre del controlador: NegociosController, tal como se muestra



Referenciar la carpeta **Models**, la librería **Data**.**SqlClient** y el **System**.**Configuration** tal como se muestra.



Dentro del controlador, defina la conexión a la base de datos Negocios2018 instanciando el **SqlConnection**, utilice el **ConfigurationManager**.

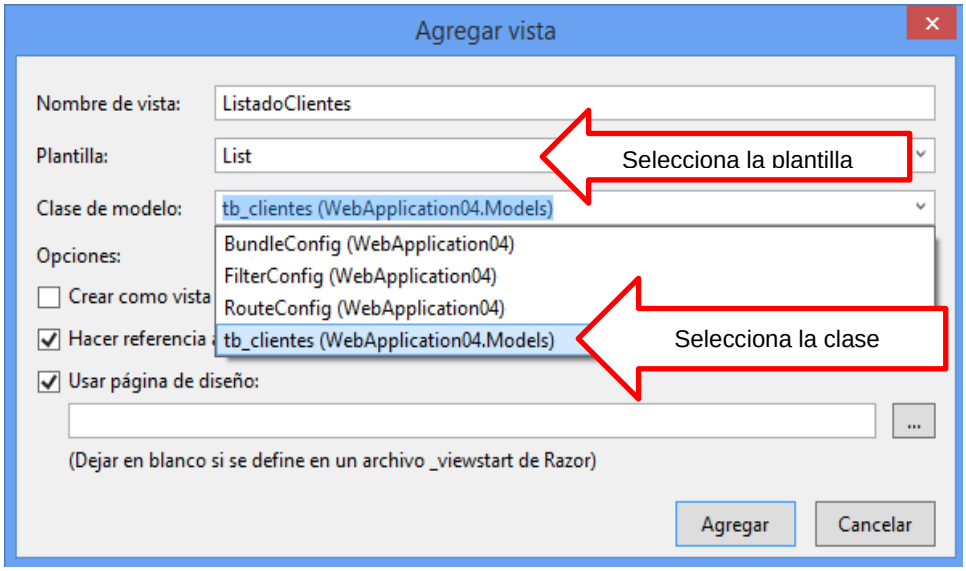


A continuación codifique la lista llamada Clientes:
1. Ejecuta el procedimiento almacenado usp_clientes a través del SqlCommand, almacenando los resultados en el SqlDataReader.
2. Para guardar los resultados en el temporal se realiza a través de un bucle (mientras se pueda leer: dr.Read()), vamos almacenando cada registro en el temporal.
3. Cerrar los objetos y enviar el objeto lista llamada temporal.

Defina el ActionResult ListadoClientes(), el cual enviará a la Vista el resultado del método Clientes(), tal como se muestra.

Agregando la Vista.

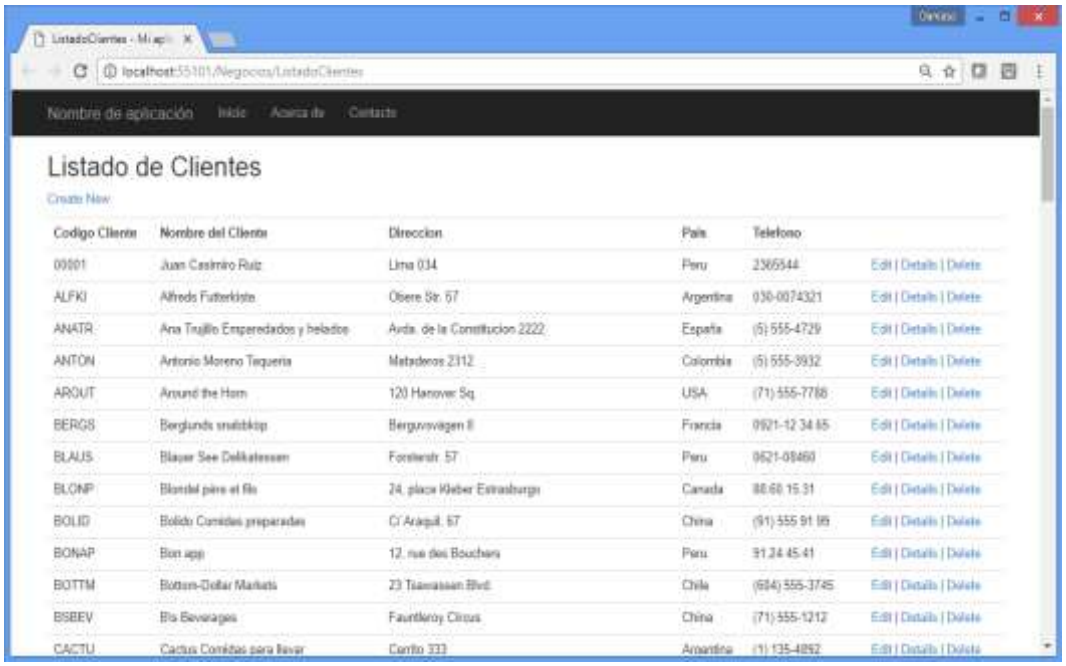
En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **List**; y la **clase de modelo** la cual es **tb_clientes**, tal como se muestra



Vista generada por la plantilla **List**, tal como se muestra


```
@model IEnumerable<WebApplication04.Models.tb_clientes>

@{ ViewBag.Title = "ListadoClientes"; }

<h2>Listado de Clientes</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.PrimaryKey */ }) |
            @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
            @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
        </td>
    </tr>
}
</table>
```

Ejecutamos la Vista, visualizando los registros de la tabla tb_clientes

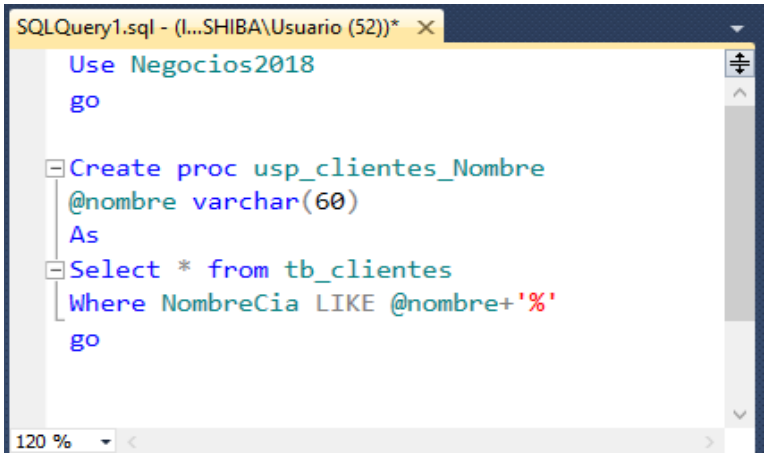


Laboratorio 4.2
Consulta de Datos con parámetros

Implemente un proyecto ASP.NET MVC, diseñe una página que permita listar los registros de la tabla tb_clientes de la base de datos Negocios2018, filtrando por las iniciales del campo NombreCia.

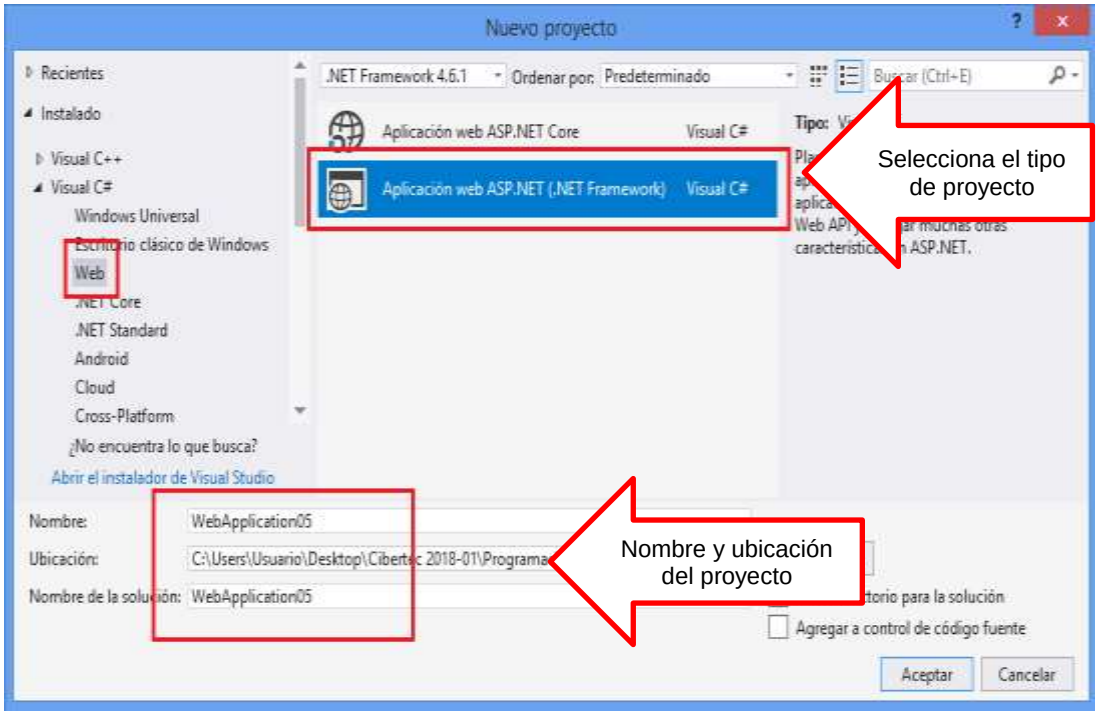
Creando el procedimiento almacenado

En el administrador del SQL Server, defina el procedimiento almacenado usp_clientes_Nombre, tal como se muestra.

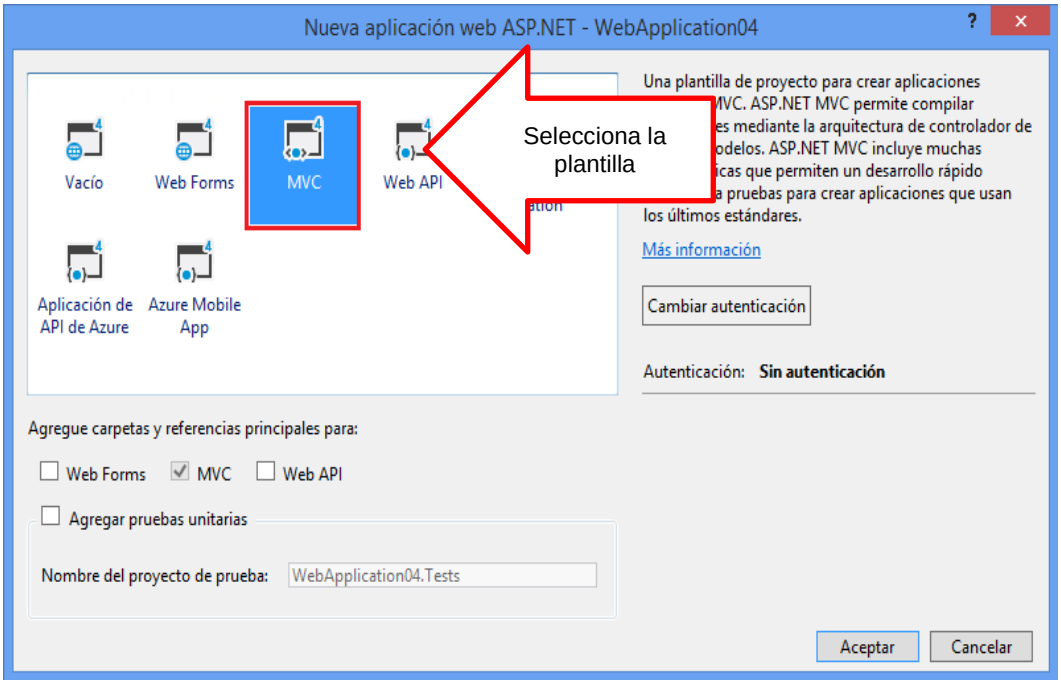


Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR

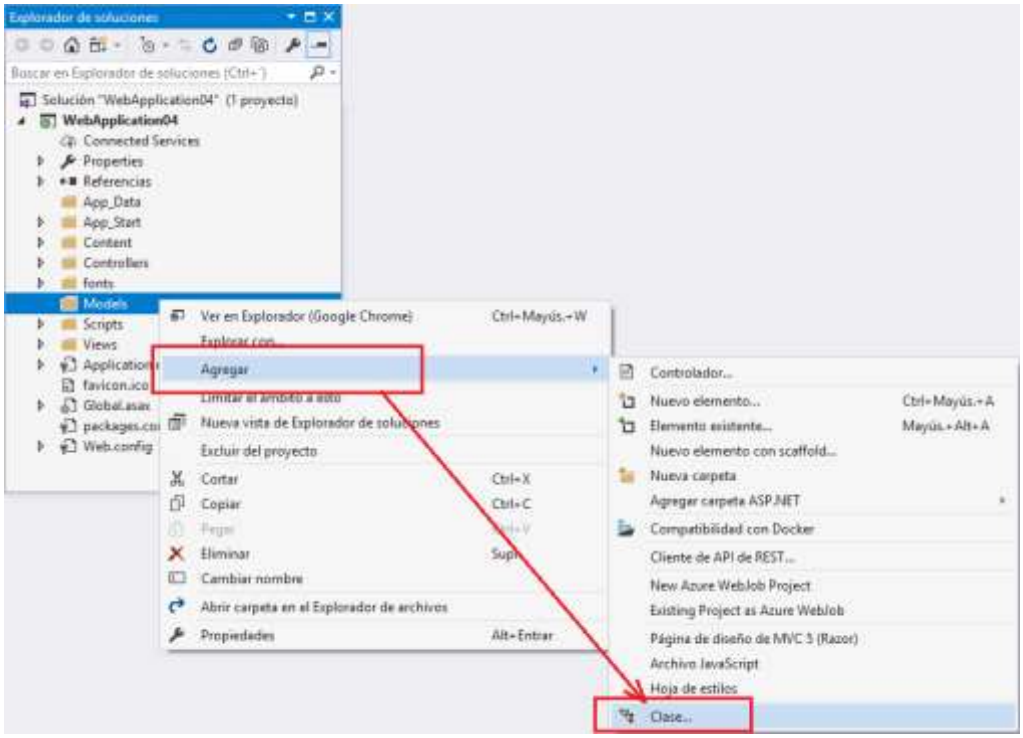


A continuación, seleccionar la plantilla del proyecto MVC.

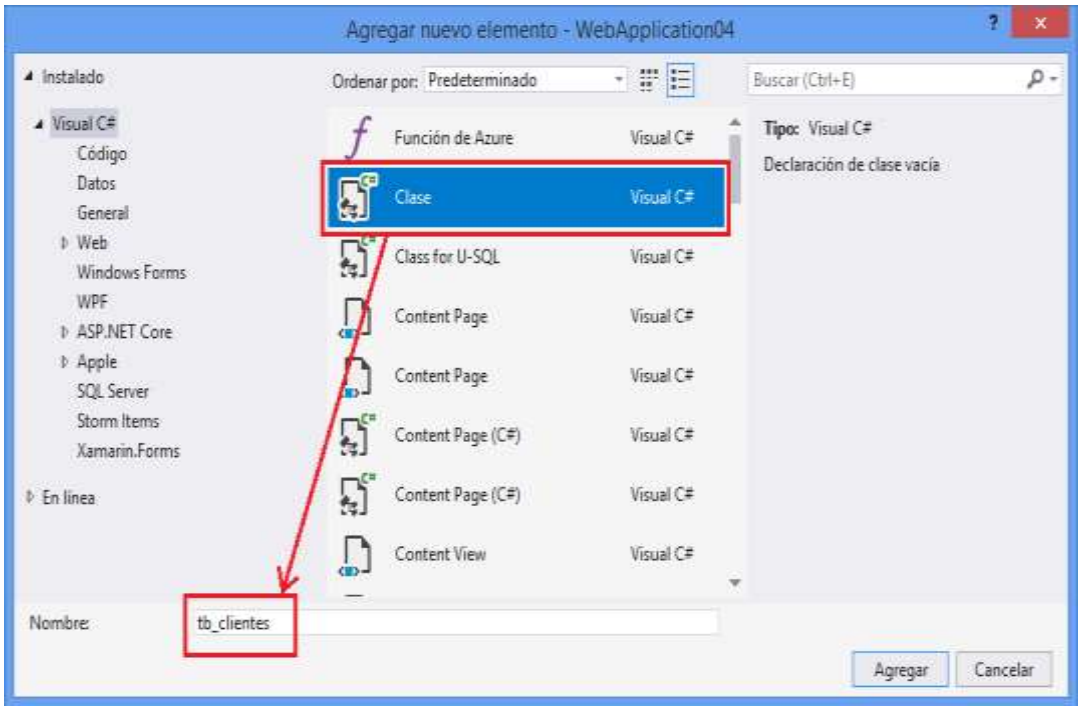


Creando la Clase del Modelo

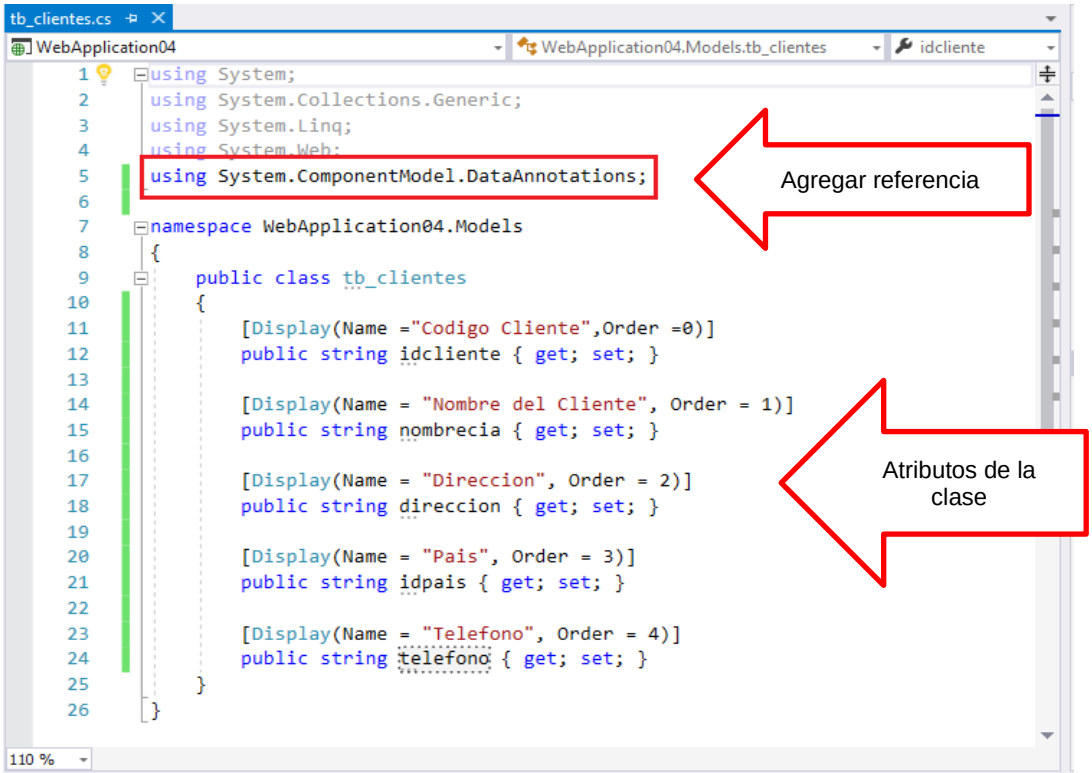
Para efectuar y realizar la consulta de los datos de la tabla tb_clientes, agregamos, en la carpeta Models, una clase, tal como se muestra.



Asigne el nombre de la clase, la cual se llamará tb_clientes, tal como se muestra. Presiona el botón AGREGAR

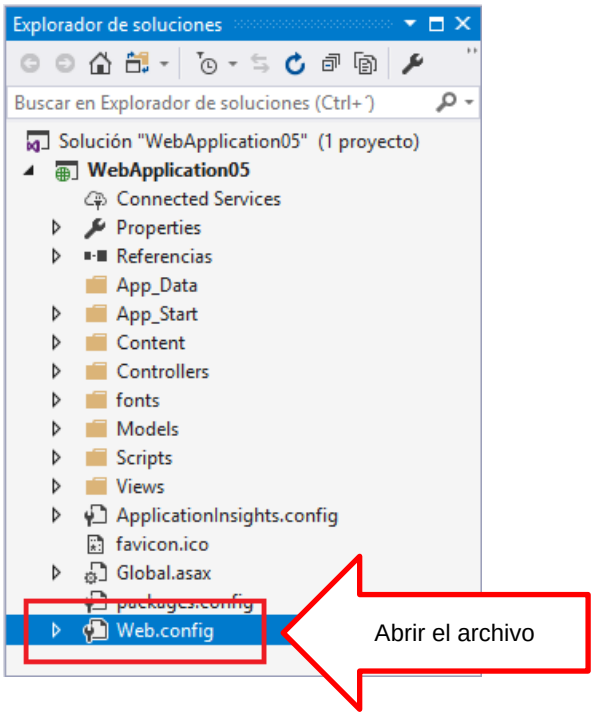


Para iniciar, agregamos la referencia DataAnnotations, luego defina los atributos de la clase, visualizando el texto y orden de cada uno de ellos [Display (Name="texto", Order="n")], tal como se muestra.

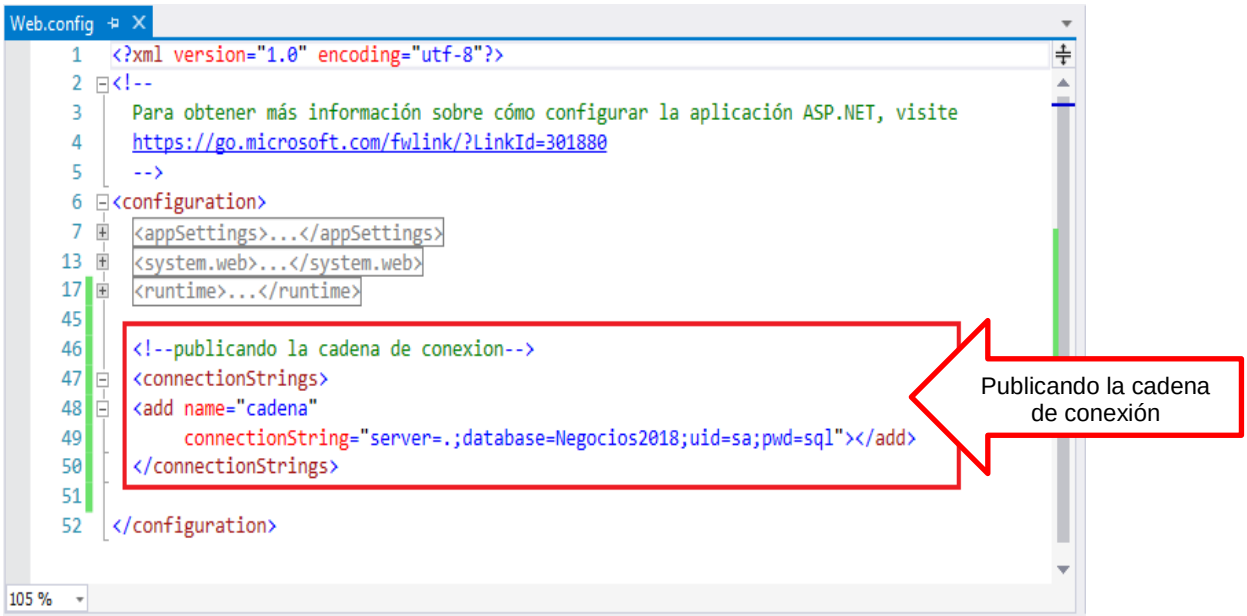


Publicando la cadena de conexión

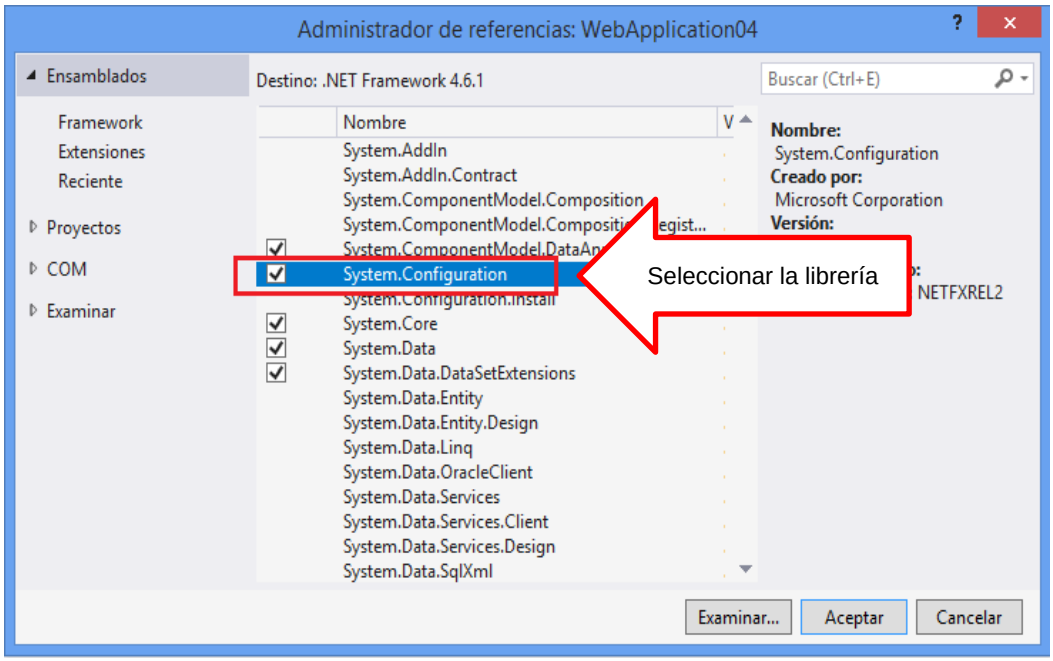
Abrir el archivo Web.config, tal como se muestra



Defina la etiqueta <connectionStrings> para agregar una cadena de conexión a la base de datos Negocios2018, tal como se muestra.

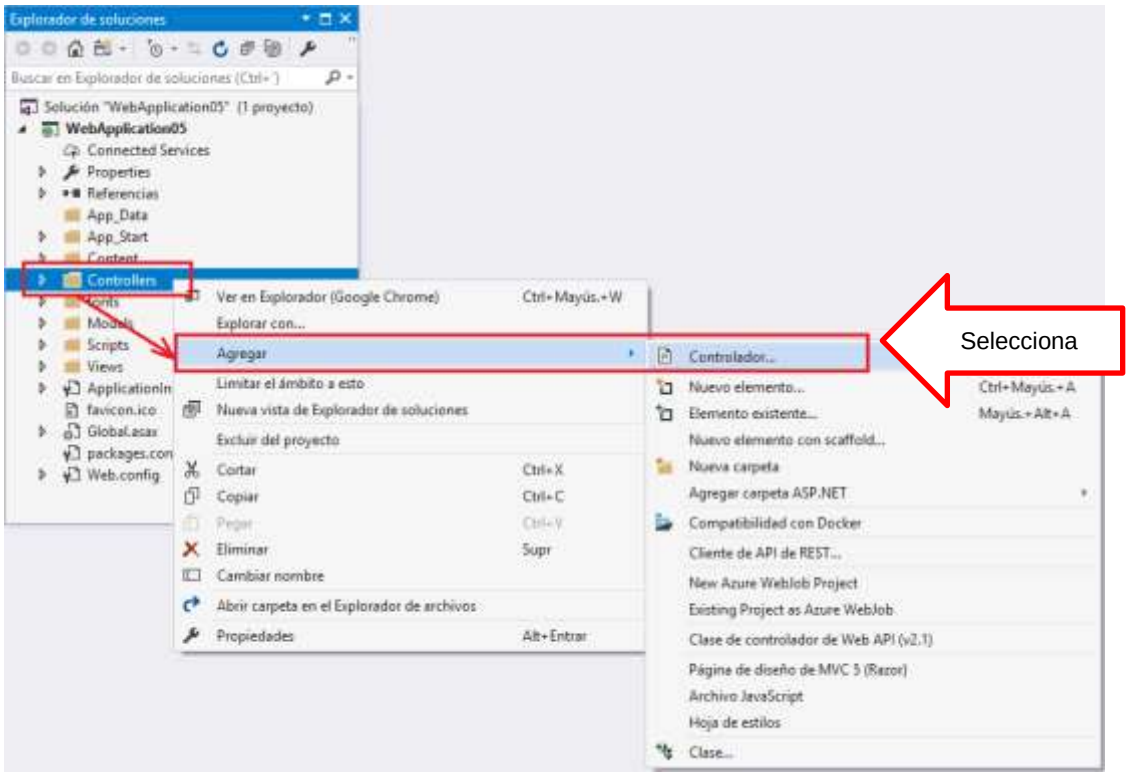


A continuación agregar la referencia al proyecto:
Selecciona desde la opción **Proyectos → Agregar Referencia**, selecciona la librería System.Configuration, tal como se muestra. Presiona el botón ACEPTAR.



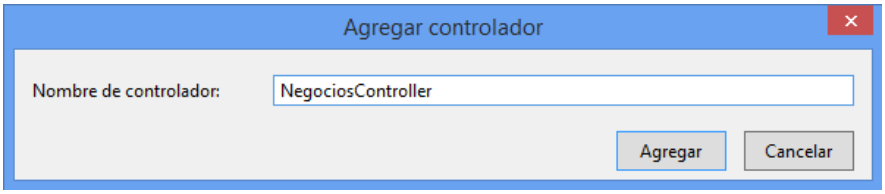
Creando un Controlador

A continuación, creamos un controlador: Desde la carpeta **Controllers**, selecciona la opción **Agregar**, **Controlador**…, tal como se muestra.

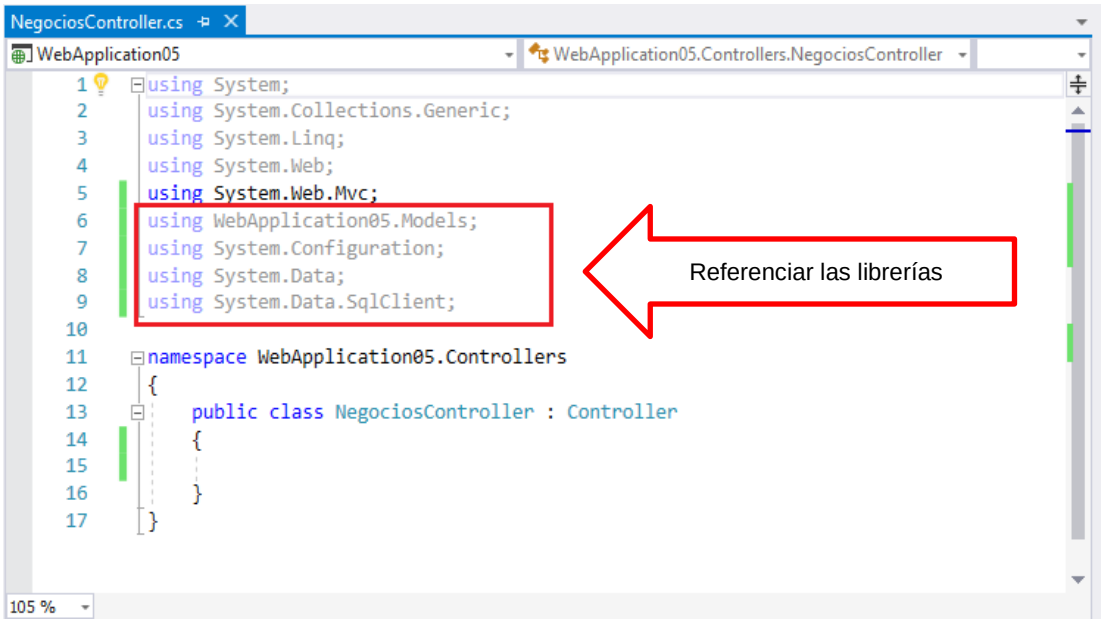


Selecciona el scaffold, Controlador de MVC5: en blanco, presiona el botón AGREGAR

Asigne el nombre del controlador: NegociosController, tal como se muestra

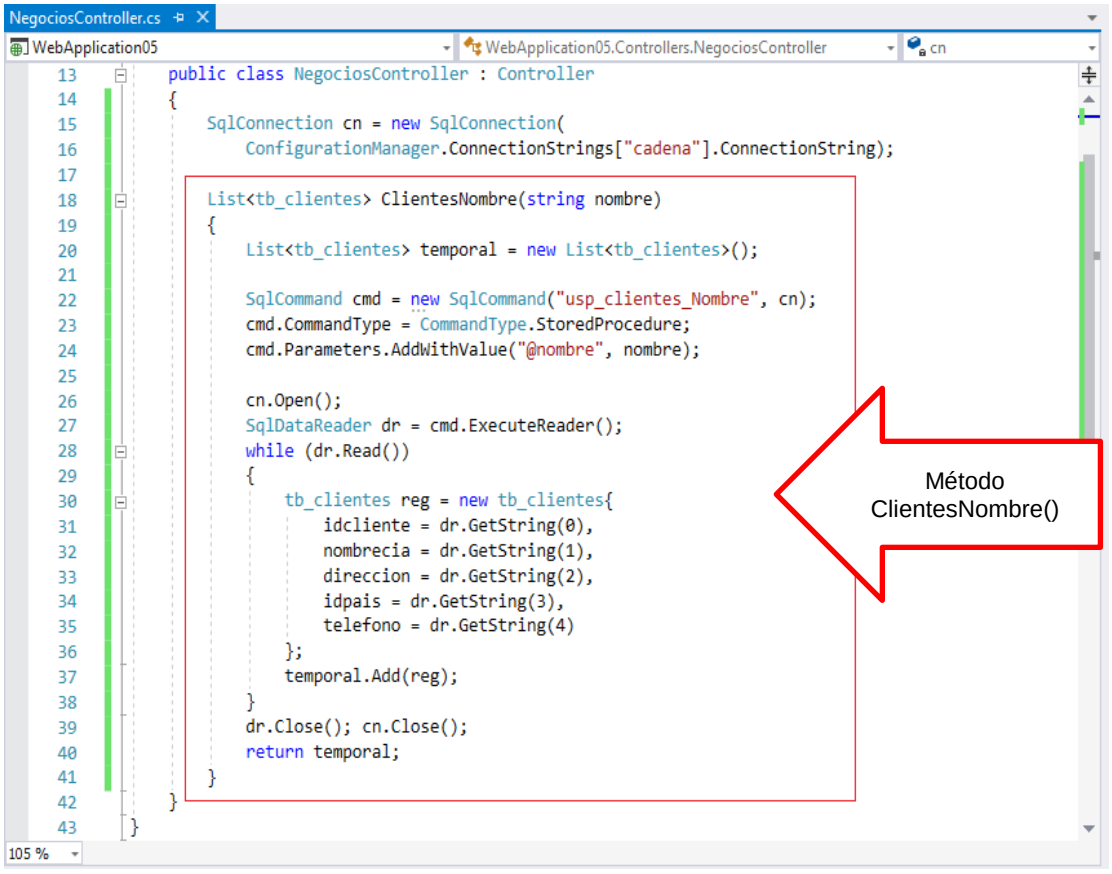


Referenciar la carpeta **Models**, la librería **Data**.**SqlClient** y el **System**.**Configuration** tal como se muestra.



Dentro del controlador, defina la conexión a la base de datos Negocios2018 instanciando el **SqlConnection**, utilice el **ConfigurationManager**.

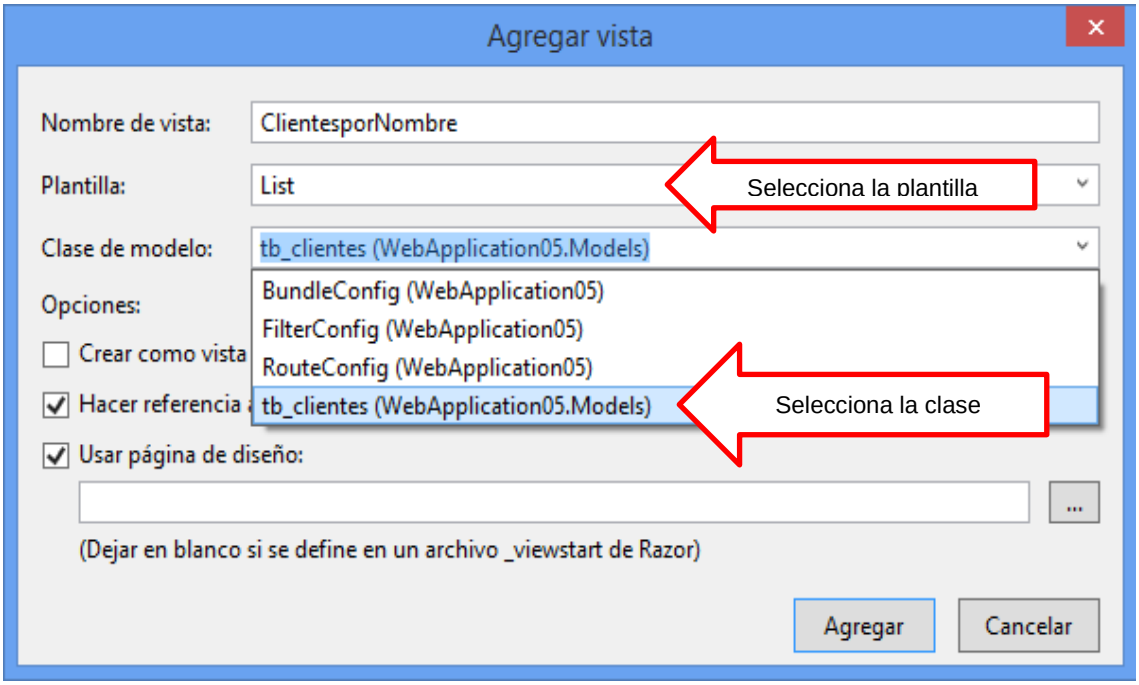
A continuación codifique la lista llamada ClientesNombre:
1. Ejecuta el procedimiento almacenado usp_clientes_Nombre a través del SqlCommand, pasando el valor de su parámetro @nombre. El resultado se almacena en el SqlDataReader.
2. Para guardar los resultados en el temporal se realiza a través de un bucle (mientras se pueda leer: dr.Read()), vamos almacenando cada registro en el temporal.
3. Cerrar los objetos y enviar el objeto lista llamada temporal.



Defina el ActionResult **ClientesporNombre**(string **nombre**), el cual enviará a la Vista el resultado del método **ClientesNombre**(). Inicialice el parámetro **nombre**, tal como se muestra.

Agregando la Vista.

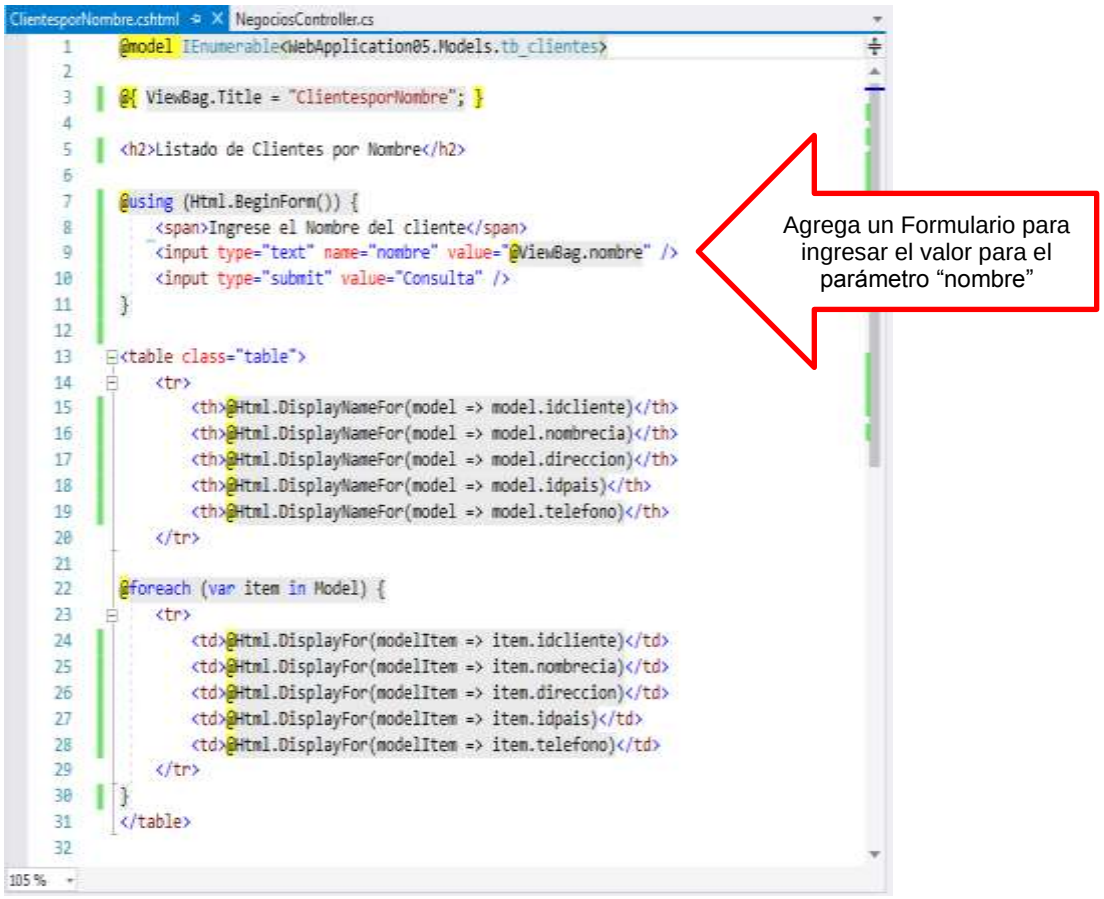
En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **List**; y la **clase de modelo** la cual es **tb_clientes**, tal como se muestra



Vista generada por la plantilla **List**, tal como se muestra


```
@model IEnumerable<WebApplication05.Models.tb_clientes>

@{ ViewBag.Title = "ClientesporNombre"; }

<h2>Listado de Clientes por Nombre</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.PrimaryKey */ }) |
            @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
            @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
        </td>
    </tr>
}
```

Modifica la Vista, agregando un formulario, tal como se muestra




```
@model IEnumerable<WebApplication05.Models.tb_clientes>

@{ ViewBag.Title = "ClientesporNombre"; }

<h2>Listado de Clientes por Nombre</h2>

@using (Html.BeginForm()) {
    <span>Ingrese el Nombre del cliente</span>
    <input type="text" name="nombre" value="@ViewBag.nombre" />
    <input type="submit" value="Consulta" />
}

<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>
    </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>
    </tr>
}
</table>
```

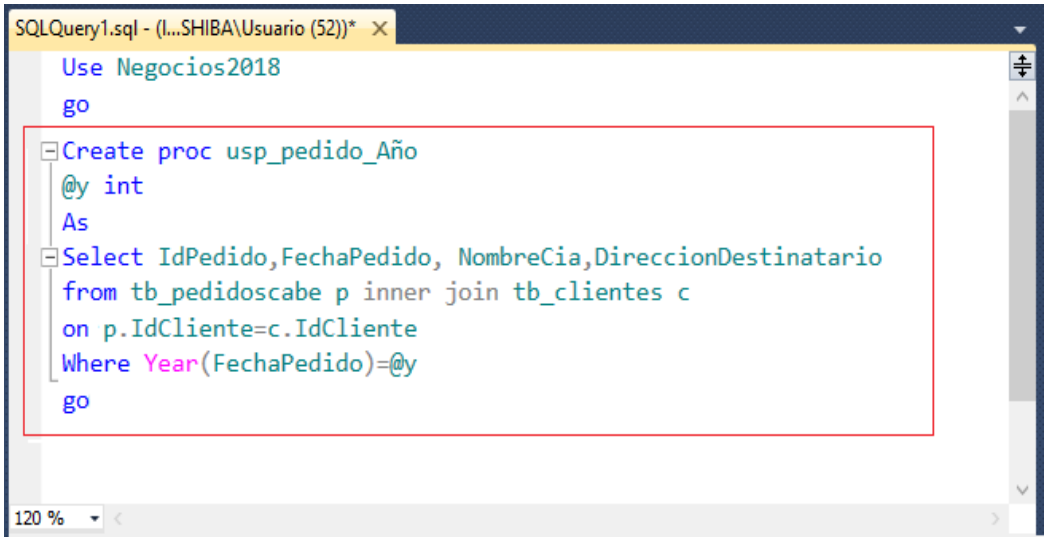
Ejecutamos la Vista, ingresa las iniciales en el campo text (nombre), al presionar el botón Consulta (submit) visualizamos los registros de la tabla tb_clientes

Sesión 02: Consulta de Datos con parámetros

Diseña una página que permita listar los registros de la tabla tb_pedidoscabe (idpedido, FechaPedido, NombreCia (tb_clientes), direccionDestinatario de la base de datos Negocios2018, filtrando por un determinado año del campo FechaPedido.

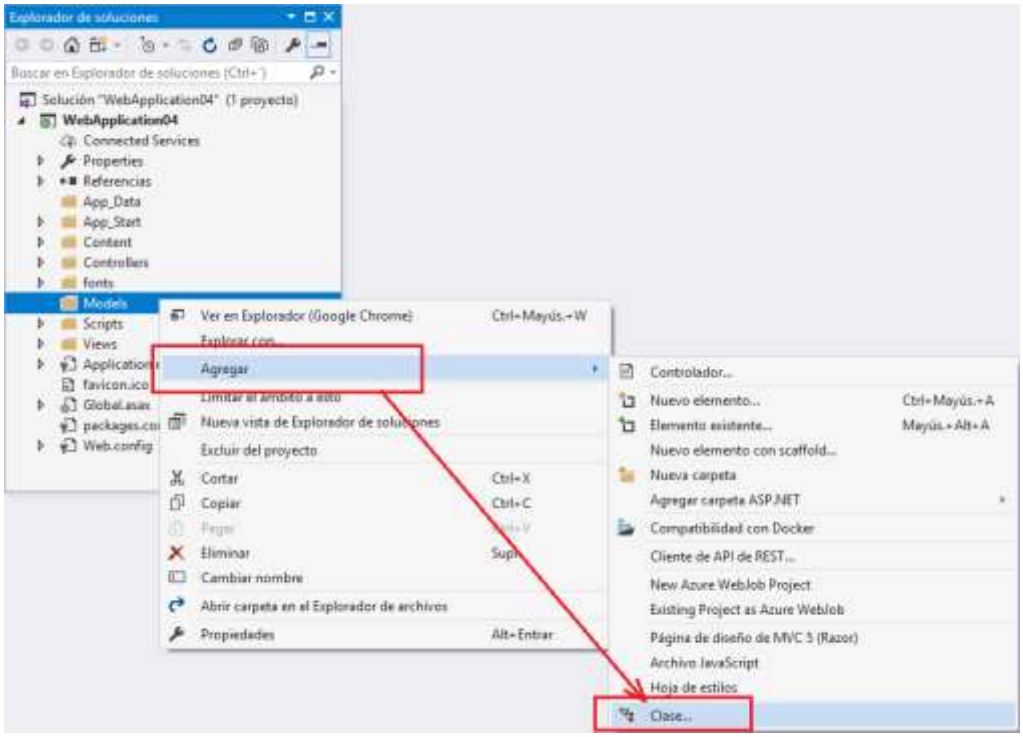
Creando el procedimiento almacenado

En el administrador del SQL Server, defina el procedimiento almacenado usp_pedido_Año, tal como se muestra

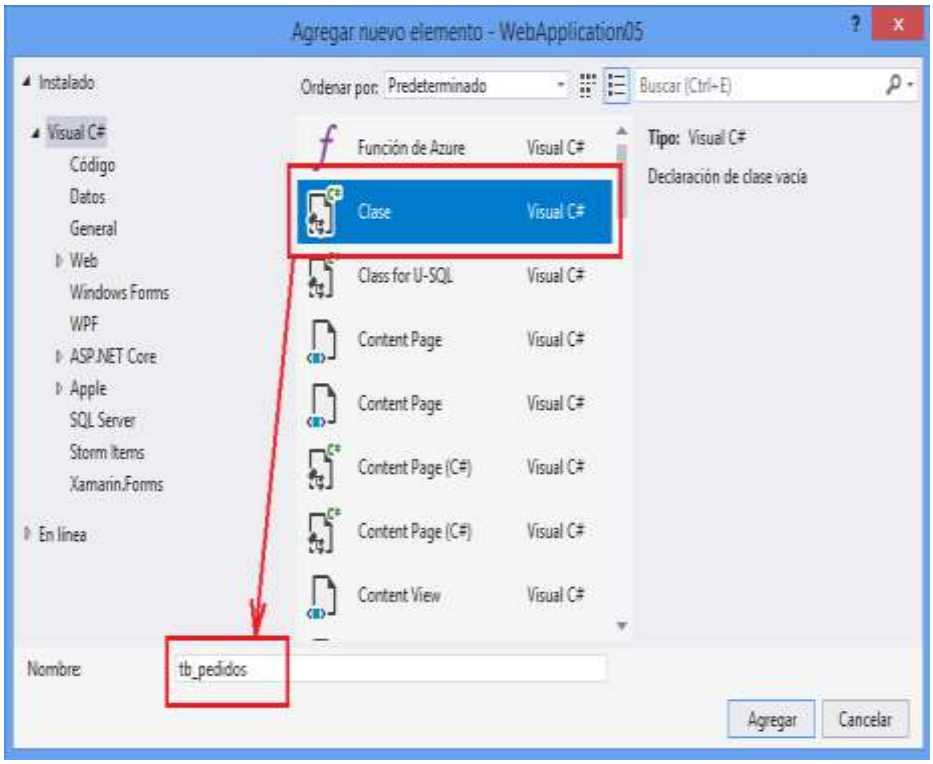


Creando la Clase del Modelo

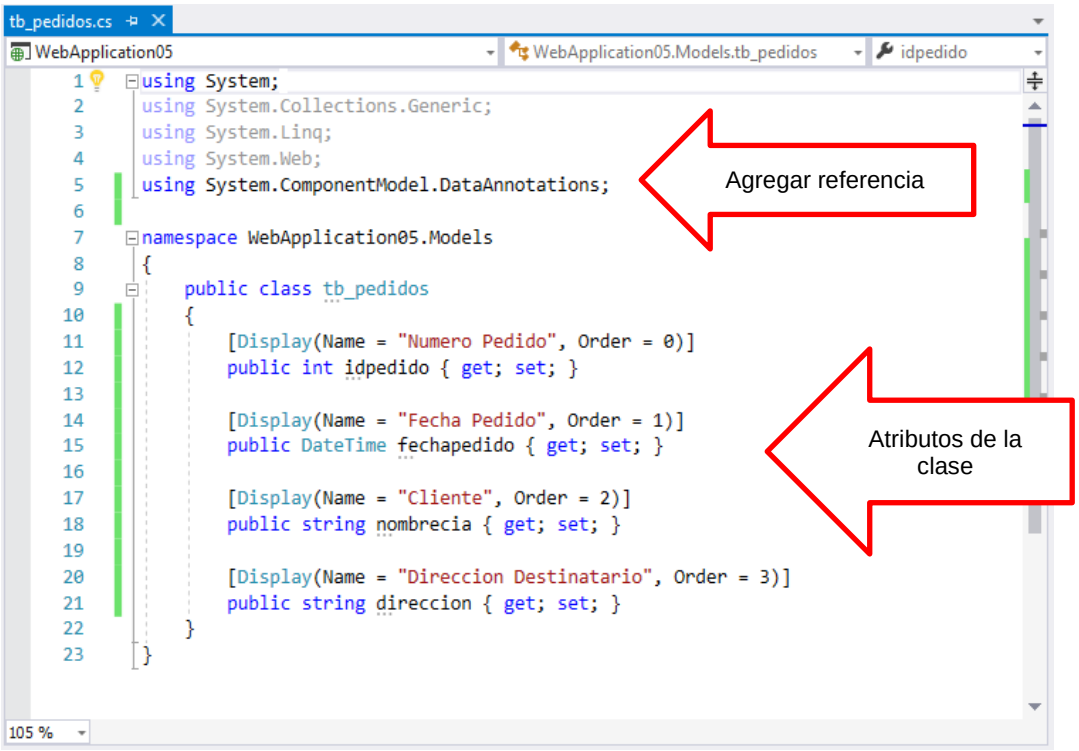
Para efectuar y realizar la consulta de los datos de la tabla tb_pedidos, agregamos, en la carpeta Models, una clase, tal como se muestra.



Asigne el nombre de la clase, la cual se llamará tb_pedidos, tal como se muestra. Presiona el botón AGREGAR

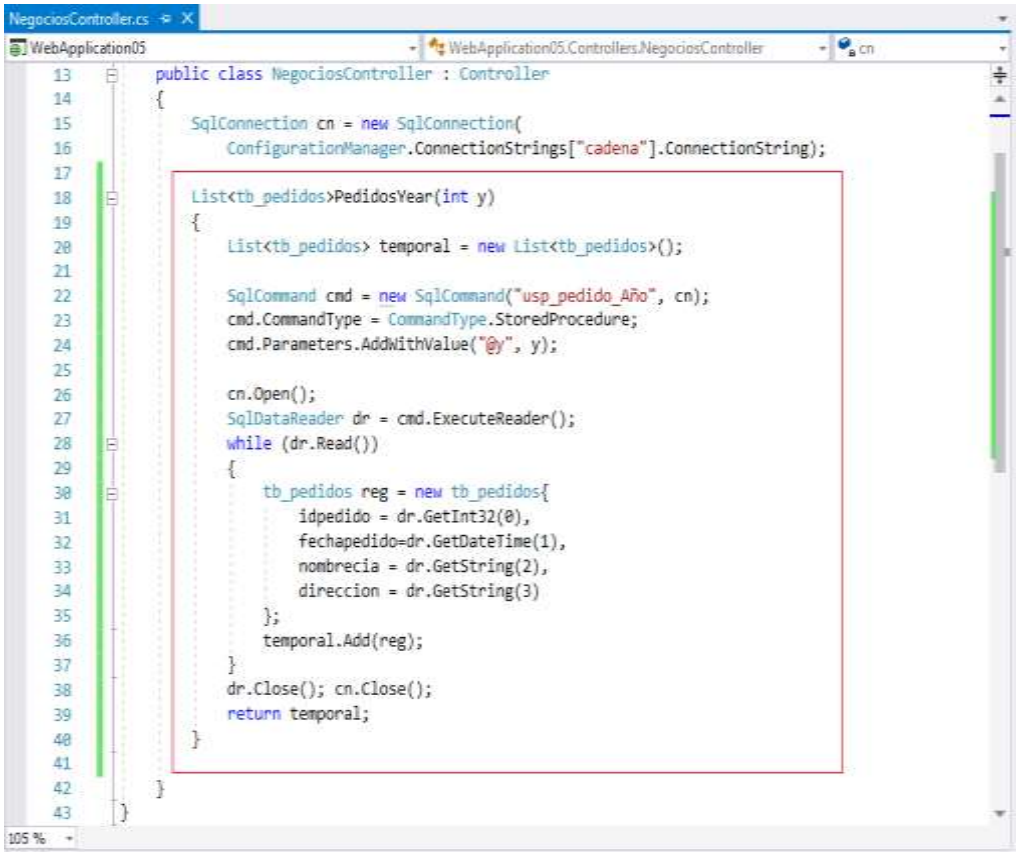


Para iniciar, agregamos la referencia DataAnnotations, luego defina los atributos de la clase, visualizando el texto y orden de cada uno de ellos [Display (Name="texto", Order="n")], tal como se muestra.

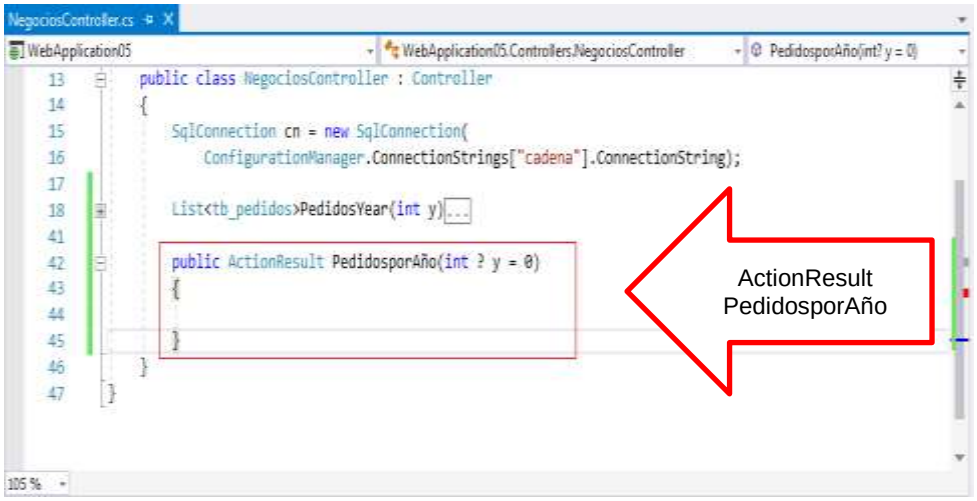


Trabajando con el ActionResult del Controlador

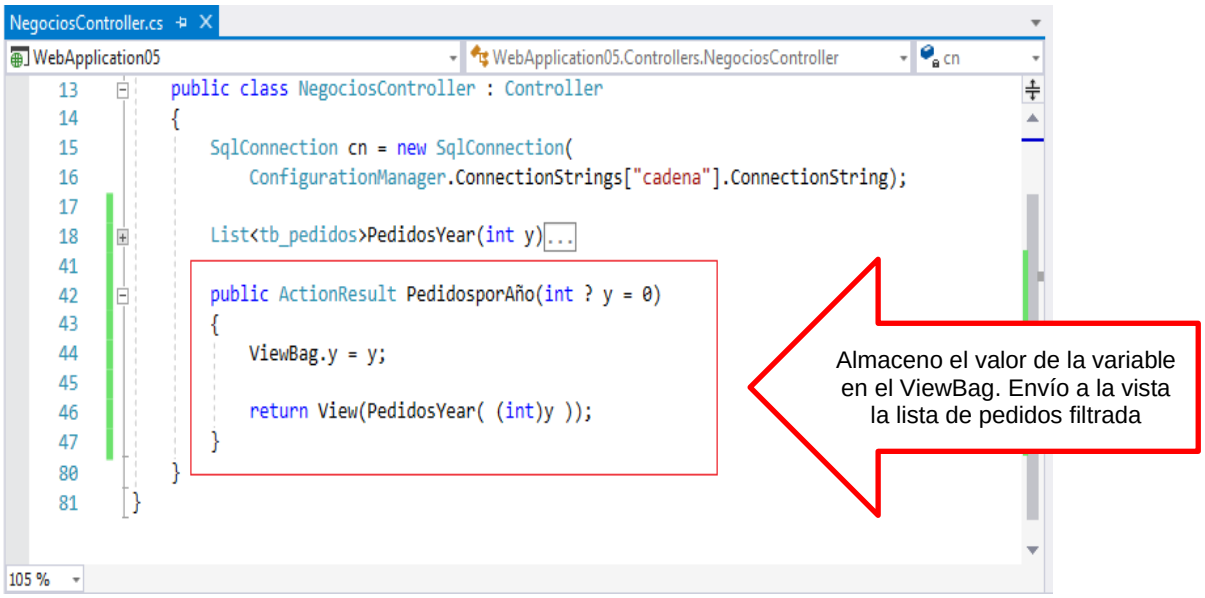
En el Controlador **NegociosController**, defina la lista llamada **PedidosYear**:
1. Defina el parámetro y (int y)
2. Ejecuta el procedimiento almacenado usp_pedido_Year a través del SqlCommand, pasando el valor de su parámetro @nombre.
3. El resultado se almacena en el SqlDataReader.
2. Para guardar los resultados en el temporal se realiza a través de un bucle (mientras se pueda leer: dr.Read()), vamos almacenando cada registro en el temporal.
3. Cerrar los objetos y enviar el objeto lista llamada temporal



A continuación, defina el **ActionResult PedidosporAño**, tal como se muestra

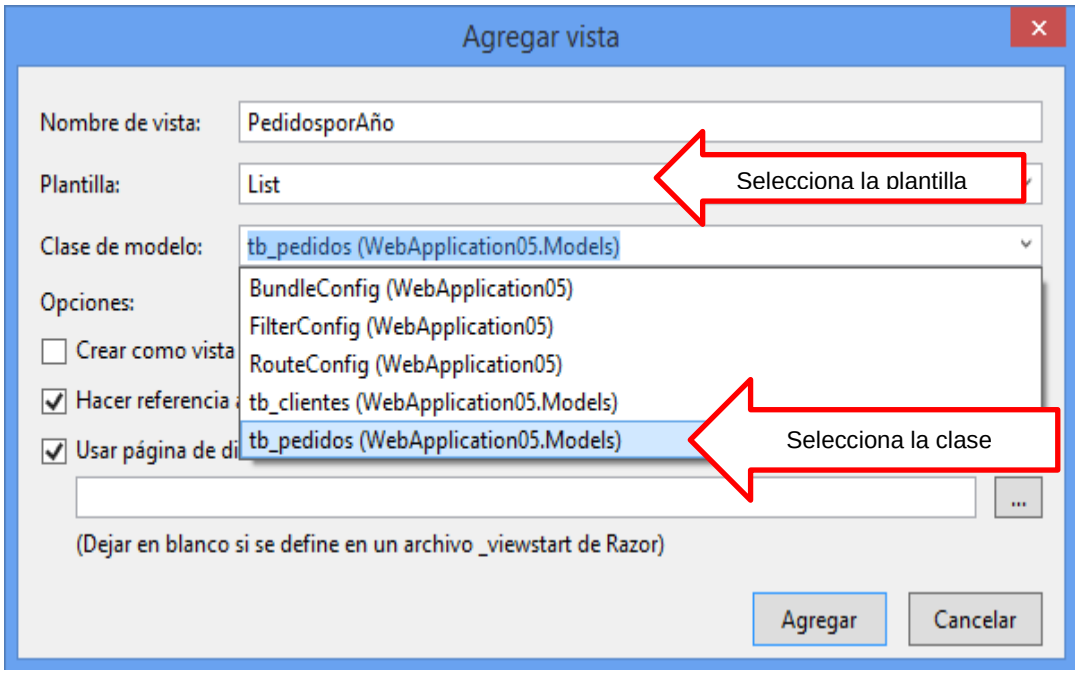


Defina en el ActionResult el **ViewBag.y** donde almaceno el valor y; retornando a la Vista el resultado del método PedidosYear filtrando con el valor del parámetro.



Agregando la Vista.

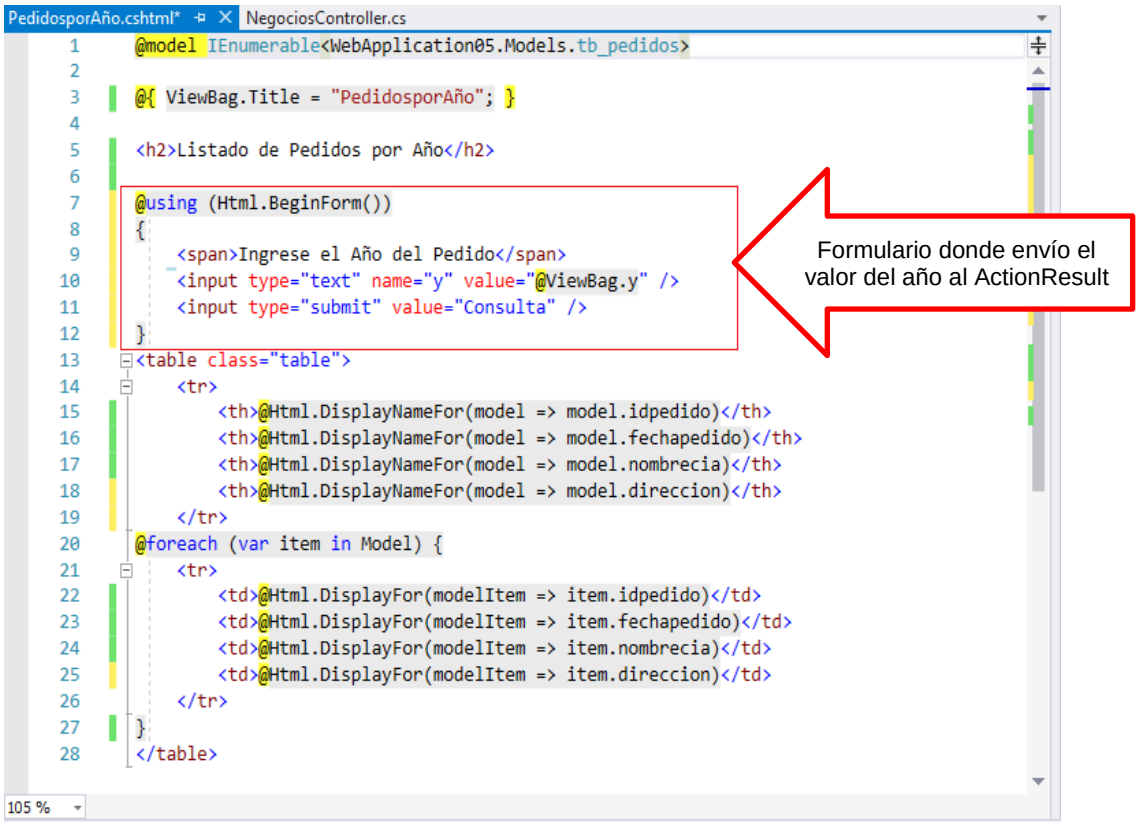
En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **List**; y la **clase de modelo** la cual es **tb_pedidos**, tal como se muestra



Vista generada por la plantilla **List**, tal como se muestra

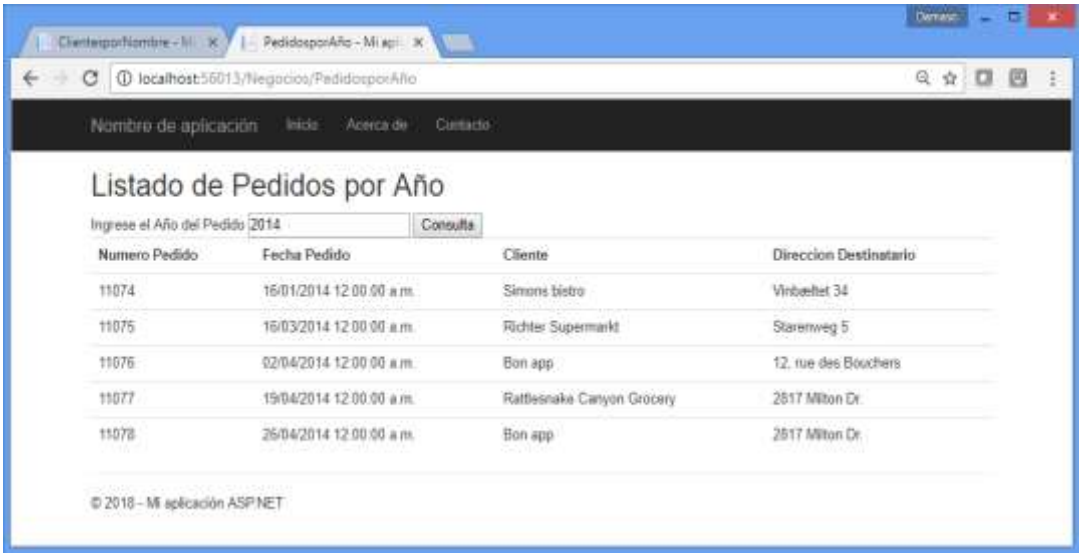

```
@model IEnumerable<WebApplication05.Models.tb_pedidos>

@{ ViewBag.Title = "PedidosporAño"; }

<h2>Listado de Pedidos por Año</h2>

<p>@Html.ActionLink("Create New", "Create")</p>
<table class="table">
    <tr>
        <th>@Html.DisplayNameFor(model => model.idpedido)</th>
        <th>@Html.DisplayNameFor(model => model.fechapedido)</th>
        <th>@Html.DisplayNameFor(model => model.nombrecia)</th>
        <th>@Html.DisplayNameFor(model => model.direccion)</th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>@Html.DisplayFor(modelItem => item.idpedido)</td>
        <td>@Html.DisplayFor(modelItem => item.fechapedido)</td>
        <td>@Html.DisplayFor(modelItem => item.nombrecia)</td>
        <td>@Html.DisplayFor(modelItem => item.direccion)</td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
            @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
            @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
        </td>
    </tr>
}
</table>
```

Modifica la Vista, agregando un formulario, tal como se muestra




```
@model IEnumerable<WebApplication05.Models.tb_pedidos>

@{ ViewBag.Title = "PedidosporAño"; }

<h2>Listado de Pedidos por Año</h2>

@using (Html.BeginForm())
{
    <span>Ingrese el Año del Pedido</span>
    <input type="text" name="y" value="@ViewBag.y" />
    <input type="submit" value="Consulta" />
}
<table class="table">
    <tr>
        <th>@Html.DisplayNameFor(model => model.idpedido)</th>
        <th>@Html.DisplayNameFor(model => model.fechapedido)</th>
        <th>@Html.DisplayNameFor(model => model.nombrecia)</th>
        <th>@Html.DisplayNameFor(model => model.direccion)</th>
    </tr>
@foreach (var item in Model) {
    <tr>
        <td>@Html.DisplayFor(modelItem => item.idpedido)</td>
        <td>@Html.DisplayFor(modelItem => item.fechapedido)</td>
        <td>@Html.DisplayFor(modelItem => item.nombrecia)</td>
        <td>@Html.DisplayFor(modelItem => item.direccion)</td>
    </tr>
}
</table>
```

Ejecutamos la Vista, ingresa el año en el campo text (y), al presionar el botón Consulta (submit) visualizamos los registros de la tabla tb_pedidos



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, DropDown lists y list boxes.
- Hay también un HTML Helper para el elemento Label, los cuales nos asocial descriptivas de texto a un formulario Web. Cada HTML Helper nos da una forma rápida de crear HTML valido 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
    - http://andresfelipetrujillo.com/2014/08/08/aprendiendo-asp-net-mvc-4-parte-5-vistas/

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 5: Manipulación de datos (5 horas)

5.1 Operaciones de actualización sobre un origen de datos.
5.2 Manejo de la clase Command
5.3 Ejecutando operaciones de actualización de datos con sentencias SQL o procedimiento almacenado.
5.4 Manejo de transacciones: clase Transaction.
5.5 Implementando un proceso con transacciones explicitas.

ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando el patrón ASP.NET MVC para realizar operaciones de actualización de datos.
- Los alumnos desarrollan los laboratorios de esta semana

5. OPERACIÓN DE MODIFICACION DE DATOS SOBRE UN ORIGEN DE DATOS

Las instrucciones SQL que modifican datos (por ejemplo INSERT, UPDATE o DELETE) no devuelven ninguna fila. De la misma forma, muchos procedimientos almacenados realizan alguna acción pero no devuelven filas. Para ejecutar comandos que no devuelvan filas, cree un objeto Command con el comando SQL adecuado y una Connection, incluidos los Parameters necesarios. El comando se debe ejecutar con el método ExecuteNonQuery del objeto Command.

El método ExecuteNonQuery devuelve un entero que representa el número de filas que se ven afectadas por la instrucción o por el procedimiento almacenado que se haya ejecutado. Si se ejecutan varias instrucciones, el valor devuelto es la suma de los registros afectados por todas las instrucciones ejecutadas.

5.1 Manejo de la clase COMMAND

Una vez establecida una conexión a un origen de datos, puede ejecutar comandos y devolver resultados desde el mismo mediante un objeto **DbCommand**.

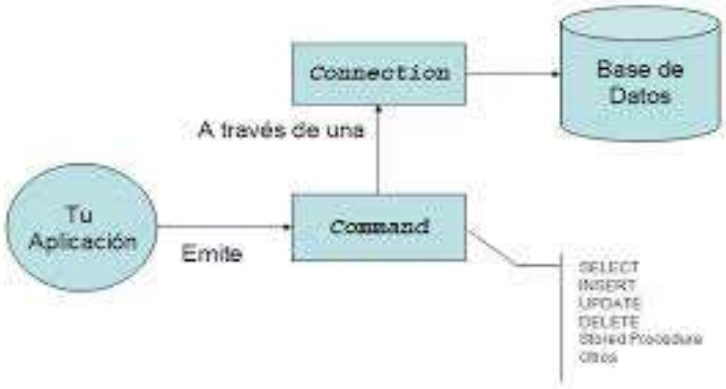


Figura 1: Diagrama del objeto Command
Referencia: http://www.dotnetero.com/2006/08/adonet-para-novatos.html

Para crear un comando, puede utilizar uno de los constructores de comando del proveedor de datos .NET Framework con el que esté trabajando. Los constructores pueden aceptar argumentos opcionales, como una instrucción SQL para ejecutar en el origen de datos, un objeto **DbConnection** o un objeto **DbTransaction**. También puede configurar dichos objetos como propiedades del comando. También puede crear un comando para una determinada conexión mediante el método **CreateCommand** de un objeto **DbConnection**. La instrucción SQL que ejecuta el comando se puede configurar mediante la propiedad CommandText.

Cada proveedor de datos de .NET FrameWork cuenta con un objeto Command:

Proveedor	Descripcion
OledbCommand	Proveedor de datos para OLEDB
SqlCommand	Proveedor de datos para SQL Server
OdbcCommand	Proveedor de datos para ODBC
OracleCommand	Proveedor de datos para Oracle

EJECUCION DE UN OBJETO COMMAND

Cada proveedor de datos .NET Framework incluido en .NET Framework dispone de su propio objeto command que hereda de DbCommand.

El proveedor de datos .NET Framework para OLE DB incluye un objeto **OleDbCommand**, el proveedor de datos .NET Framework para SQL Server incluye un objeto **SqlCommand**, el proveedor de datos .NET Framework para ODBC incluye un objeto **OdbcCommand** y el proveedor de datos .NET Framework para Oracle incluye un objeto **OracleCommand**.

Cada uno de estos objetos expone métodos para ejecutar comandos que se basan en el tipo de comando y el valor devuelto deseado, tal como se describe en la tabla siguiente:

Commando	Valor de retorno
ExecuteReader	Devuelve un objeto **DataReader**.
ExecuteScalar	Devuelve un solo valor escalar.
ExecuteNonQuery	Ejecuta un comando que no devuelve ninguna fila.
ExecuteXMLReader	Devuelve un valor **XmlReader**. Solo está disponible para un objeto SqlCommand.

Cada objeto command fuertemente tipado admite también una enumeración **CommandType** que especifica cómo se interpreta una cadena de comando, tal como se describe en la tabla siguiente

Commando	Valor de retorno
Text	Comando de SQL que define las instrucciones que se van a ejecutar en el origen de dato
StoredProcedure	Nombre del procedimiento almacenado. Puede usar la propiedad **Parameters** de un comando para tener acceso a los parámetros de entrada y de salida y a los valores devueltos, independientemente del método Execute al que se llame. Al usar **ExecuteReader**, no es posible el acceso a los valores devueltos y los parámetros de salida hasta que se cierra **DataReader**
TableDirect	Nombre de una tabla.

EJECUCIÓN DE UNA CONSULTA QUE RETORNE UN CONJUNTO DE RESULTADOS

El objeto Command de ADO.NET tiene un método **ExecuteReader** que permite ejecutar una consulta que retorna uno o más conjunto de resultados.

Al ejecutar el método **ExecuteReader**, podemos pasar un parámetro al método el cual representa la enumeración **CommandBehavior**, que permite controlar al Command como se ejecutado.

A continuación mostramos la descripción de los enumerables del CommandBehavior:

Valor	Descripción
CommandBehavior.CloseConnection	Se utiliza para cerrar la conexión en forma automática, tan pronto como el dataReader es cerrado.
CommandBehavior.SingleResult	Retorna un único conjunto de resultados
CommandBehavior.SingleRow	Retorna una fila

5.2 EJECUTANDO OPERACIONES DE ACTUALIZACION DE DATOS UTILIZANDO SQL O PROCEDIMIENTO ALMACENADO

Los procedimientos almacenados ofrecen numerosas ventajas en el caso de aplicaciones que procesan datos. Mediante el uso de procedimientos almacenados, las operaciones de bases de datos se pueden encapsular en un solo comando, optimizar para lograr el mejor rendimiento, y mejorar con seguridad adicional.

Aunque es cierto que para llamar a un procedimiento almacenado basta con pasar en forma de instrucción SQL su nombre seguido de los argumentos de parámetros, el uso de la colección **Parameters** del objeto **DbCommand** de ADO.NET permite definir más explícitamente los parámetros del procedimiento almacenado, y tener acceso a los parámetros de salida y a los valores devueltos.

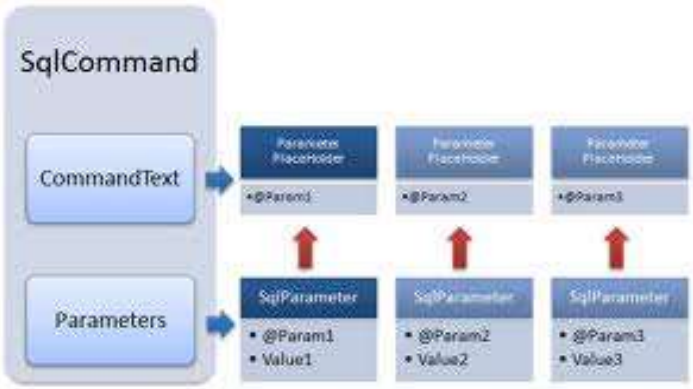


Figura 2: Diagrama del objeto SqlCommand
Referencia: http://yinyangit.wordpress.com/2011/08/05/ado-net-tutorial-lesson-06-adding-parameters-to-sqlcommands/

MANEJO DE PARAMETROS EN ACTUALIZACION DE DATOS

Cuando se usan parámetros con **SqlCommand** para ejecutar un procedimiento almacenado de SQL Server, los nombres de los parámetros agregados a la colección **Parameters** deben coincidir con los nombres de los marcadores de parámetro del procedimiento almacenado.

El proveedor de datos de .NET Framework para SQL Server no admite el uso del marcador de posición de signo de interrogación de cierre (?) para pasar parámetros a una instrucción SQL o a un procedimiento almacenado. Este proveedor trata los parámetros del procedimiento almacenado como parámetros con nombre y busca marcadores de parámetros coincidentes.

Para crear un objeto **DbParameter**, se puede usar su constructor o bien se puede agregar a **DbParameterCollection** mediante una llamada al método Add de la colección **DbParameterCollection**.

El método Add acepta como entrada argumentos del constructor o cualquier objeto de parámetro ya existente, en función del proveedor de datos.

En el caso de los parámetros que no sean de entrada (INPUT), debe de asignarse la propiedad ParameterDirection y especifique cual es el tipo de dirección del parámetro: **InputOutput**, **Output** o **ReturnValue**

El tipo de datos de un parámetro es específico del proveedor de datos de .NET Framework. Al especificar el tipo, el valor de Parameter se convierte en el tipo del proveedor de datos de

.NET Framework antes de pasar el valor al origen de datos. Si lo desea, puede especificar el tipo de un objeto Parameter de forma genérica estableciendo la propiedad **DbType** del objeto **Parameter** en un **DbType** determinado.

Las instrucciones SQL que modifican datos (por ejemplo INSERT, UPDATE o DELETE) no devuelven ninguna fila. De la misma forma, muchos procedimientos almacenados realizan alguna acción pero no devuelven filas. Para ejecutar comandos que no devuelvan filas, cree un objeto Command con el comando SQL adecuado y una Connection, incluidos los **Parameters** necesarios. El comando se debe ejecutar con el método **ExecuteNonQuery** del objeto **Command**.

El método **ExecuteNonQuery** devuelve un entero que representa el número de filas que se ven afectadas por la instrucción o por el procedimiento almacenado que se haya ejecutado. Si se ejecutan varias instrucciones, el valor devuelto es la suma de los registros afectados por todas las instrucciones ejecutadas.

5.3 MANEJO DE TRANSACCIONES EN .NET

Cuando se compra un libro de una librería en línea, se intercambia dinero (en forma de crédito) por el libro. Si tiene disponibilidad de crédito, una serie de operaciones relacionadas garantiza que se obtenga el libro y la librería obtiene el dinero. Sin embargo, si la operación sufre de un error durante el intercambio comercial, el erro afecta a la totalidad del proceso. No se obtiene el libro y la librería no obtiene el dinero.

Una transacción consiste en un comando único o en un grupo de comandos que se ejecutan como un paquete. Las transacciones permiten combinar varias operaciones en una sola unidad de trabajo. Si en un punto de la transacción se produjera un error, todas las actualizaciones podrían revertirse y devolverse al estado que tenían antes de la transacción.

Una transacción debe ajustarse a las propiedades: atomicidad, coherencia, aislamiento y durabilidad para poder garantizar la coherencia de datos. La mayoría de los sistemas de bases de datos relacionales, como Microsoft SQL Server, admiten transacciones, al proporcionar funciones de bloqueo, registro y administración de transacciones cada vez que una aplicación cliente realiza una operación de actualización, inserción o eliminación.

5.3.1 IMPLEMENTANDO UNA TRANSACCION IMPLICITA Y EXPLICITA

Una transacción explícita es aquella en que se define explícitamente el inicio y el final de la transacción. Las aplicaciones utilizan las instrucciones BEGIN TRANSACTION, COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION o ROLLBACK WORK de Transact-SQL para definir transacciones explícitas.

BEGIN TRANSACTION

Marca el punto de inicio de una transacción explícita para una conexión.

COMMIT TRANSACTION o COMMIT WORK

Se utiliza para finalizar una transacción correctamente si no hubo errores. Todas las modificaciones de datos realizadas en la transacción se convierten en parte permanente de la base de datos. Se liberan los recursos ocupados por la transacción.

ROLLBACK TRANSACTION o ROLLBACK WORK

Se utiliza para eliminar una transacción en la que se encontraron errores. Todos los datos modificados por la transacción vuelven al estado en el que estaban al inicio de la transacción. Se liberan los recursos ocupados por la transacción.

En ADO, utilice el método BeginTrans en un objeto Connection para iniciar una transacción explícita. Para finalizar la transacción, llame a los métodos CommitTrans o RollbackTrans del objeto Connection.

En el proveedor administrado de SqlCliente de ADO.NET, utilice el método BeginTransaction en un objeto SqlConnection para iniciar una transacción explícita. Para finalizar la transacción, llame a los métodos Commit() o Rollback() del objeto SqlTransaction.

El modo de transacciones explícitas se mantiene solamente durante la transacción. Cuando la transacción termina, la conexión vuelve al modo de transacción en que estaba antes de iniciar la transacción explícita, es decir, el modo implícito o el modo de confirmación automática.

Cada proveedor de datos de .NET Framework tiene su propio objeto Transaction para realizar transacciones locales. Si necesita que se realice una transacción en una base de datos de SQL Server, seleccione una transacción de **System**.**Data**.**SqlClient**. En transacciones de Oracle, utilice el proveedor **System**.**Data**.**OracleClient**. Además, existe una nueva clase **DbTransaction** disponible para la escritura de código independiente del proveedor que requiere transacciones.

En ADO.NET, las transacciones se controlan con el objeto Connection. Puede iniciar una transacción local con el método **BeginTransaction**. Una vez iniciada una transacción, puede inscribir un comando en esa transacción con la propiedad Transaction de un objeto **Command**. Luego, puede confirmar o revertir las modificaciones realizadas en el origen de datos según el resultado correcto o incorrecto de los componentes de la transacción. Las operaciones para confirmar una transacción es **Commit** y la operación para revertir o deshacer una transacción es **RollBack**.

Una transacción implícita inicia una nueva transacción en una conexión a SQL Server Database Engine (Motor de base de datos de SQL Server) después de confirmar o revertir la transacción actual. No tiene que realizar ninguna acción para delinear el inicio de una transacción, sólo tiene que confirmar o revertir cada transacción. El modo de transacciones implícitas genera una cadena continua de transacciones.

La transacción sigue activa hasta que emita una instrucción COMMIT o ROLLBACK. Una vez que la primera transacción se ha confirmado o revertido, la instancia Motor de base de datos inicia automáticamente una nueva transacción la siguiente vez que la conexión ejecuta una de estas instrucciones. La instancia continúa generando una cadena de transacciones implícitas hasta que se desactiva el modo de transacciones implícitas.

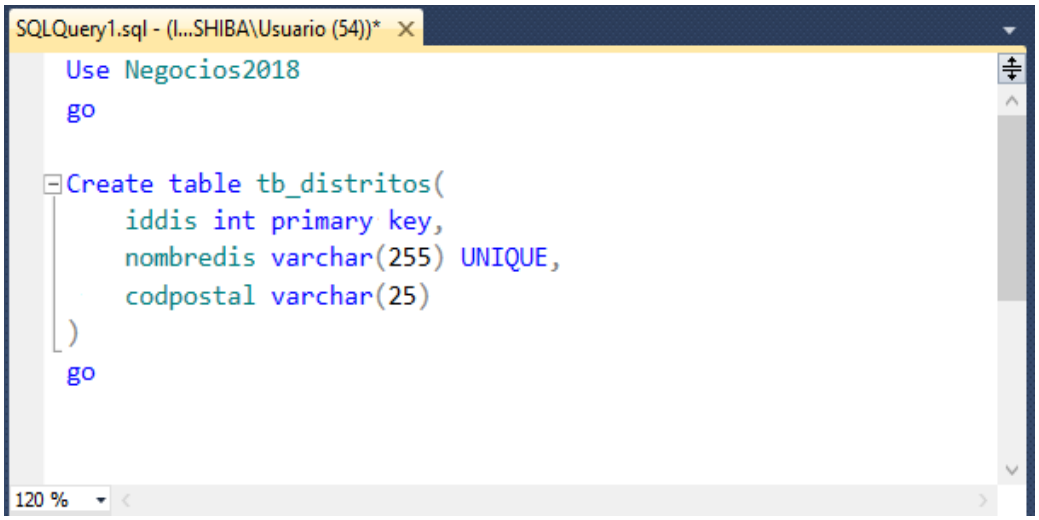
El modo de transacciones implícitas se establece mediante la instrucción SET de Transact-SQL o a través de funciones y métodos de la API de bases de datos.

Laboratorio 5.1
Manipulación de Datos

Implemente un proyecto ASP.NET MVC, donde permita INSERTAR, CONSULTAR y ACTUALIZAR los datos de la tabla tb_distritos.

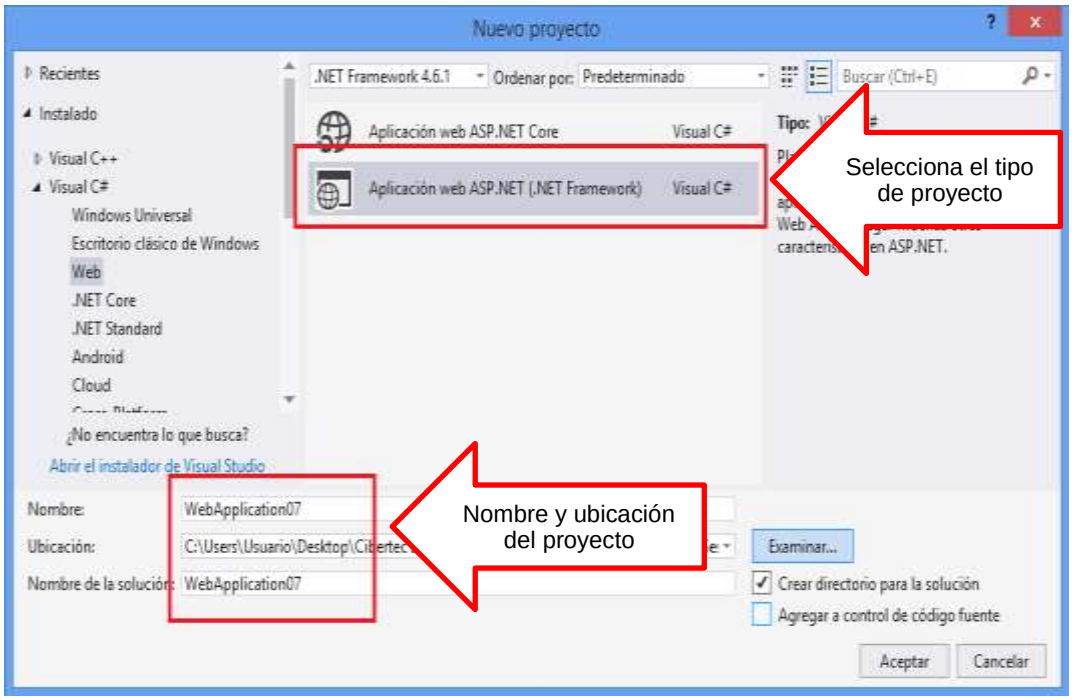
Creando la tabla tb_distritos

En el administrador del SQL Server, defina la tabla tb_distritos, tal como se muestra.

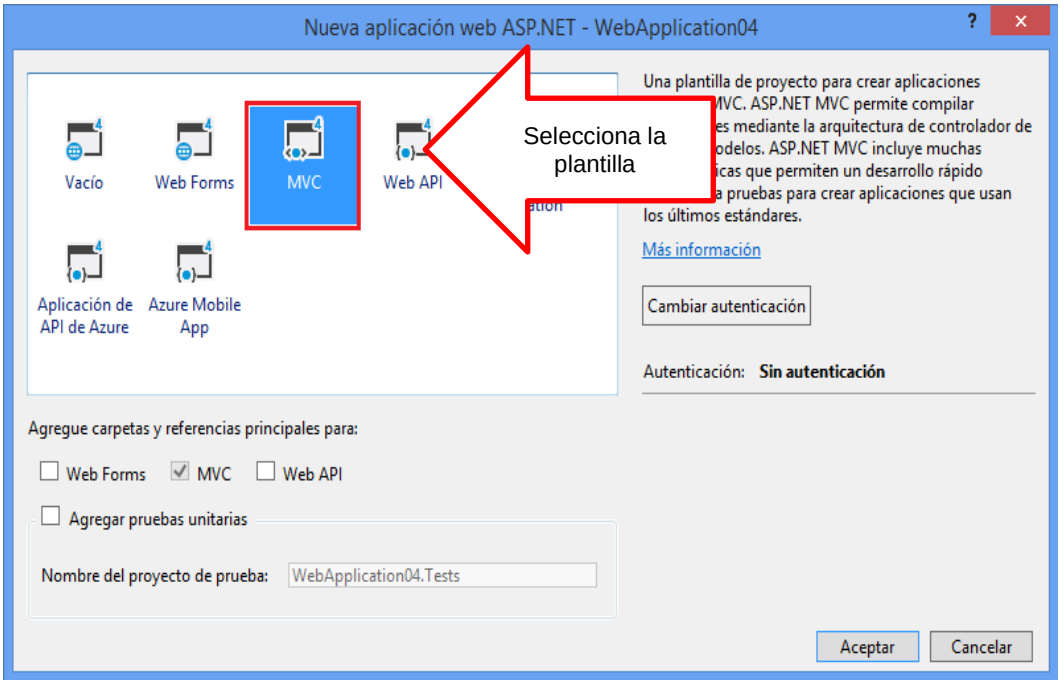


Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR

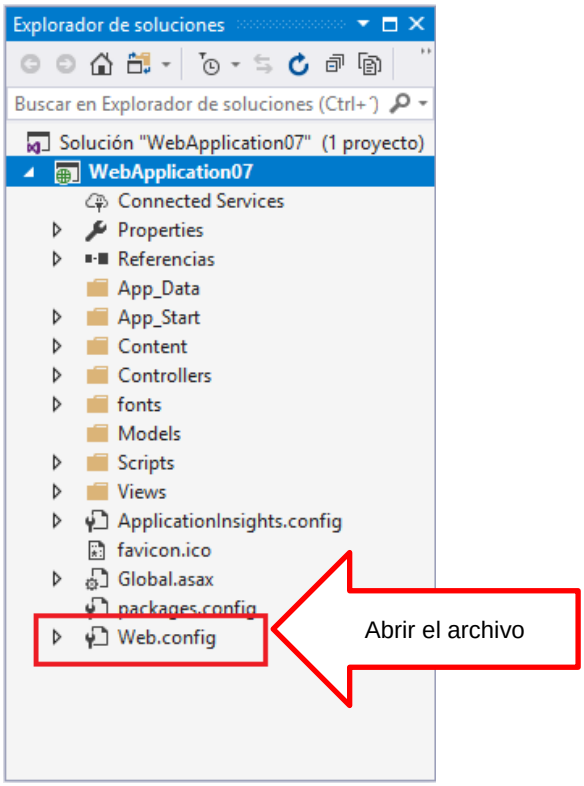


A continuación, seleccionar la plantilla del proyecto MVC.

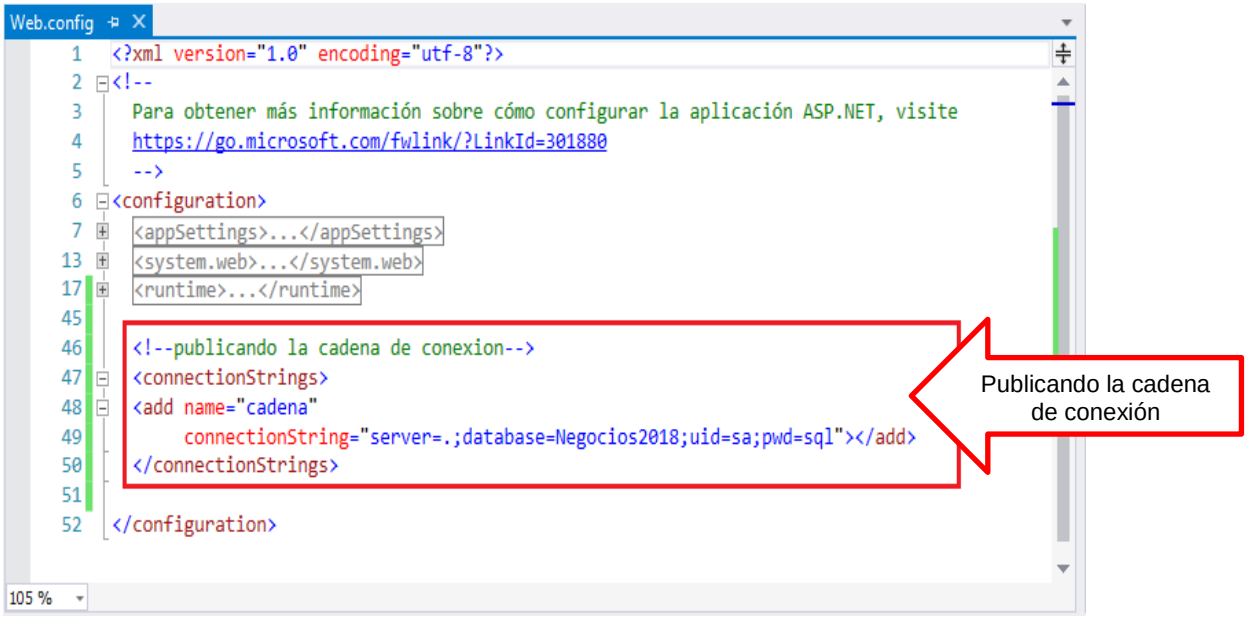


Publicando la cadena de conexión

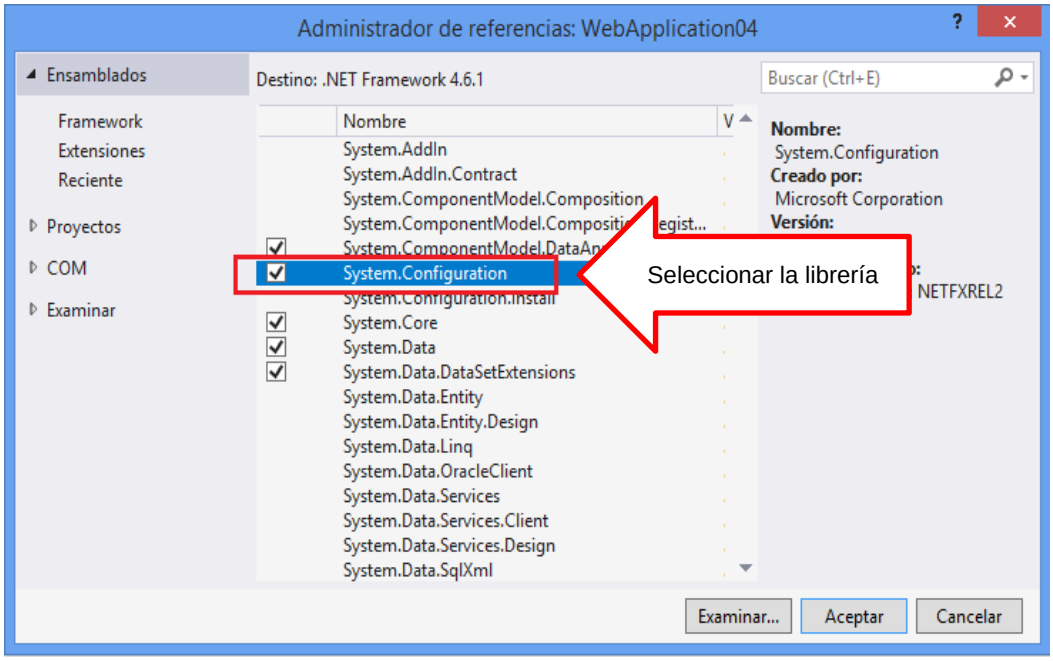
Abrir el archivo Web.config, tal como se muestra



Defina la etiqueta <connectionStrings> para agregar una cadena de conexión a la base de datos Negocios2018, tal como se muestra.

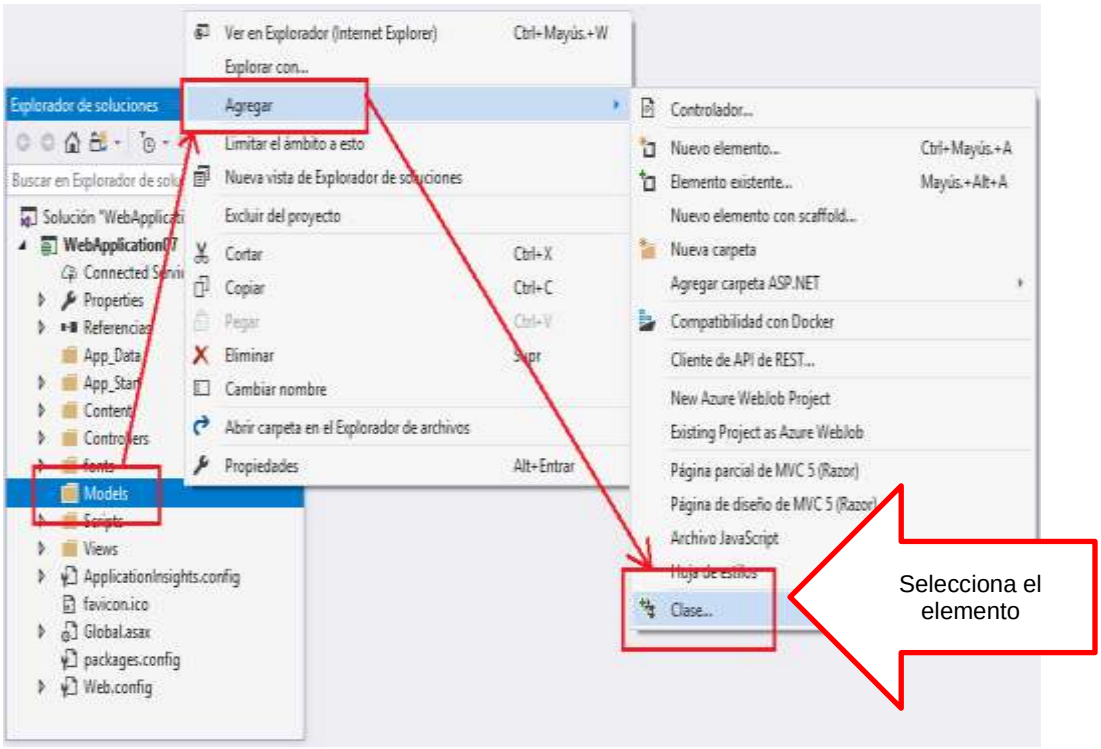


A continuación agregar la referencia al proyecto:
Selecciona desde la opción **Proyectos → Agregar Referencia**, selecciona la librería System.Configuration, tal como se muestra. Presiona el botón ACEPTAR.

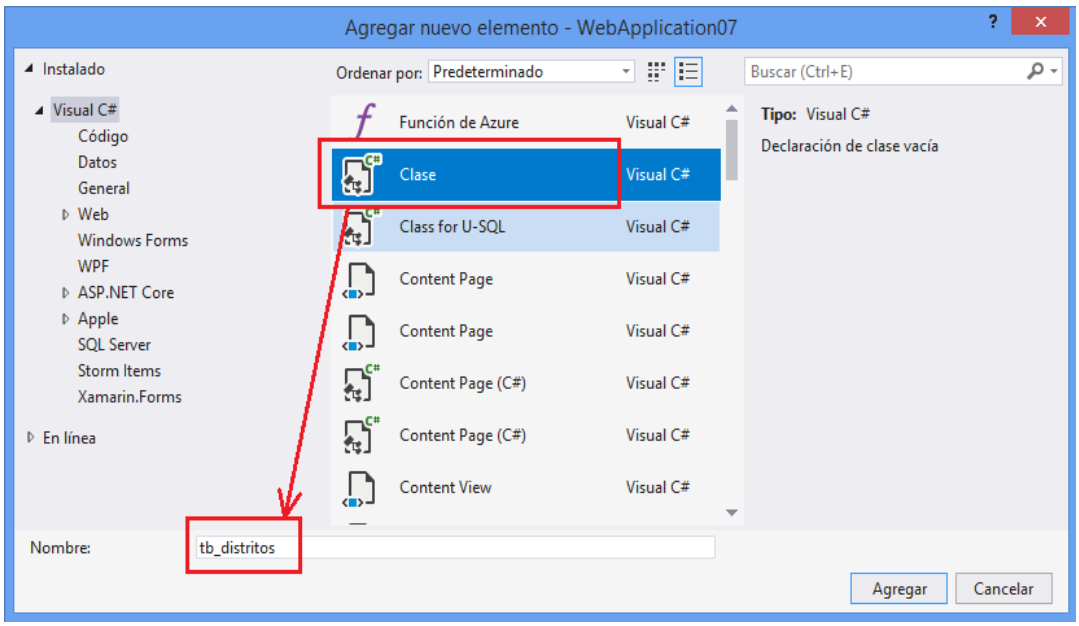


Creando la Clase del Modelo

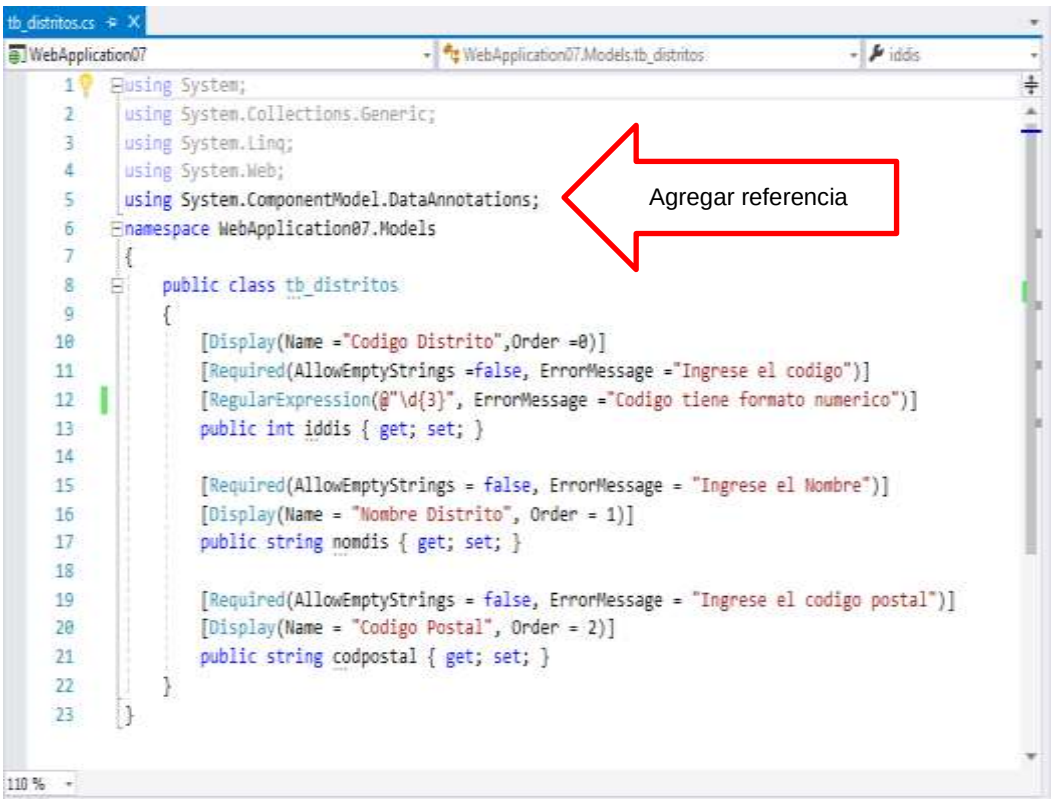
Para realizar las operaciones a la tabla tb_distritos, agregamos, en la carpeta Models, una clase, tal como se muestra.



Asigne el nombre de la clase, la cual se llamará tb_distritos, tal como se muestra. Presiona el botón AGREGAR

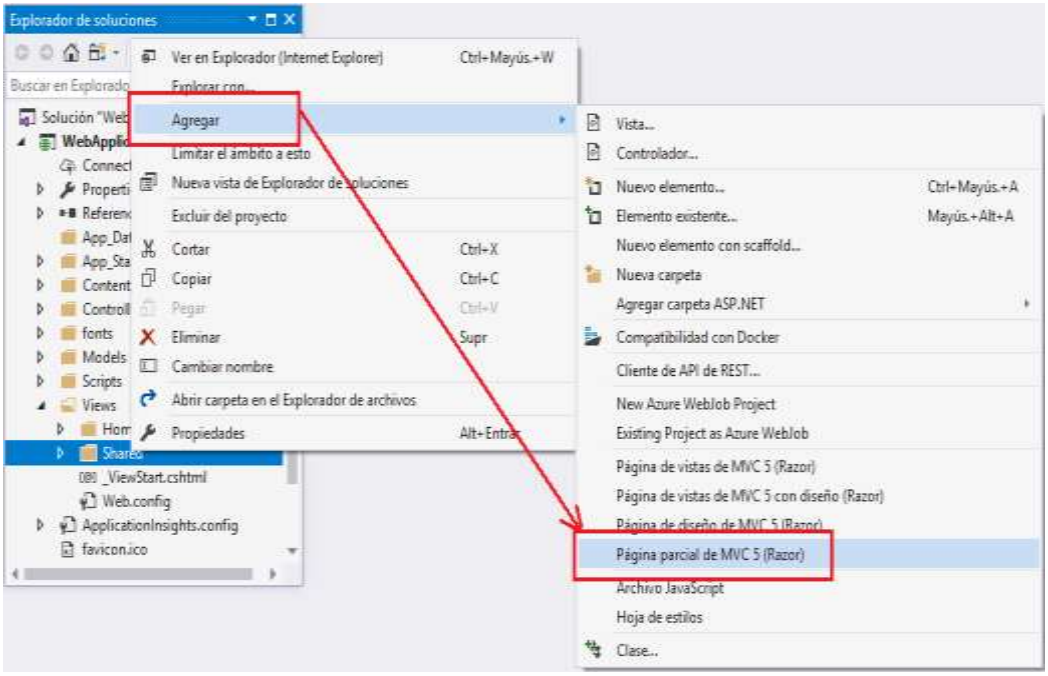


Para iniciar, agregamos la referencia DataAnnotations, luego defina los atributos de la clase, visualizando el texto y orden de cada uno de ellos [Display (Name="texto", Order="n")], tal como se muestra. Defina la validación para cada uno de los atributos.

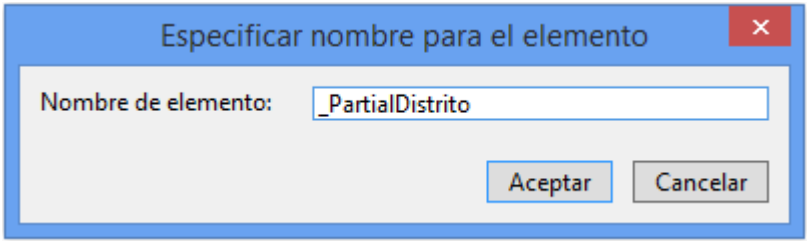


Agregando una Vista Parcial

Defina una Vista Parcial para listar de los registros de tb_distritos. En la carpeta **Shared**, selecciona la opción **AGREGAR → Página parcial de MVC 5 (Razor)**, tal como se muestra.

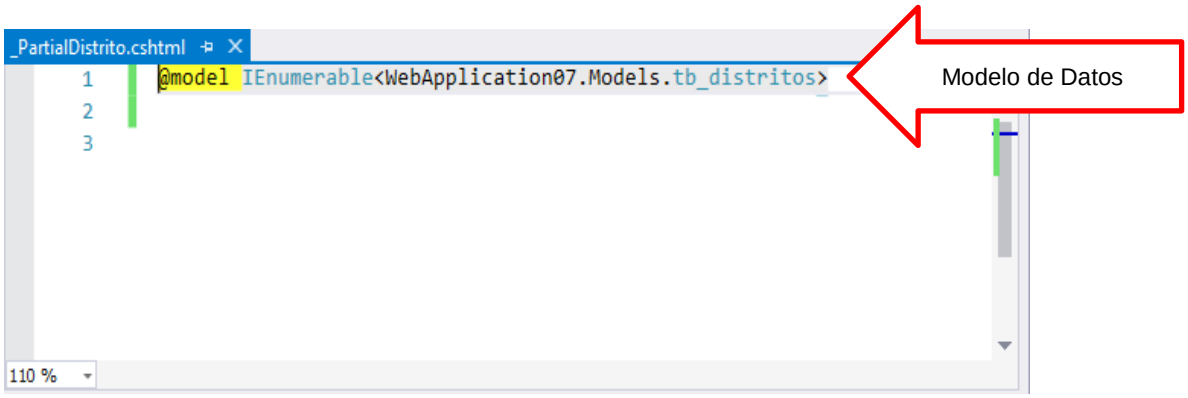


Asigne el nombre: **_PartialDistrito**, tal como se muestra

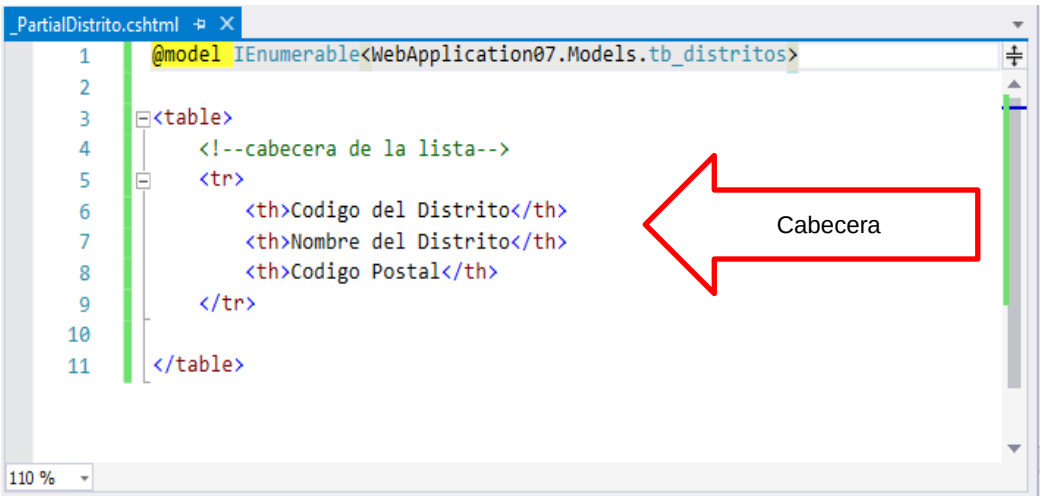


A continuación, codifique la vista parcial:

1. Defina la estructura de datos que recibirá la vista (IEnumerable de tb_distritos)



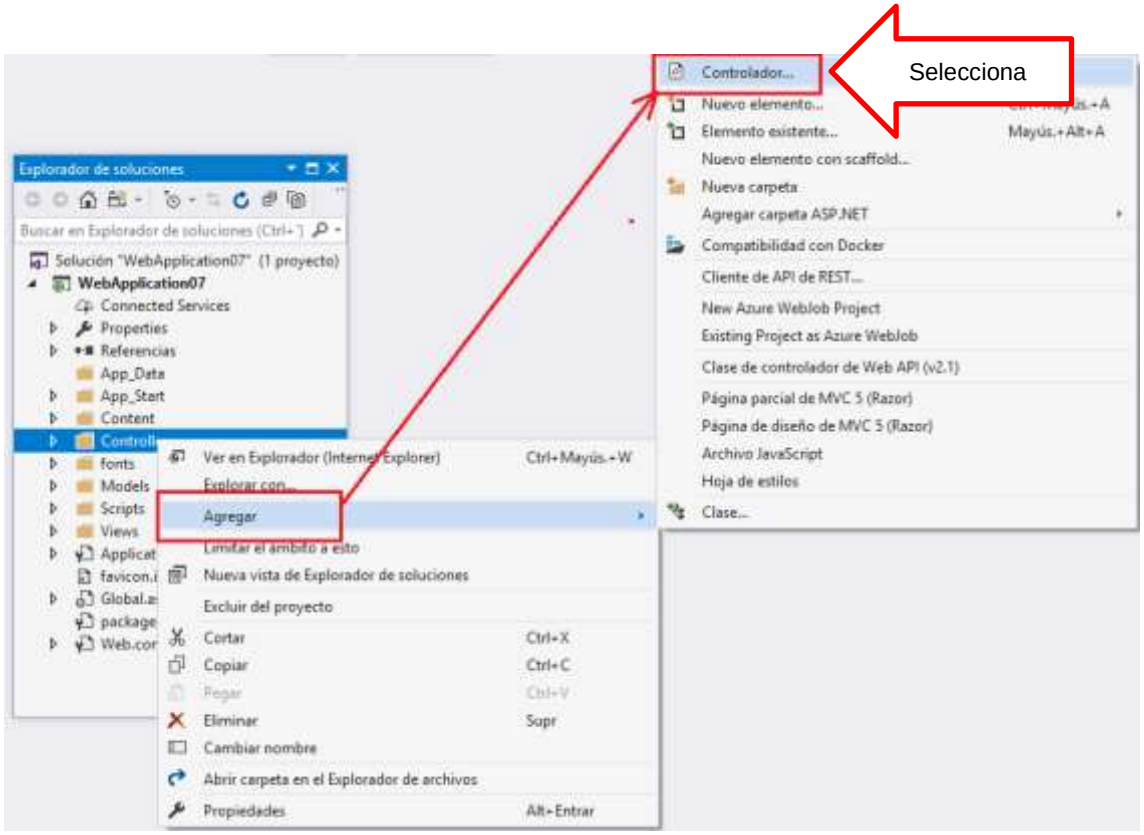
2. A continuación, definimos una etiqueta **<table>**, donde visualizamos la cabecera y los registros de la consulta. En esta primera parte, dentro del **<table>** definimos la cabecera, utilizando la etiqueta **<th>**, tal como se muestra



3. Definida la cabecera, vamos a definir la estructura para listar los registros. Dentro de un **foreach**, vamos a listar cada uno de los campos del registro de la consulta, tal como se muestra.

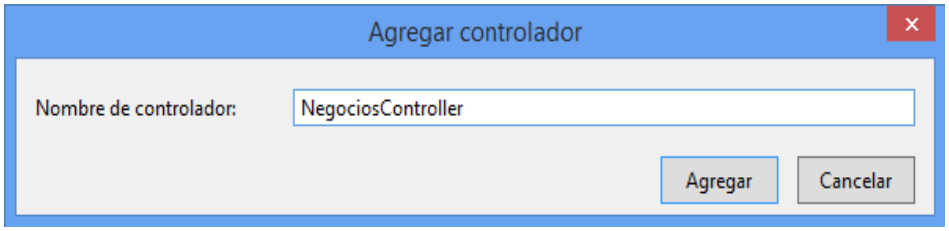
Creando un Controlador

A continuación, creamos un controlador: Desde la carpeta **Controllers**, selecciona la opción **Agregar**, **Controlador**…, tal como se muestra.

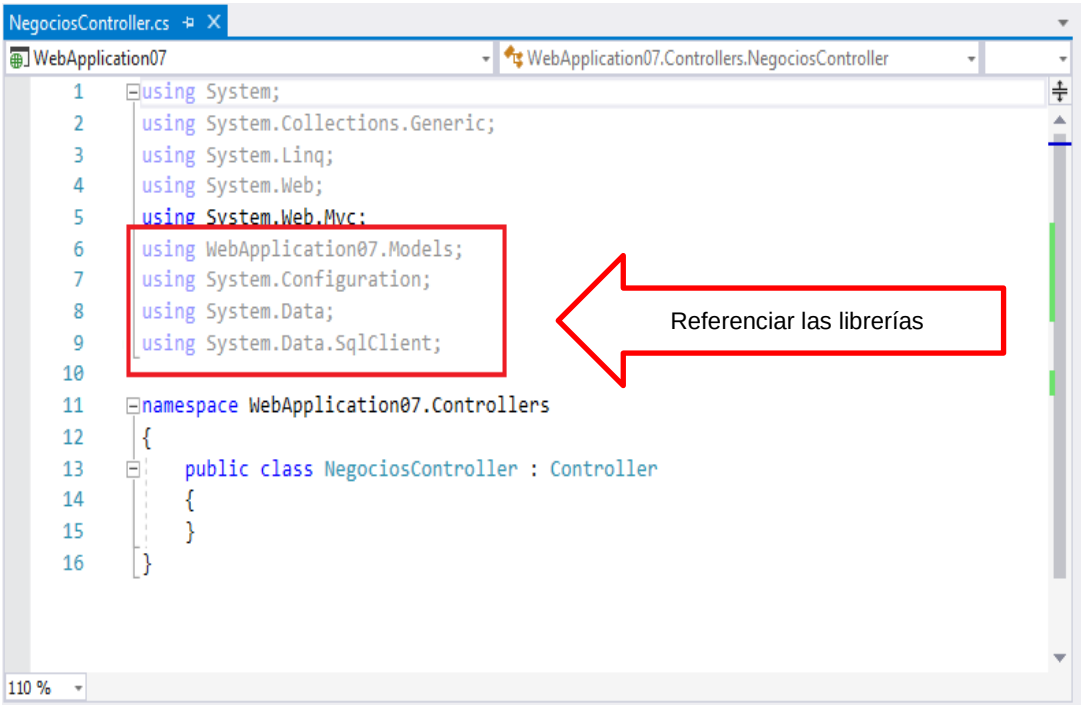


Selecciona el scaffold, Controlador de MVC5: en blanco, presiona el botón AGREGAR

Asigne el nombre del controlador: NegociosController, tal como se muestra



Referenciar la carpeta **Models**, la librería **Data**.**SqlClient** y el **System**.**Configuration** tal como se muestra.



Dentro del controlador, defina la conexión a la base de datos Negocios2018 instanciando el **SqlConnection**, utilice el **ConfigurationManager**.

Trabajando con el ActionResult Home

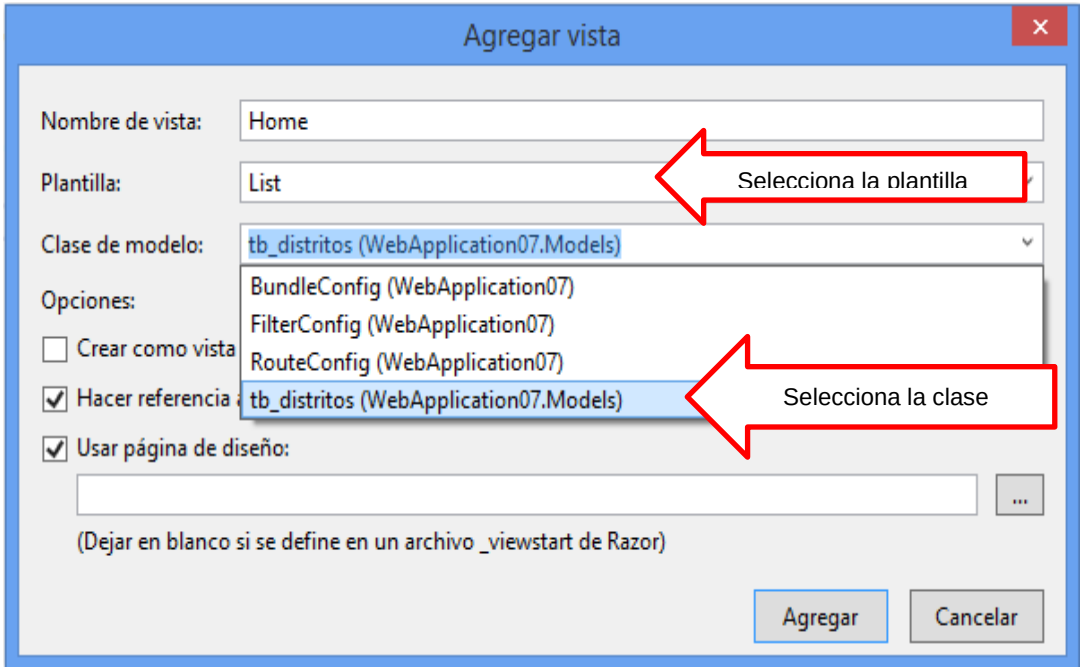
Codifique el método llamado Distritos():

1. Ejecuta la sentencia SQL que liste los registros de tb_distritos a través del SqlCommand. El resultado se almacena en el SqlDataReader.
2. Para guardar los resultados en el temporal se realiza a través de un bucle (mientras se pueda leer: dr.Read()), vamos almacenando cada registro en el temporal.
3. Cerrar los objetos y enviar el objeto lista llamada temporal.

Defina el ActionResult **Home**(), el cual enviará a la Vista el resultado del método **Distritos**(), tal como se muestra.

Agregando la Vista.

En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **List**; y la **clase de modelo** la cual es **tb_distritos**, tal como se muestra

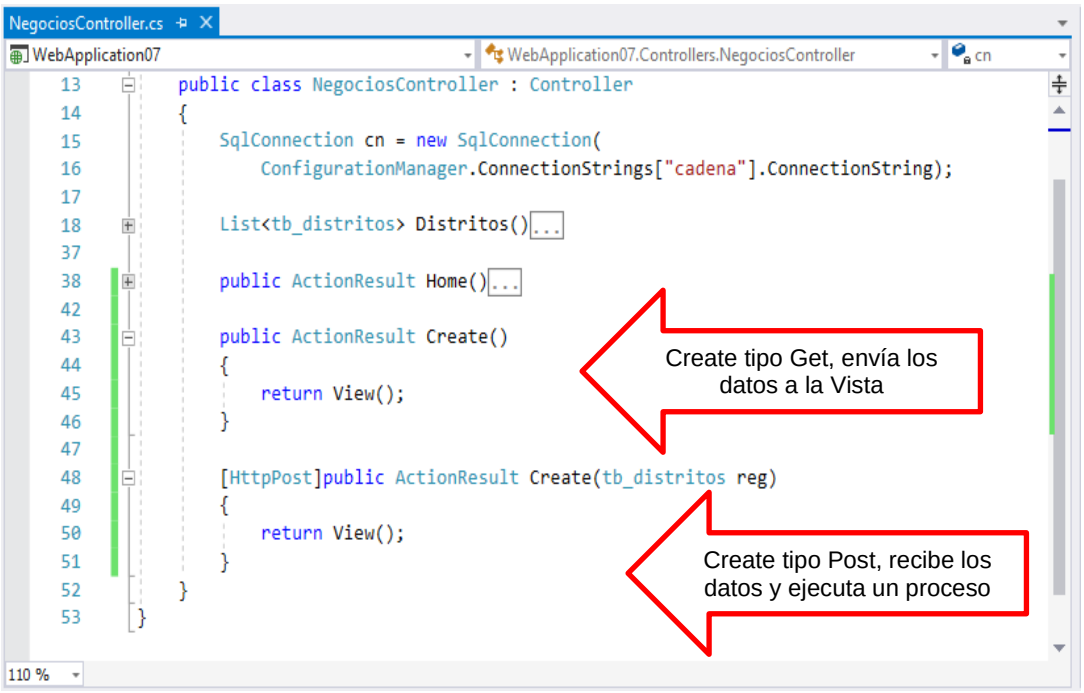


Modifica la Vista:
1. Agregar un ActionLink el cual ejecutará el ActionResult Create.
2. Agrega la vista parcial llamado _partialDistrito

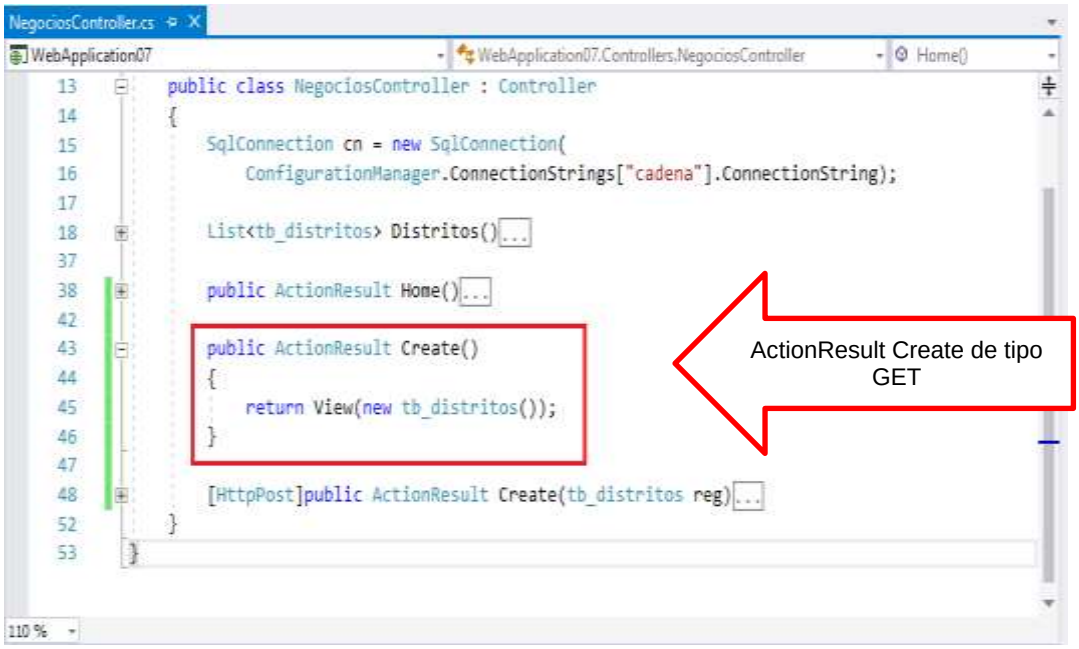
Ejecutamos la Vista, visualizamos los registros de la tabla tb_distritos.

Trabajando con el ActionResult Create

En el controlador Negocios, defina el ActionResult **Create**(), de tipo Get y Post, tal como se muestra.



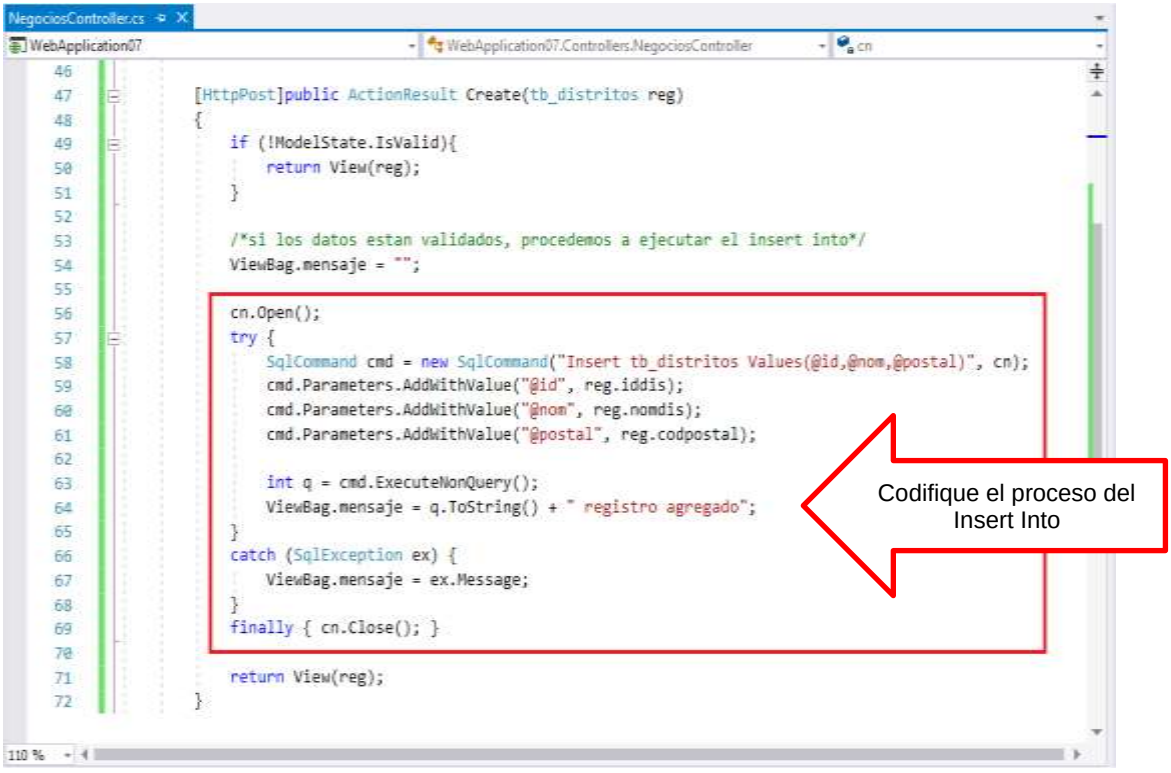
El **ActionResult** Create de tipo GET, envía a la vista un nuevo registro de tb_distritos, en blanco, tal como se muestra.



El **ActionResult** Create de tipo POST, recibe los datos de la vista, los valida; si está OK, procederá a ejecutar el comando INSERT INTO, retornando un mensaje y los datos del registro agregado, tal como se muestra


```
public class NegociosController : Controller
{
    SqlConnection cn = new SqlConnection(
        ConfigurationManager.ConnectionStrings["cadena"].ConnectionString);

    List<tb_distritos> Distritos()...
    public ActionResult Home()...

    public ActionResult Create()...

    [HttpPost]public ActionResult Create(tb_distritos reg)
    {
        if (!ModelState.IsValid){
            return View(reg);
        }

        /*si los datos estan validados, procedemos a ejecutar el insert into*/
        ViewBag.mensaje = "";

        try { }
        catch (SqlException ex) { }
        finally { }

        return View(reg);
    }
}
}
```

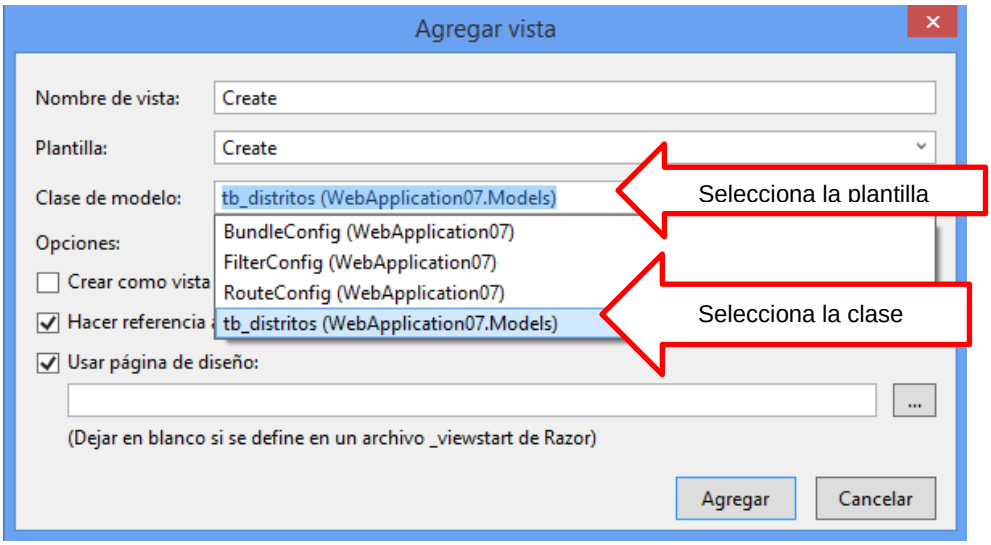
Codifique el proceso del Insert into en el ActionResult Create tipo POST



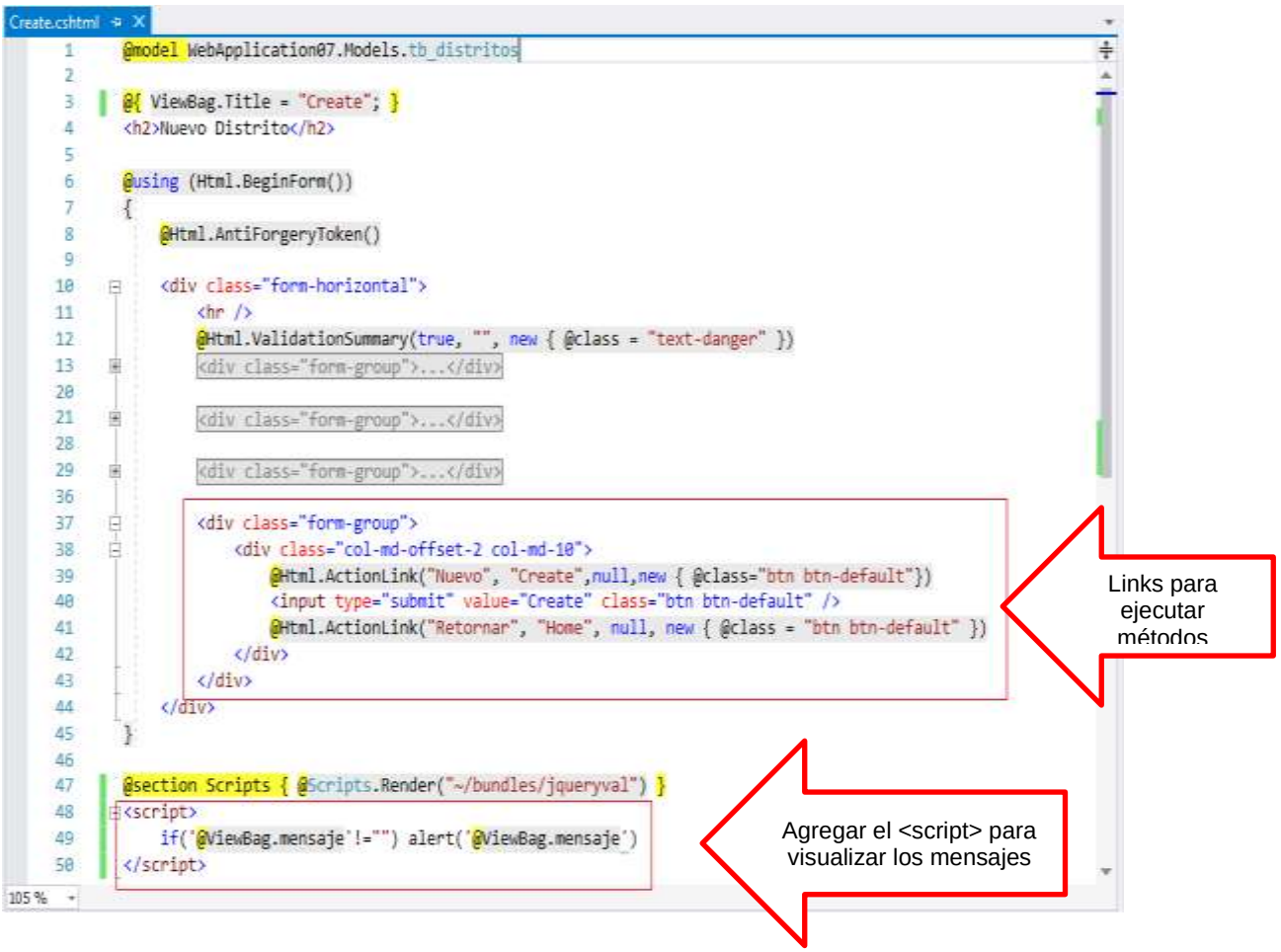

```
[HttpPost]public ActionResult Create(tb_distritos reg)
{
    if (!ModelState.IsValid){
        return View(reg);
    }

    /*si los datos estan validados, procedemos a ejecutar el insert into*/
    ViewBag.mensaje = "";

    cn.Open();
    try {
        SqlCommand cmd = new SqlCommand("Insert tb_distritos Values(@id,@nom,@postal)", cn);
        cmd.Parameters.AddWithValue("@id", reg.iddis);
        cmd.Parameters.AddWithValue("@nom", reg.nomdis);
        cmd.Parameters.AddWithValue("@postal", reg.codpostal);

        int q = cmd.ExecuteNonQuery();
        ViewBag.mensaje = q.ToString() + " registro agregado";
    }
    catch (SqlException ex) {
        ViewBag.mensaje = ex.Message;
    }
    finally { cn.Close(); }

    return View(reg);
}
```

Agregando la Vista.

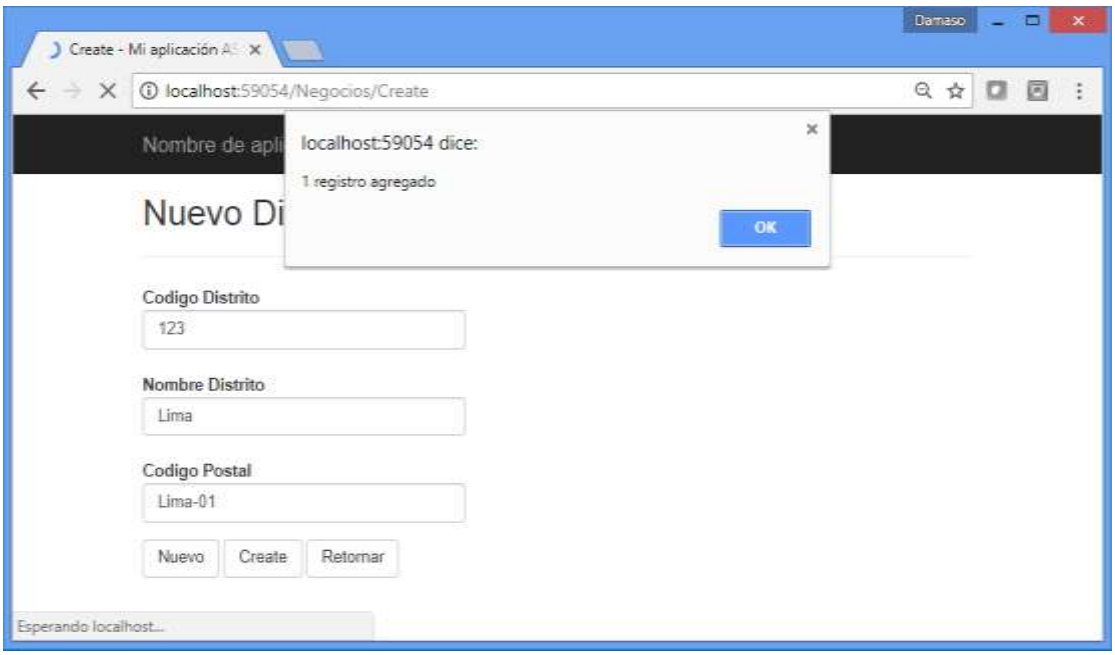
En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **Create**; y la **clase de modelo** la cual es **tb_distritos**, tal como se muestra



Modifique el diseño de la vista, tal como se muestra

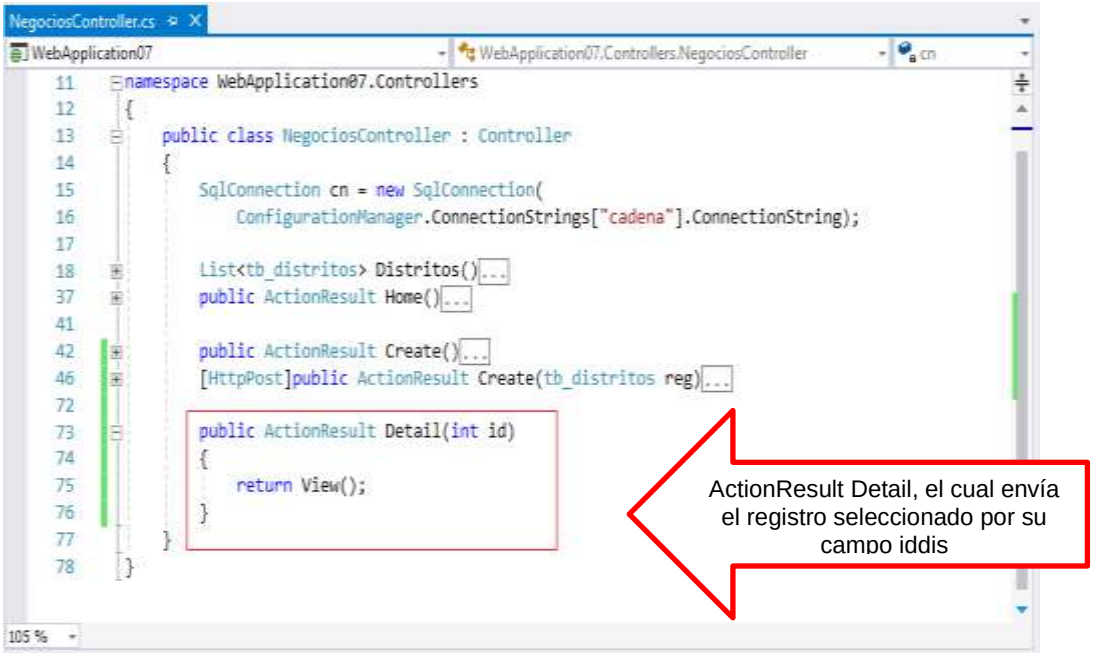


Ejecuta la Vista, ingresa los datos, al presionar el botón CREATE, agregamos un registro y visualizamos un mensaje.
Para retornar, presione el botón RETORNAR



Trabajando con el ActionResult Detail

En el controlador Negocios, defina el ActionResult **Detail**(int id) el cual envía los datos de un distrito seleccionado por su campo iddis, tal como se muestra.

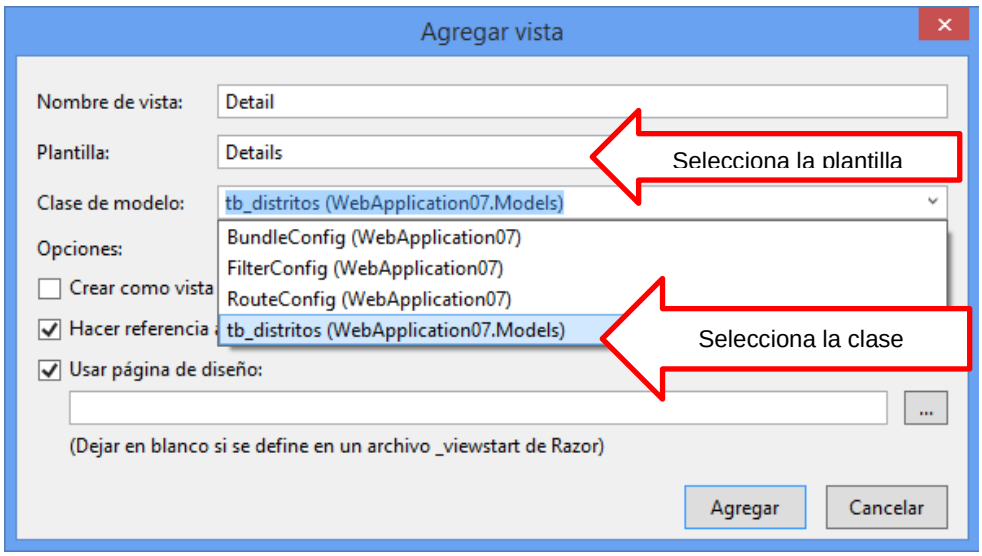


Codifique el ActionResult:
1. Filtre el método Distritos(), buscando por su campo iddis, retornando el registro de la búsqueda.
2. Retornar el registro a la Vista().

```
NegociosController.cs
WebApplication07 | WebApplication07.Controllers.NegociosController | cn
11  namespace WebApplication07.Controllers
12  {
13      public class NegociosController : Controller
14      {
15          SqlConnection cn = new SqlConnection(
16              ConfigurationManager.ConnectionStrings["cadena"].ConnectionString);
17
18          List<tb_distritos> Distritos()...
37          public ActionResult Home()...
41
42          public ActionResult Create()...
46          [HttpPost]public ActionResult Create(tb_distritos reg)...
72
73          public ActionResult Detail(int id)
74          {
75              tb_distritos reg = Distritos().Where(d => d.iddis == id).FirstOrDefault();
76
77              return View(reg);
78          }
79      }
80  }
105 %
```

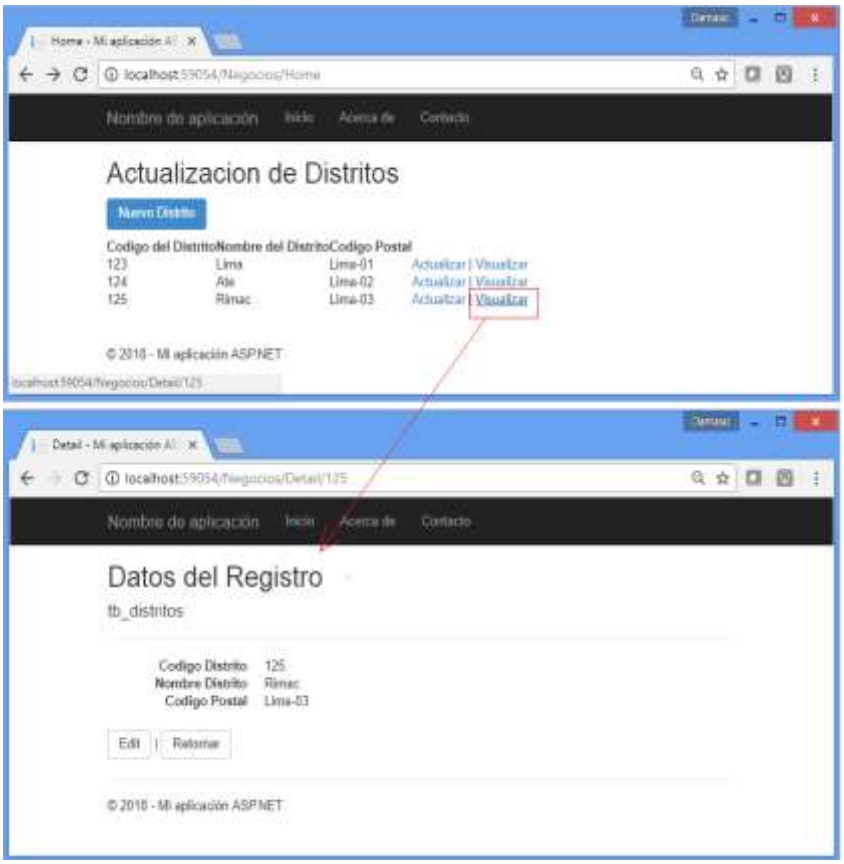
Agregando la Vista.

En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **Details**; y la **clase de modelo** la cual es **tb_distritos**, tal como se muestra



Modifique el código de la Vista tal como se muestra.

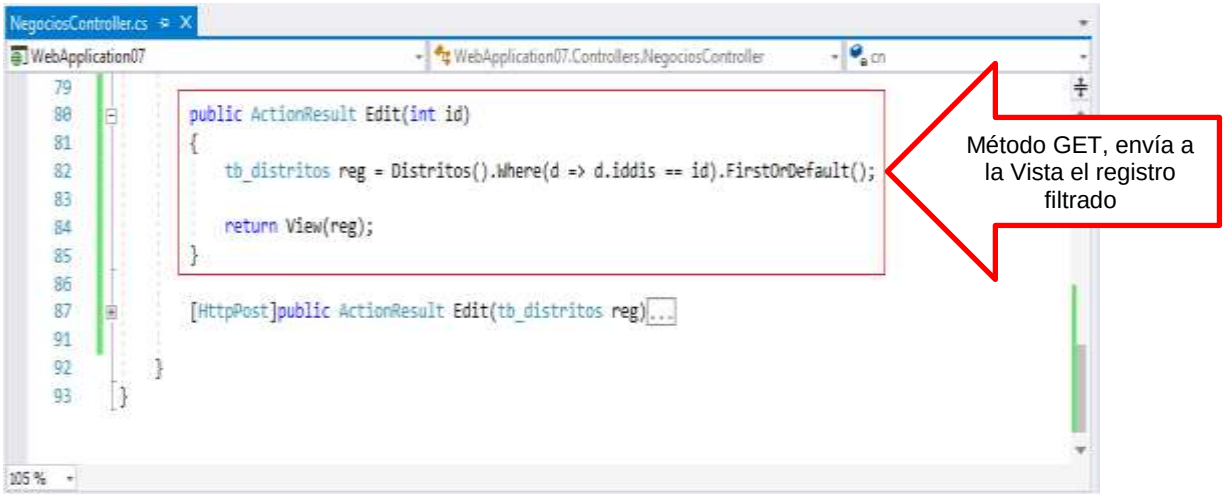
Ejecuta la Vista Home, al hacer click en el link “Visualizar”, direccionamos a la vista Detail, visualizando los datos del distrito seleccionado



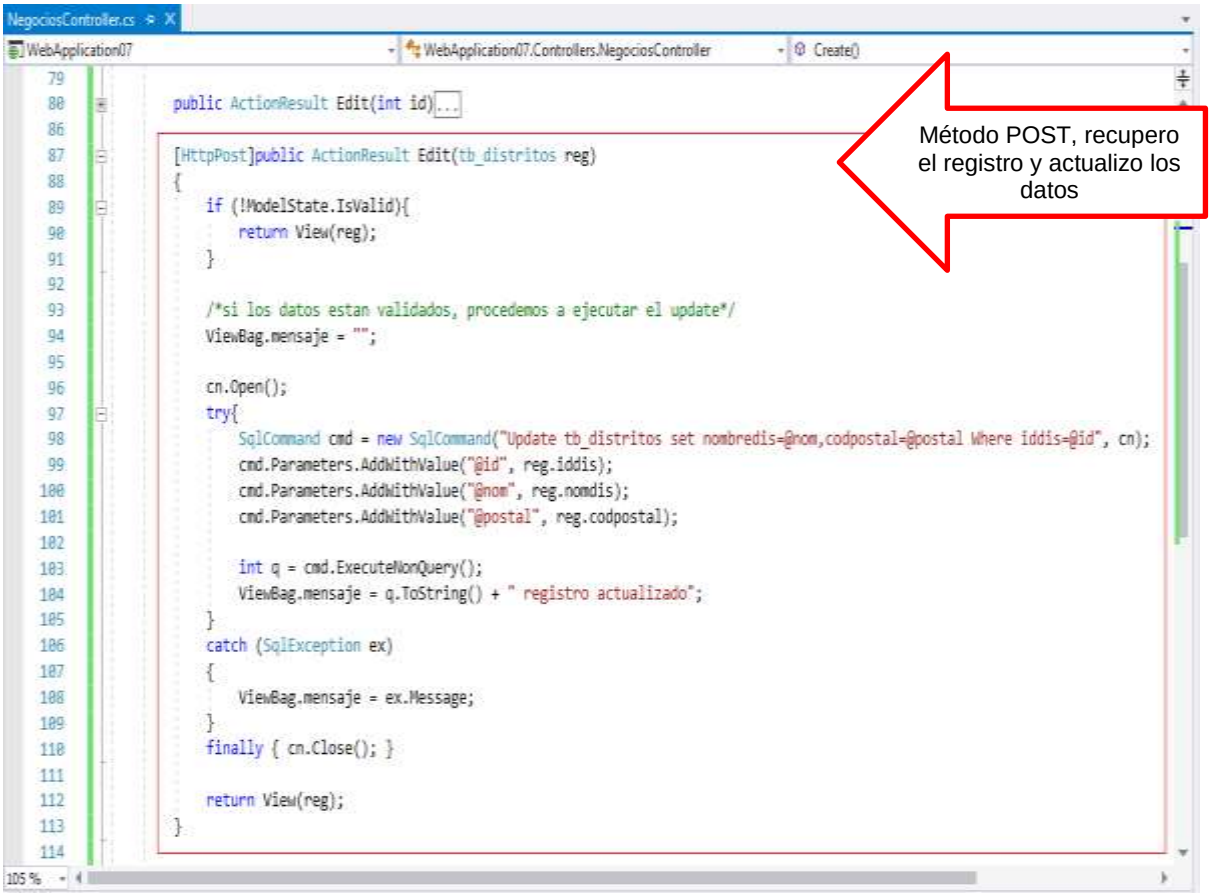
Trabajando con el ActionResult Edit

En el controlador Negocios, defina el ActionResult **Edit** GET y POST, tal como se muestra

Codifique el **ActionResult Edit** (int id) método GET, donde filtra y envía el registro de la tabla tb_distritos filtrado por su campo iddis, tal como se muestra



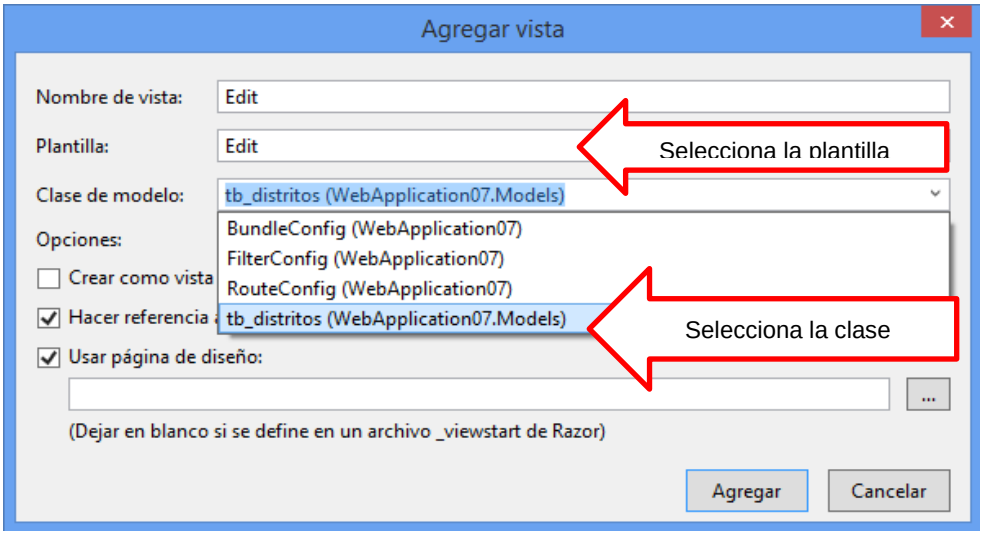
El **ActionResult** Edit de tipo POST, recibe los datos de la vista, los valida; si está OK, procederá a ejecutar el comando UPDATE, retornando un mensaje y los datos del registro agregado, tal como se muestra



Agregando la Vista.

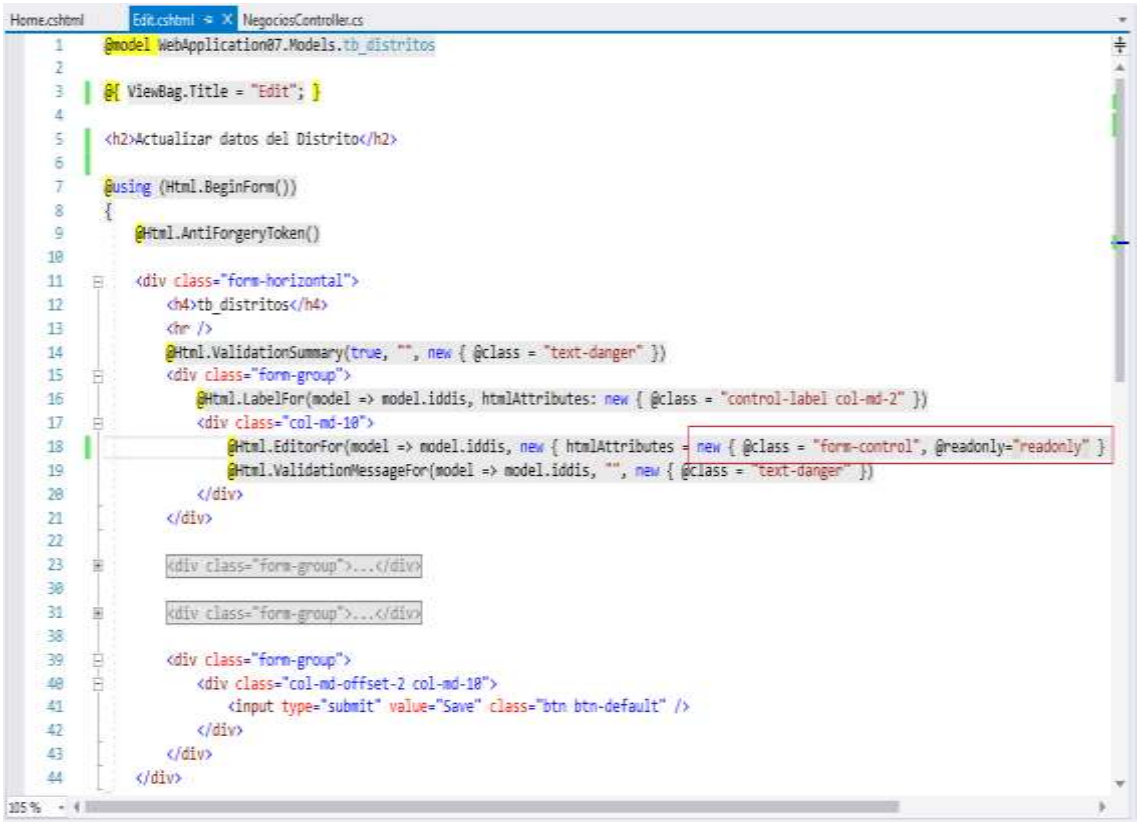
En el ActionResult, agrega la vista.

En dicha ventana, selecciona la **plantilla**, la cual será **Edit**; y la **clase de modelo** la cual es **tb_distritos**, tal como se muestra



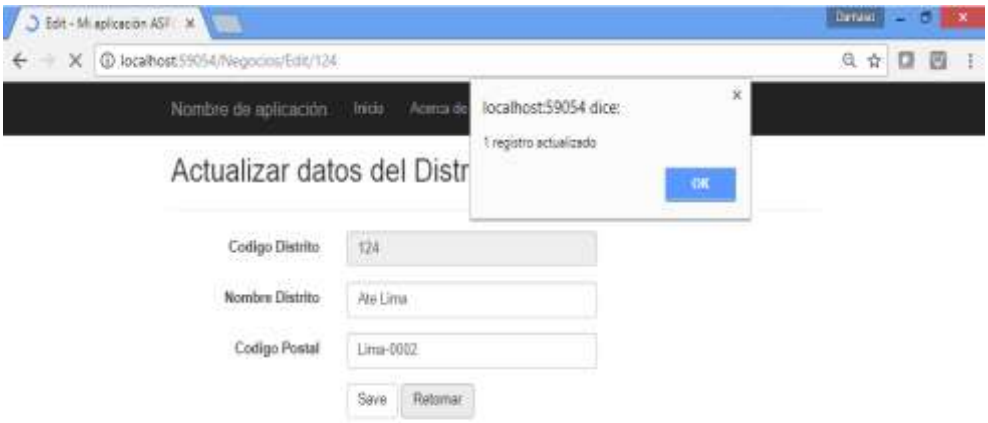
Modifique el diseño de la Vista

Para que el control EditFor del iddis sea solo lectura, agregar la propiedad @readonly=”readonly”, tal como se muestra



Modifique el diseño del ActionLink Retornar, tal como se muestra

Ejecuta la vista, selecciona el Distrito, donde visualizamos los datos del registro en la vista Edit.
Modifique los datos, al presionar el botón SAVE, se visualiza un mensaje indicando que se ha actualizado el registro.

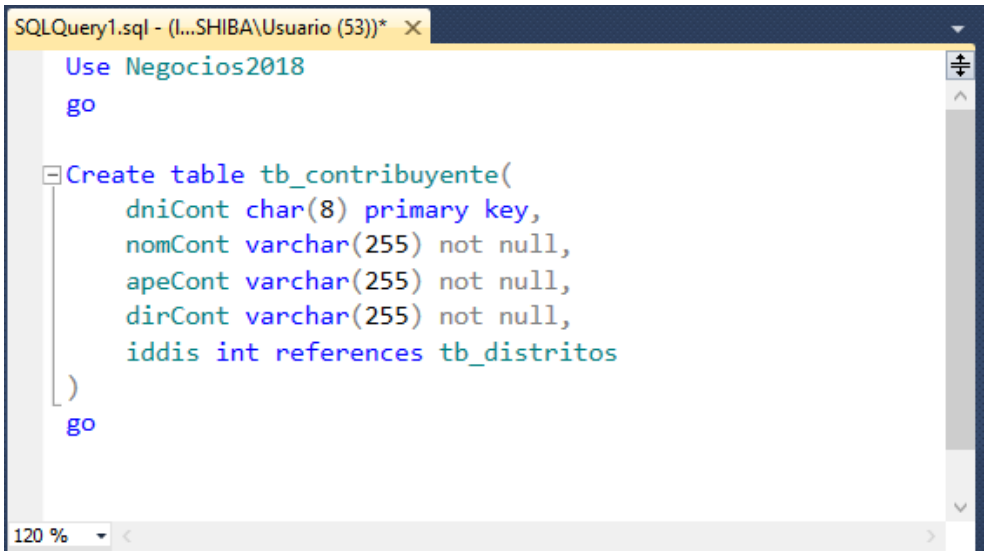


Laboratorio 5.2
Manipulación de Datos - Transacciones

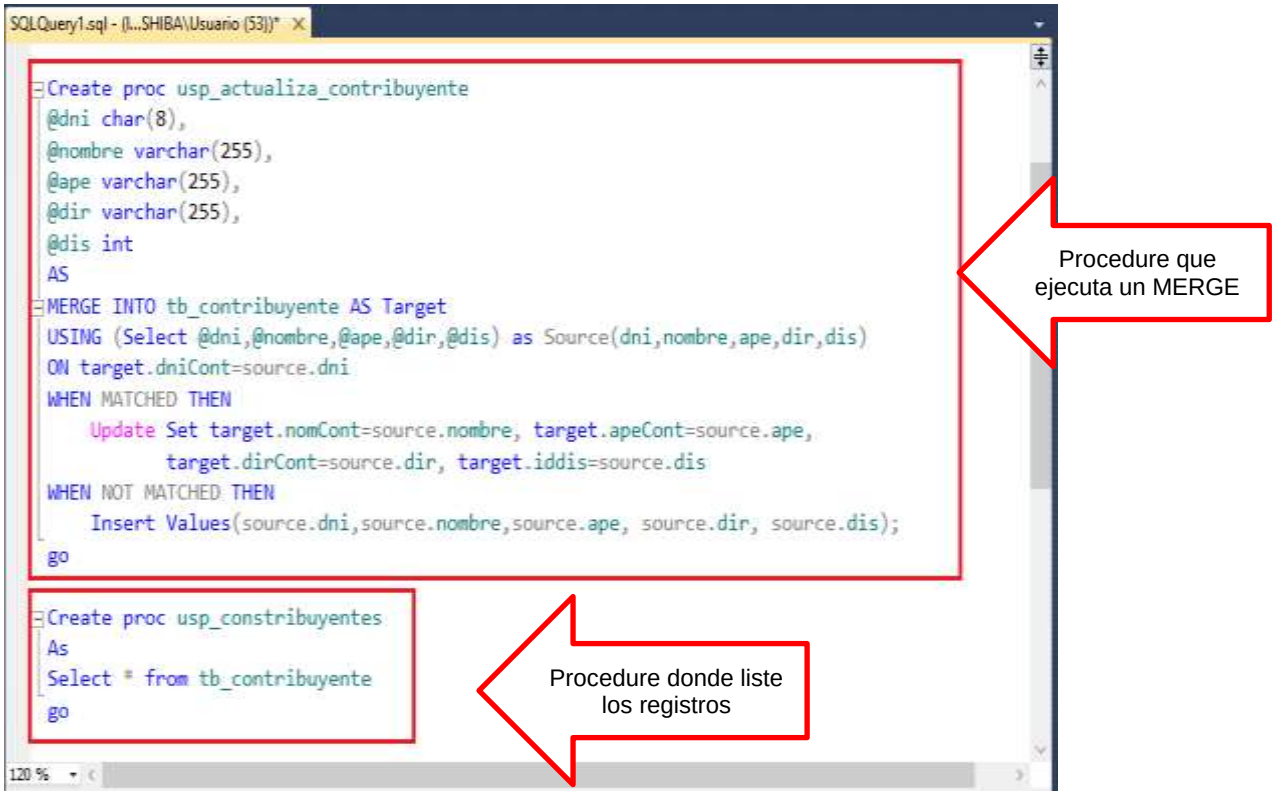
Implemente un proyecto ASP.NET MVC, donde permita INSERTAR, CONSULTAR y ACTUALIZAR los datos de la tabla tb_contribuyente.

Creando la tabla tb_contribuyente

En el administrador del SQL Server, defina la tabla tb_contribuyente, tal como se muestra.

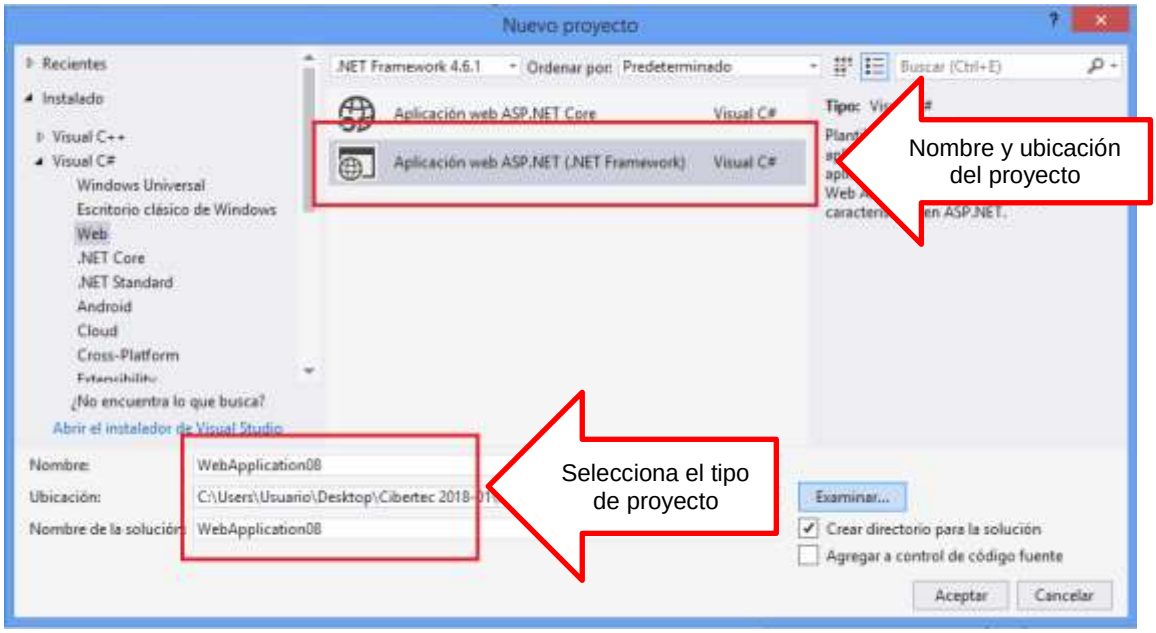


Crea los procedimientos almacenados: usp_actualiza_contribuyente, donde inserta o actualiza un registro (uso del MERGE), usp_contribuyentes, lista los registros

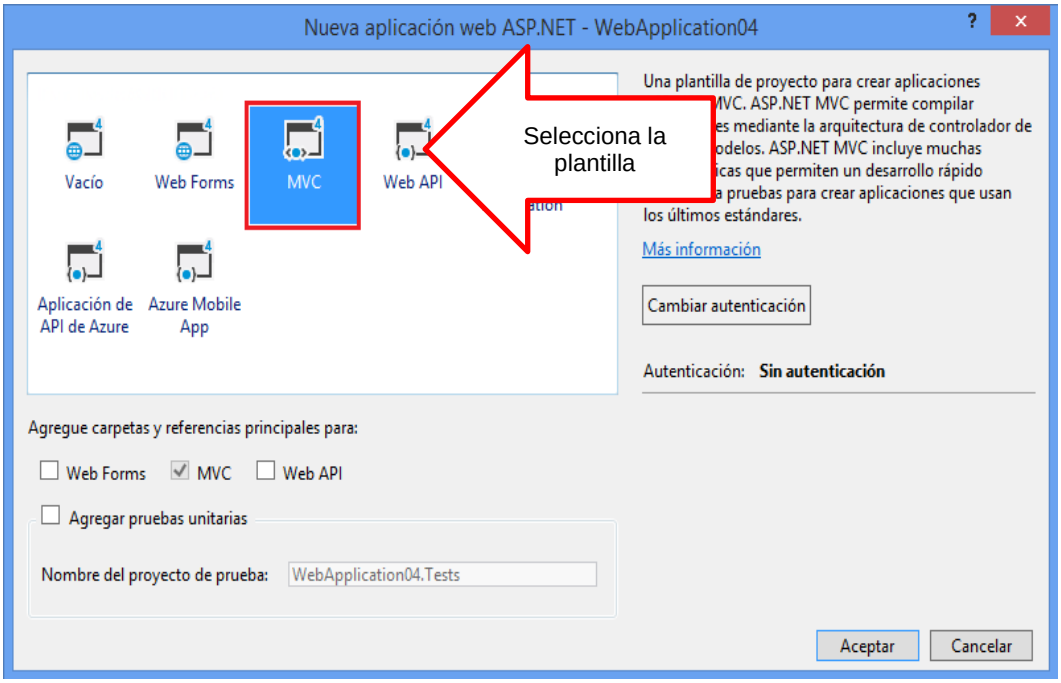


Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR



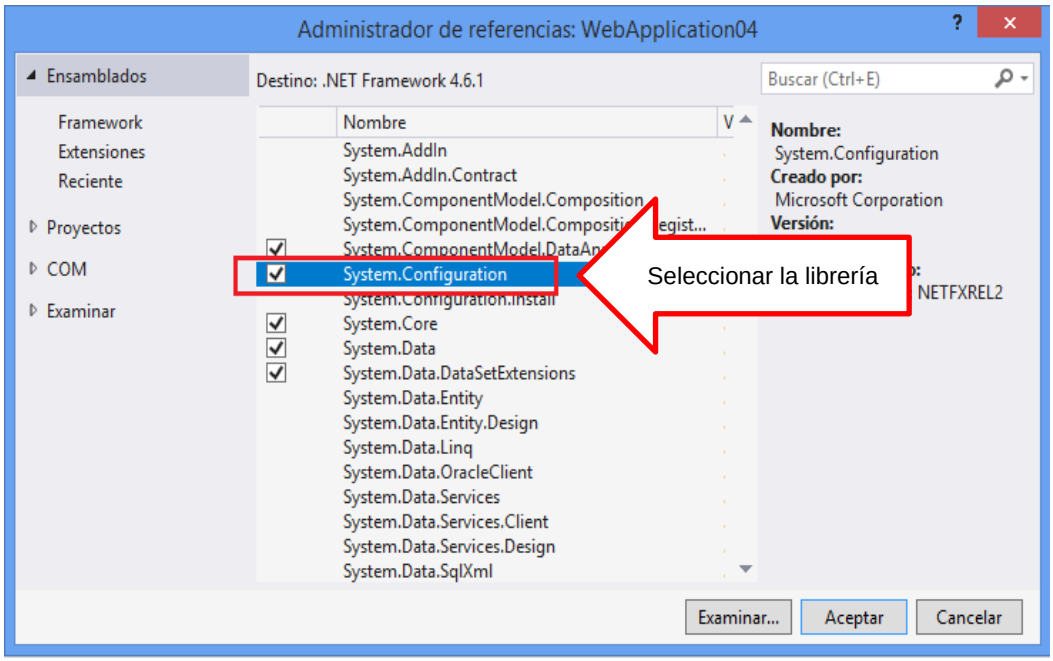
A continuación, seleccionar la plantilla del proyecto MVC.



Publicando la cadena de conexión

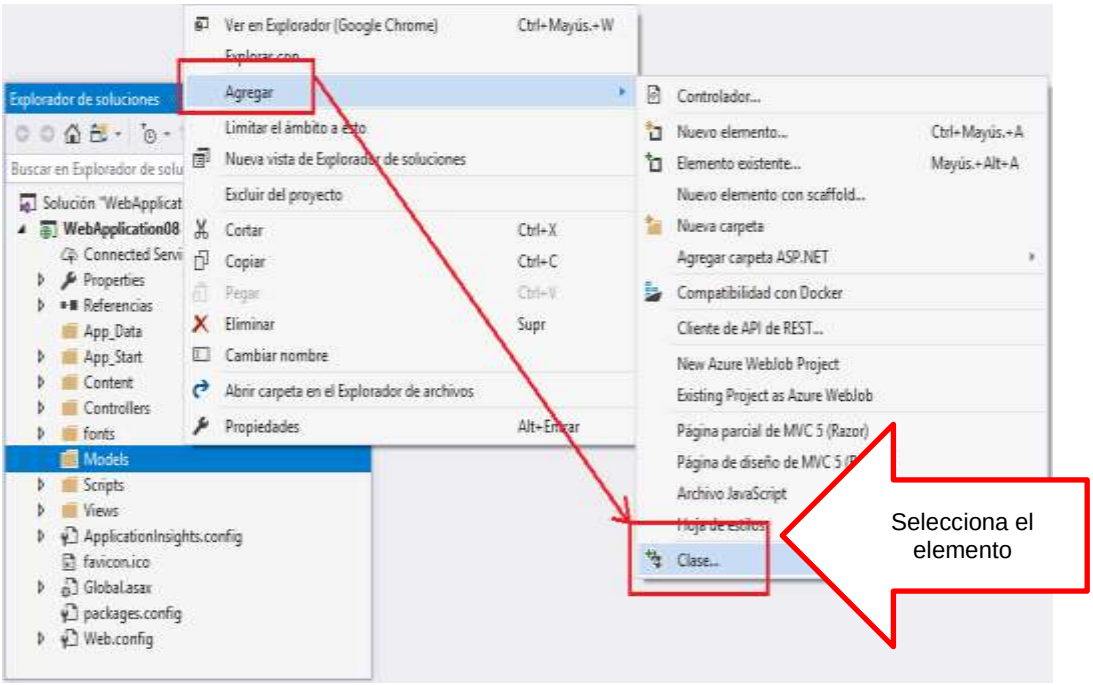
Abrir el archivo Web.config, defina la etiqueta <connectionStrings> para agregar una cadena de conexión a la base de datos Negocios2018, tal como se muestra.

A continuación agregar la referencia al proyecto:
Selecciona desde la opción **Proyectos → Agregar Referencia**, selecciona la librería System.Configuration, tal como se muestra. Presiona el botón ACEPTAR.

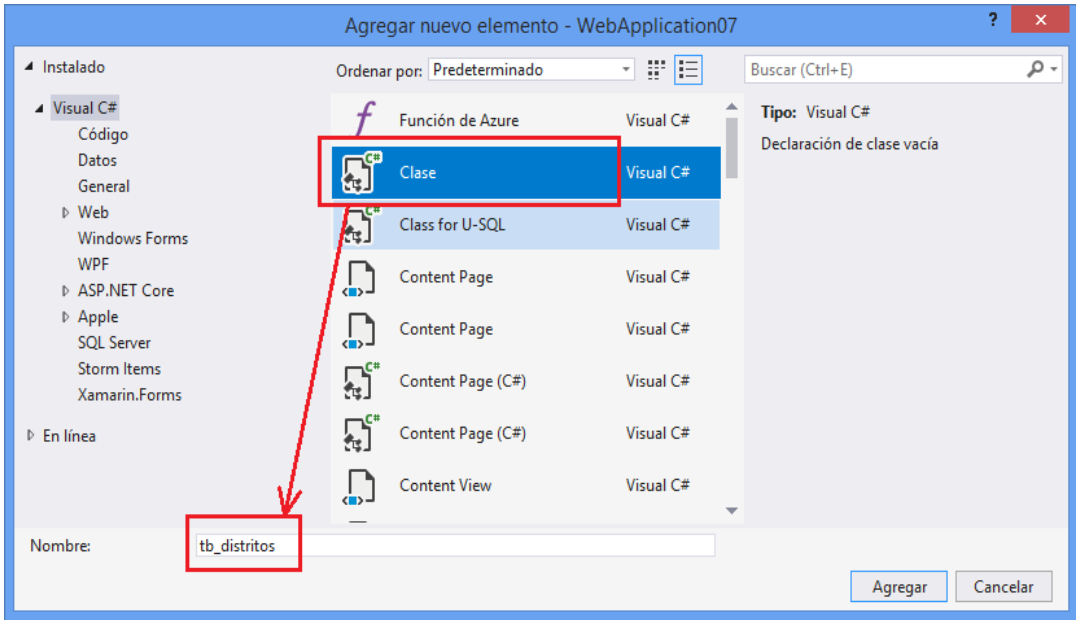


Creando las Clases del Modelo

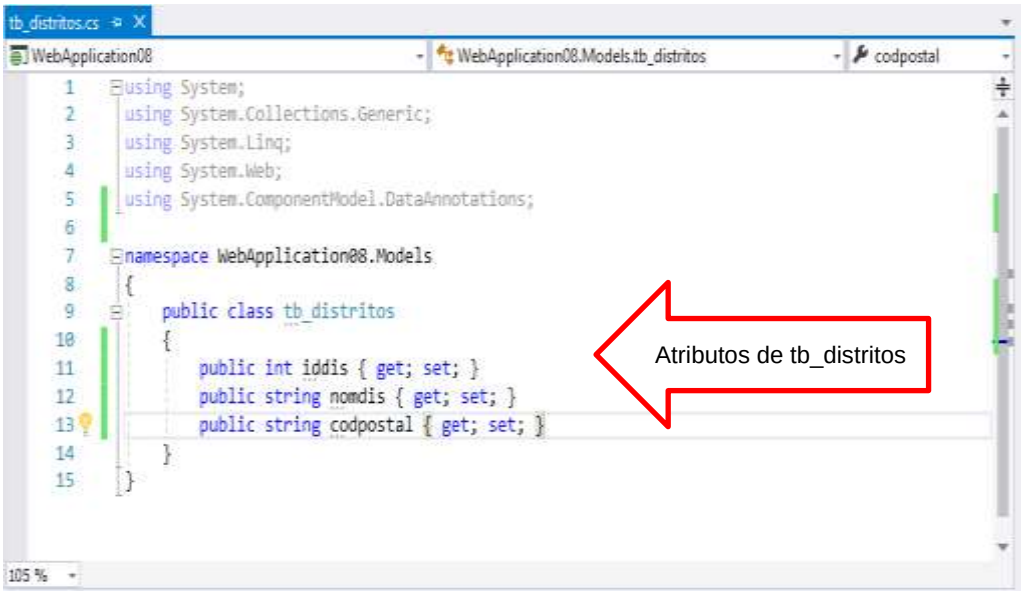
Para realizar las operaciones a la tabla tb_contribuyente y tb_distritos, agregamos, en la carpeta Models, las clases, tal como se muestra.



Asigne el nombre de la clase, la cual se llamará tb_distritos, tal como se muestra. Presiona el botón AGREGAR

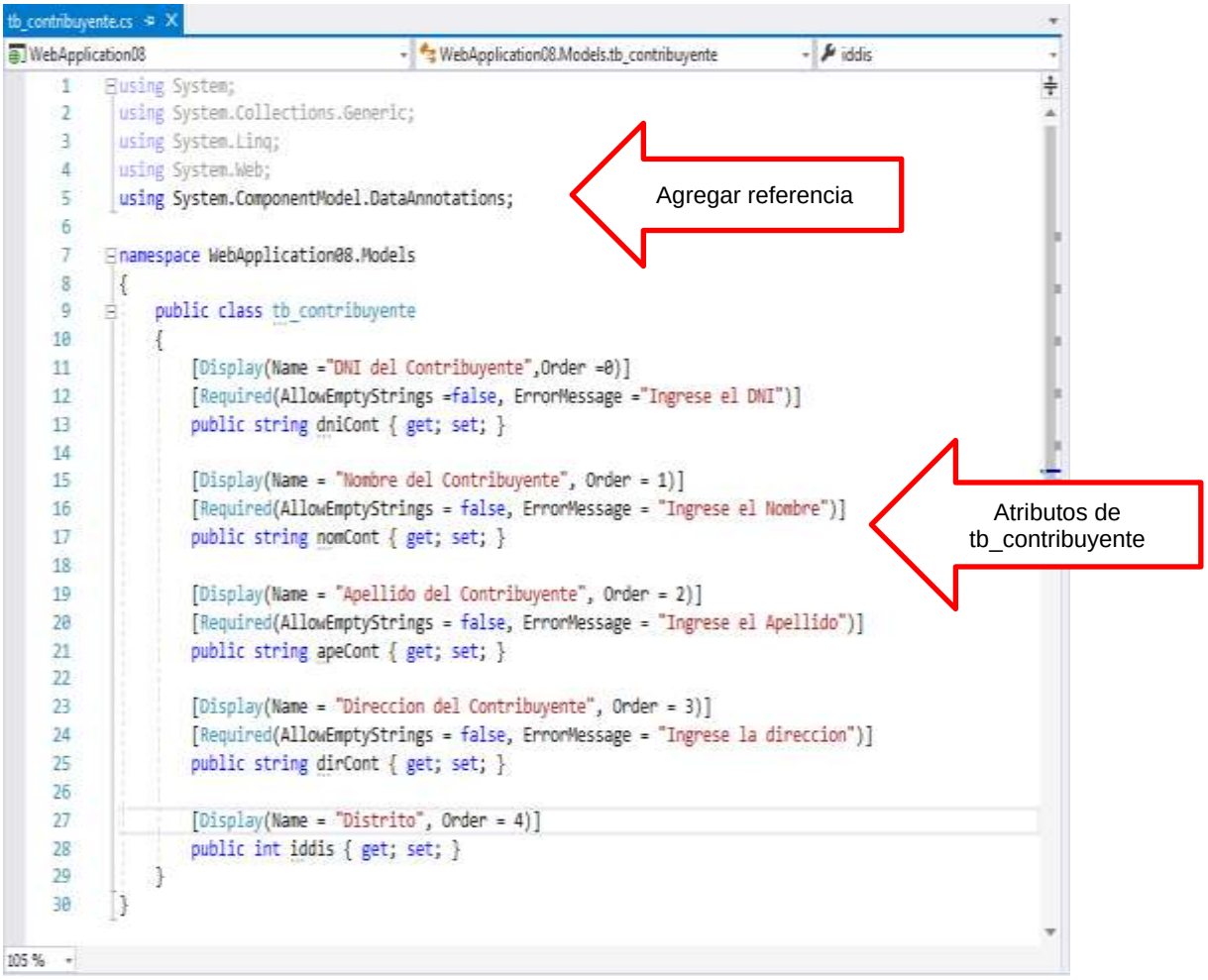


Defina los atributos de la clase tb_distritos, tal como se muestra.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace WebApplication08.Models
{
    public class tb_distritos
    {
        public int iddis { get; set; }
        public string nomdis { get; set; }
        public string codpostal { get; set; }
    }
}
```

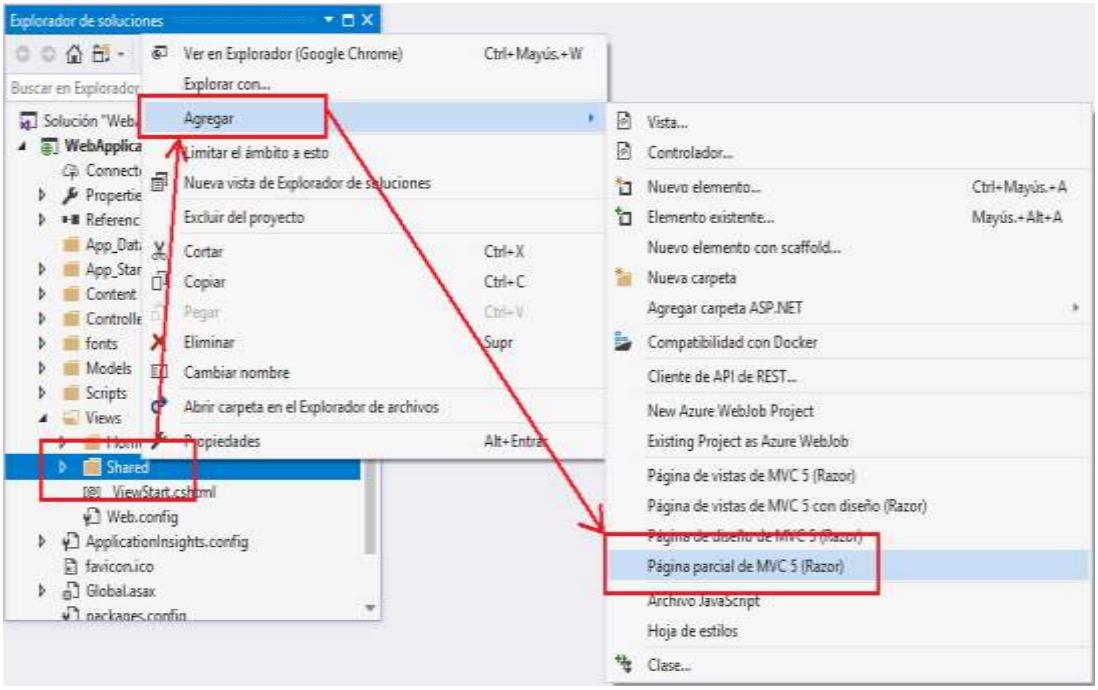
Defina los atributos de la clase tb_contribuyente, asi como los atributos [Display] y [Required] para cada uno de los atributos



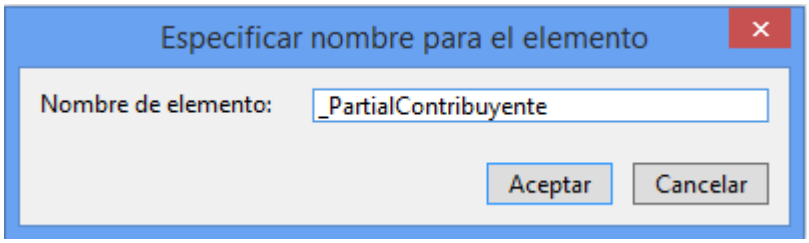

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace WebApplication08.Models
{
    public class tb_contribuyente
    {
        [Display(Name ="DNI del Contribuyente",Order =0)]
        [Required(AllowEmptyStrings =false, ErrorMessage ="Ingrese el DNI")]
        public string dniCont { get; set; }

        [Display(Name = "Nombre del Contribuyente", Order = 1)]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Nombre")]
        public string nomCont { get; set; }

        [Display(Name = "Apellido del Contribuyente", Order = 2)]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Apellido")]
        public string apeCont { get; set; }

        [Display(Name = "Direccion del Contribuyente", Order = 3)]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la direccion")]
        public string dirCont { get; set; }

        [Display(Name = "Distrito", Order = 4)]
        public int iddis { get; set; }
    }
}
```

Agregando una Vista Parcial

Defina una Vista Parcial para listar de los registros de tb_distritos. En la carpeta **Shared**, selecciona la opción **AGREGAR → Página parcial de MVC 5 (Razor)**, tal como se muestra.

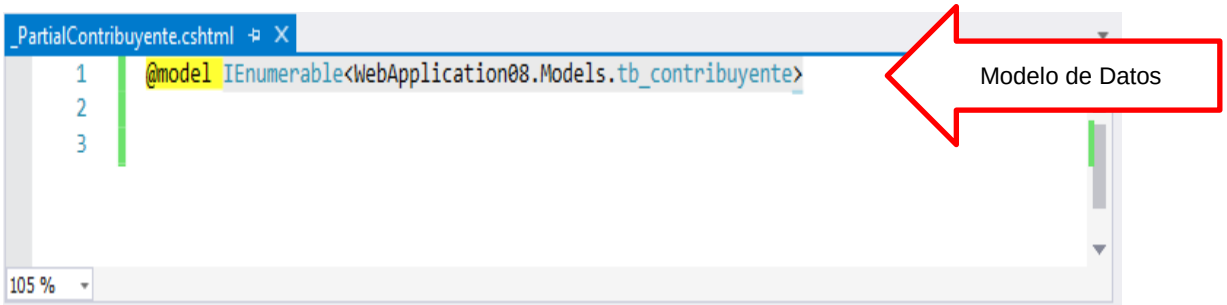


Asigne el nombre: **_PartialContribuyente**, tal como se muestra



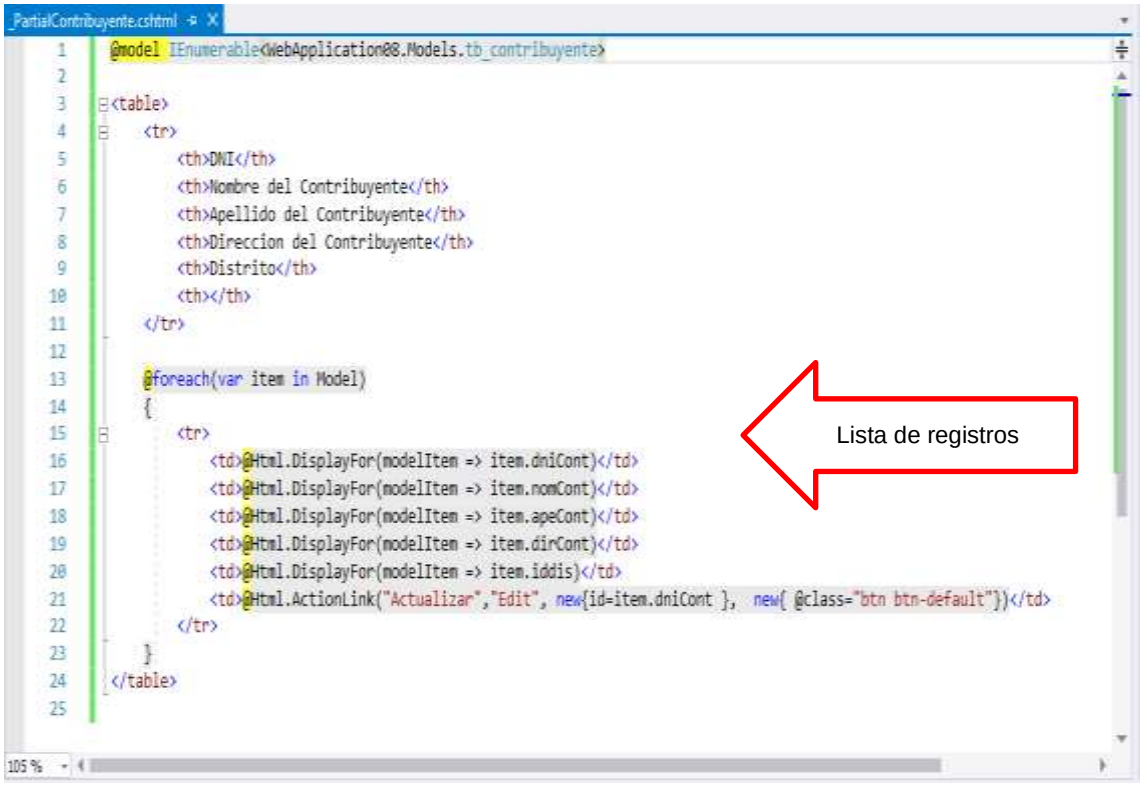
A continuación, codifique la vista parcial:

1. Defina la estructura de datos que recibirá la vista (IEnumerable de tb_contribuyente)



2. A continuación, definimos una etiqueta **<table>**, donde visualizamos la cabecera y los registros de la consulta. En esta primera parte, dentro del **<table>** definimos la cabecera, utilizando la etiqueta **<th>**, tal como se muestra

3. Definida la cabecera, vamos a definir la estructura para listar los registros. Dentro de un **foreach**, vamos a listar cada uno de los campos del registro de la consulta, tal como se muestra. Agrega un ActionLink, para ejecutar el proceso Actualizar (Edit) un registro de contribuyente por su campo dniCont.

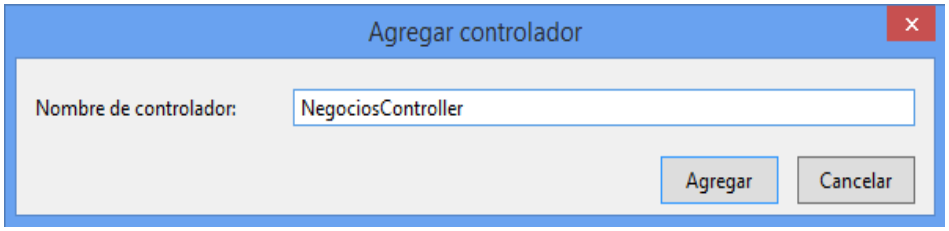


Creando un Controlador

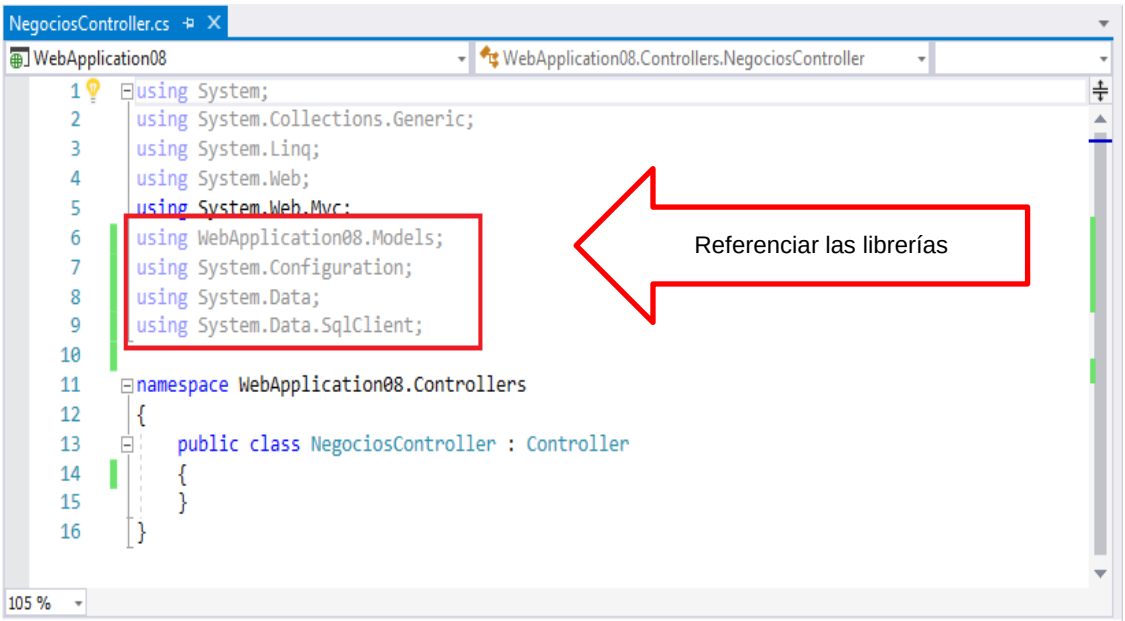
A continuación, creamos un controlador: Desde la carpeta **Controllers**, selecciona la opción **Agregar**, **Controlador**.

Selecciona el scaffold, Controlador de MVC5: en blanco, presiona el botón AGREGAR

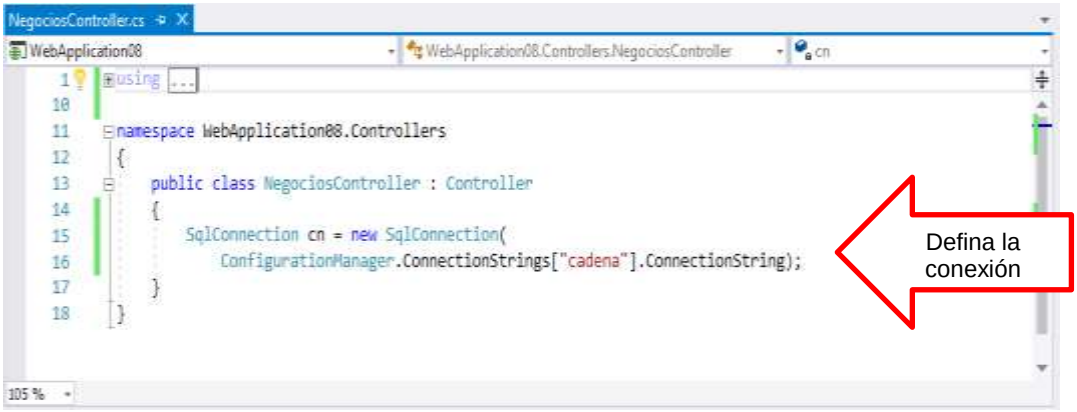
Asigne el nombre del controlador: NegociosController, tal como se muestra



Referenciar la carpeta **Models**, la librería **Data**.**SqlClient** y el **System**.**Configuration** tal como se muestra.



Dentro del controlador, defina la conexión a la base de datos Negocios2018 instanciando el **SqlConnection**, utilice el **ConfigurationManager**.

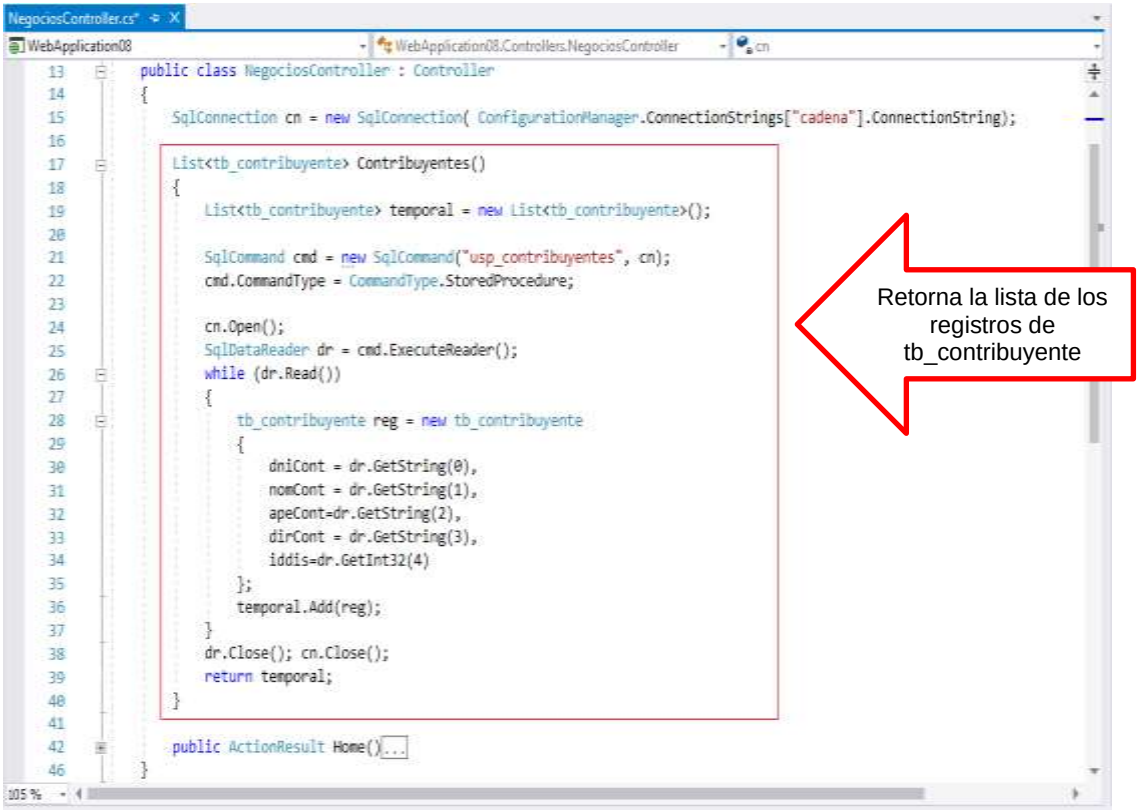


Trabajando con el ActionResult Home

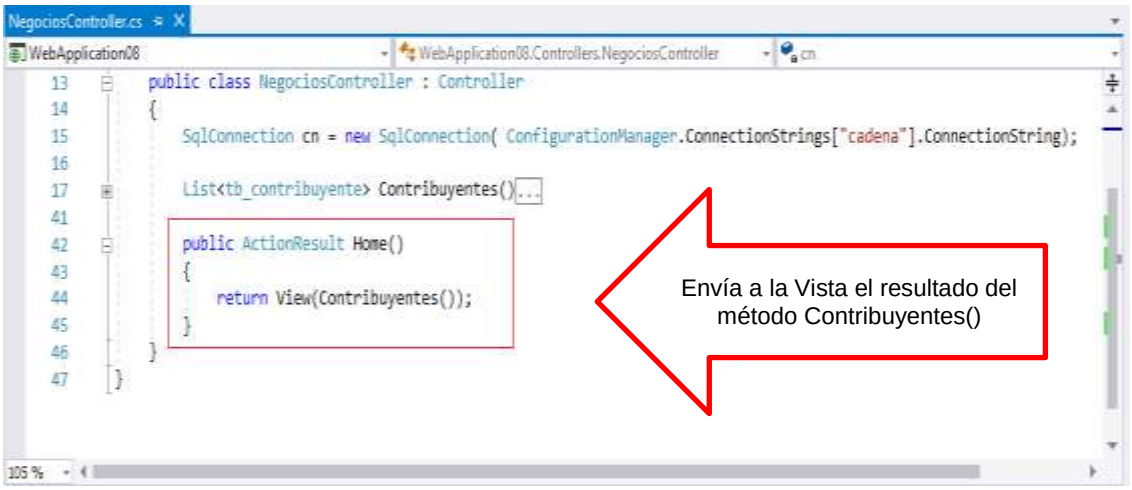
1. Defina el método Contribuyentes(), el cual ejecutará el procedimiento almacenado usp_contribuyentes, retornando los registros de tb_contribuyentes.

2. Defina el ActionResult Home(), el cual enviará a la vista la lista de los registros de tb_contribuyentes.

Codifique el método llamado Contribuyentes():
1. Ejecuta el procedure que liste los registros de tb_contribuyente a través del SqlCommand. El resultado se almacena en el SqlDataReader.
2. Para guardar los resultados en el temporal se realiza a través de un bucle (mientras se pueda leer: dr.Read()), vamos almacenando cada registro en el temporal.
3. Cerrar los objetos y enviar el objeto lista llamada temporal.

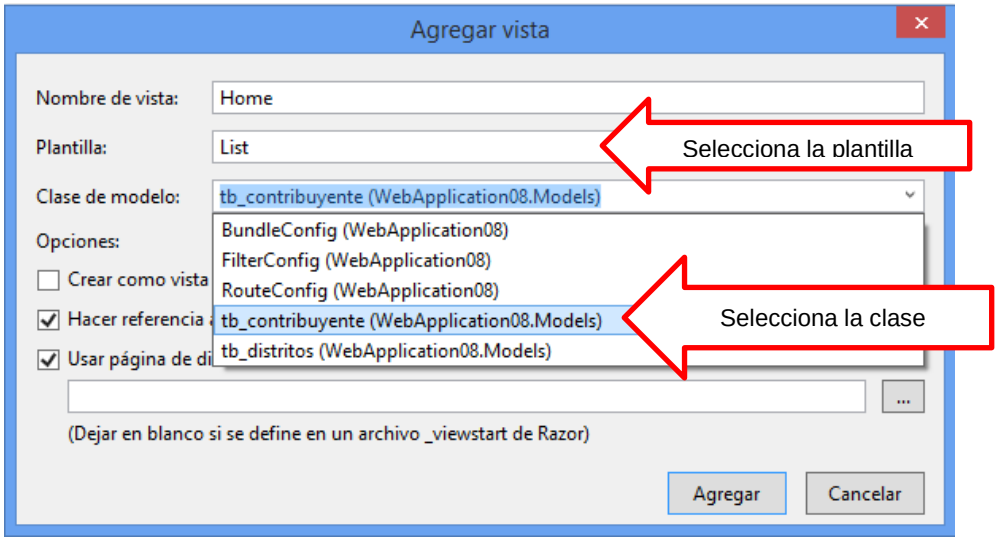


Defina el ActionResult **Home**(), el cual enviará a la Vista el resultado del método **Contribuyentes**(), tal como se muestra.



Agregando la Vista.

En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **List**; y la **clase de modelo** la cual es **tb_contribuyente**, tal como se muestra

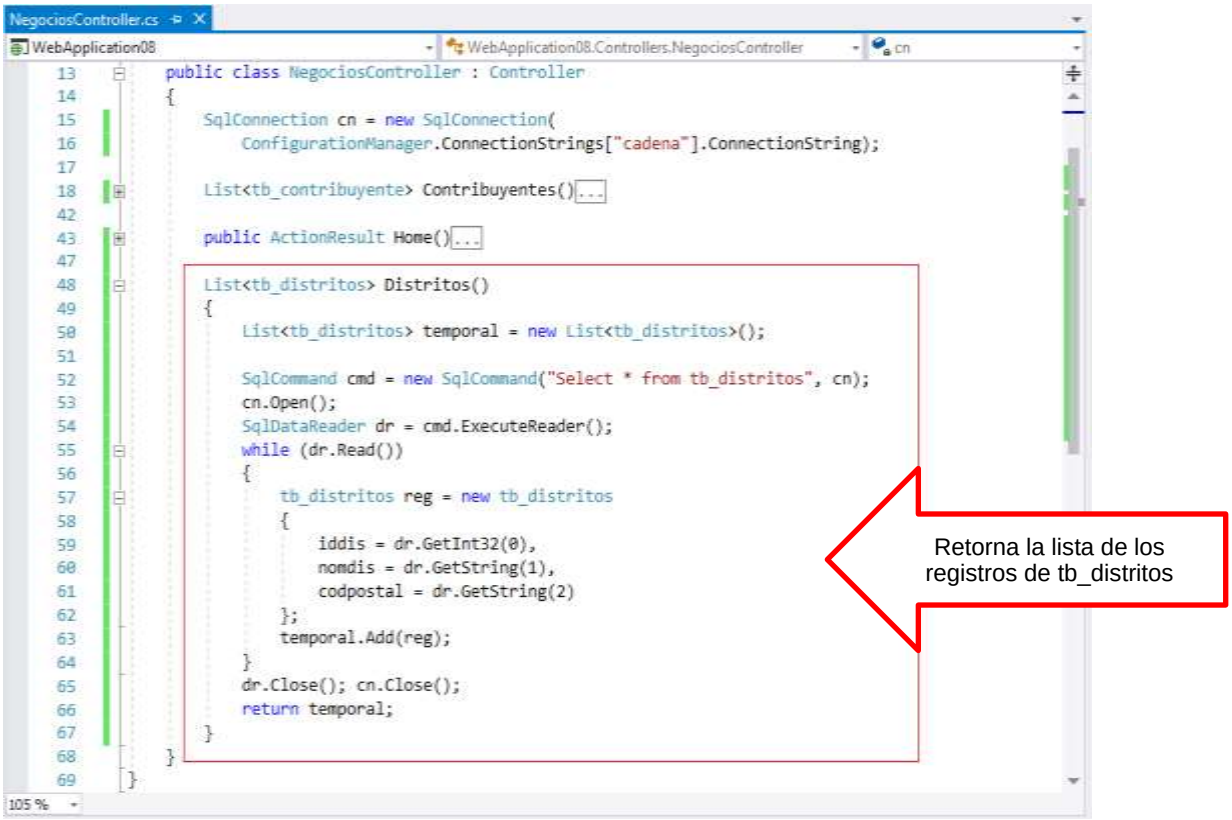


Modifica la Vista:
1. Agregar un ActionLink el cual ejecutará el ActionResult Create.
2. Agrega la vista parcial llamado _partialContribuyente

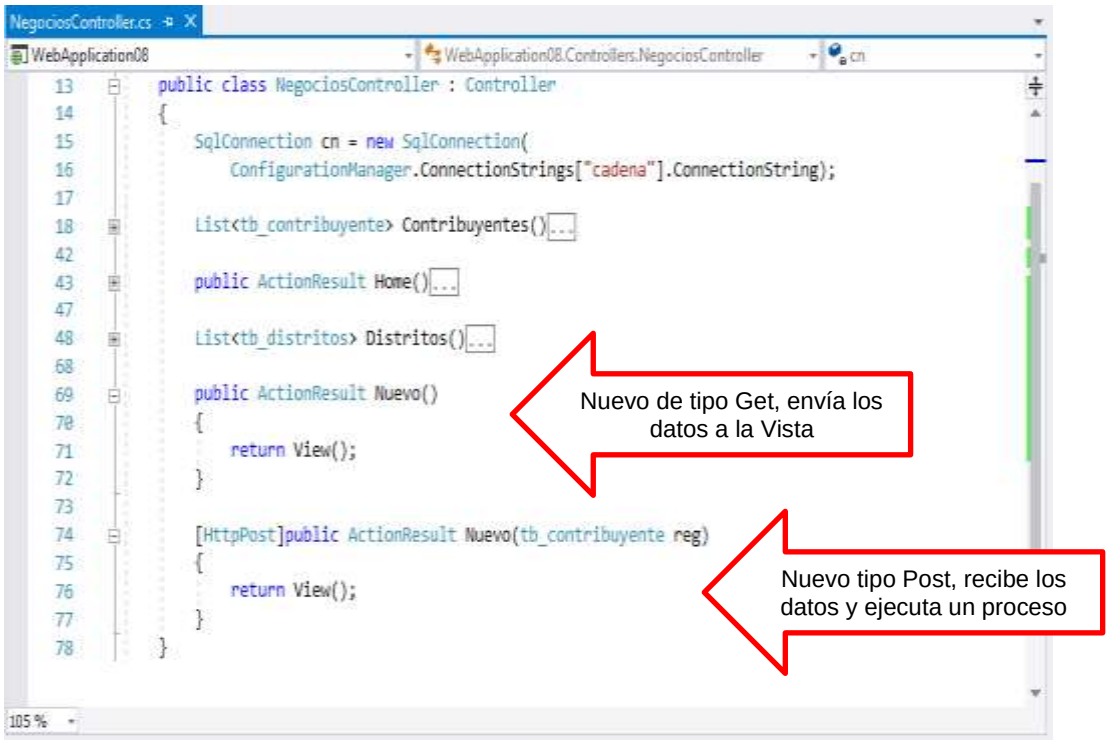
Ejecutamos la Vista, visualizamos los registros de la tabla tb_contribuyentes.

Trabajando con el ActionResult Create

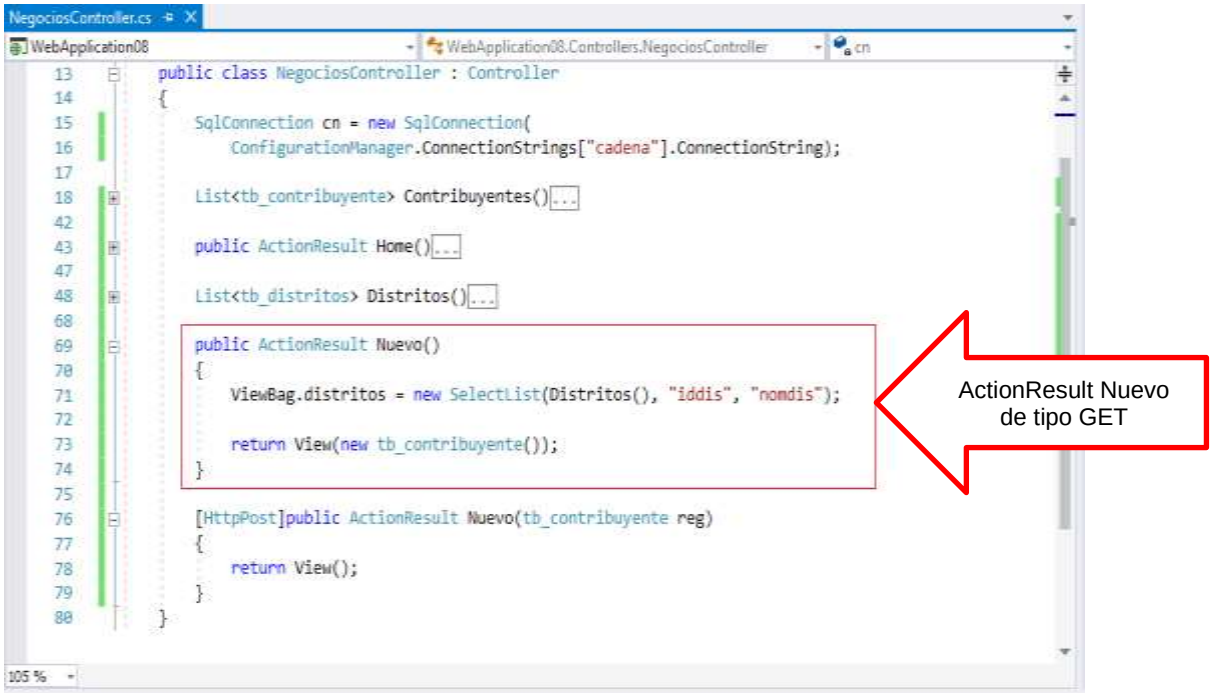
Como primer paso, definimos un método que retorna la lista de los registros de tb_distritos, el cual permitirá seleccionar el distrito donde reside el contribuyente.



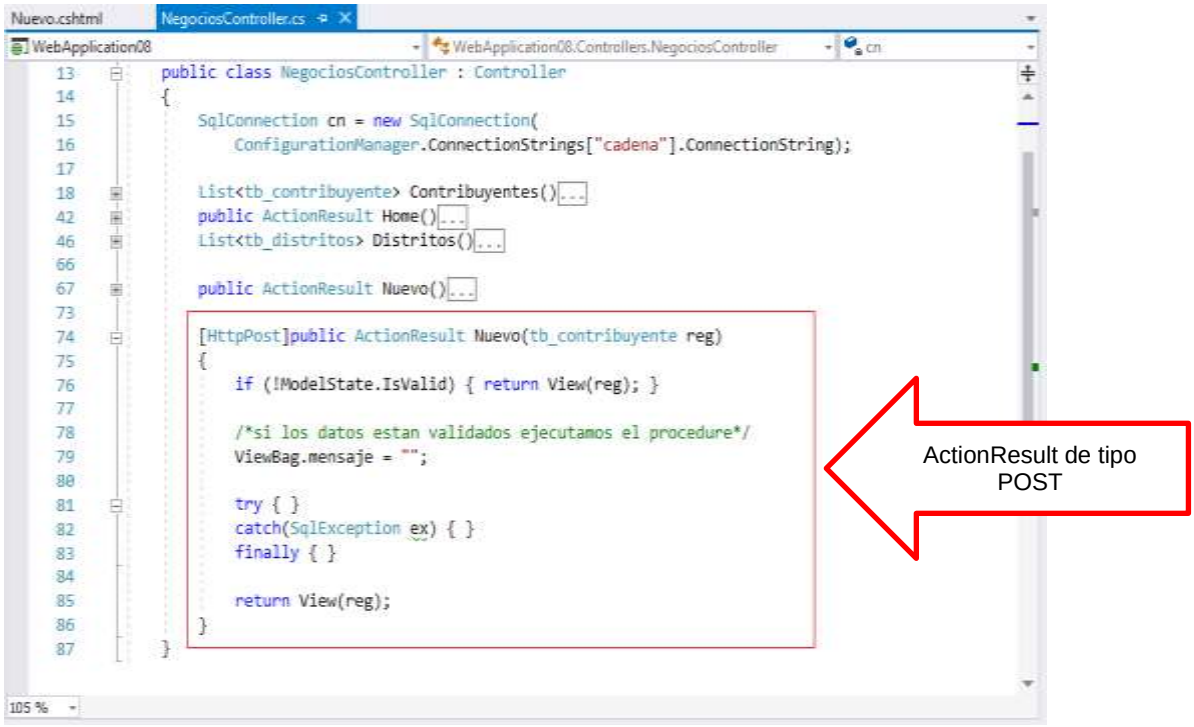
En el controlador Negocios, defina el ActionResult **Nuevo**(), de tipo Get y Post, tal como se muestra.



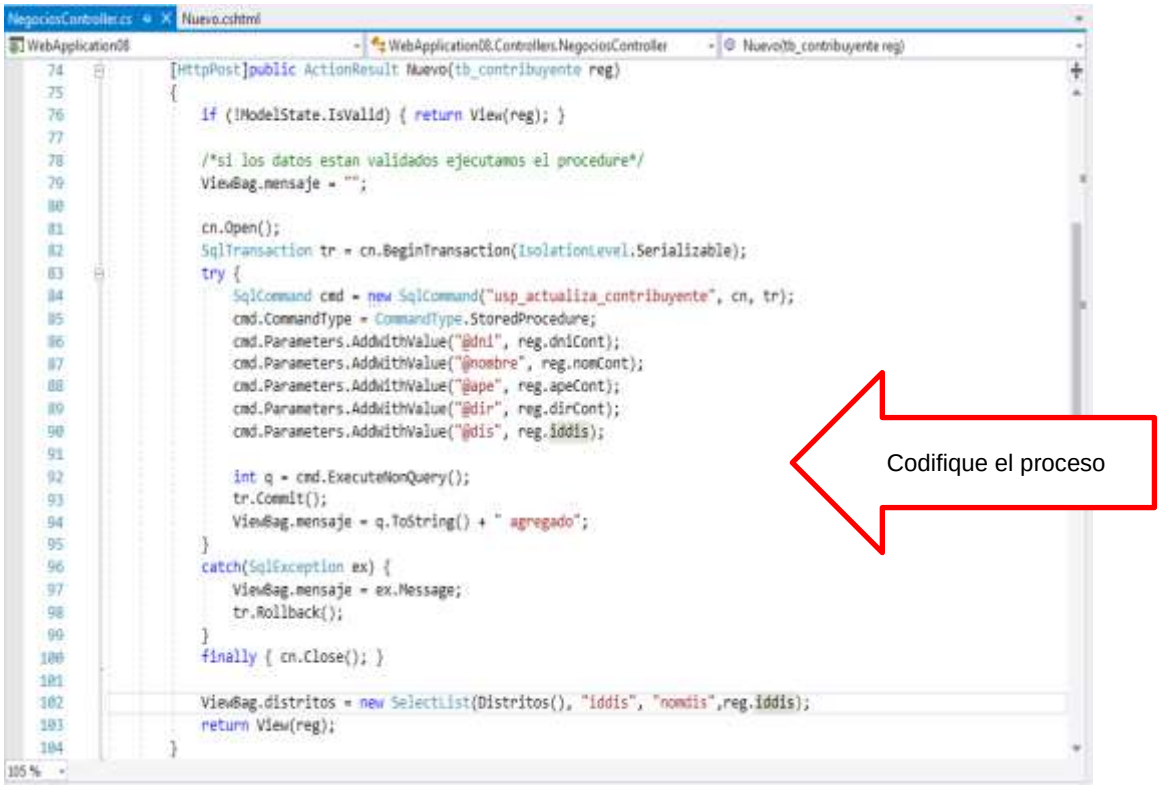
El **ActionResult Nuevo** de tipo GET, almaceno los distritos en el ViewBag.distritos; envía a la vista un nuevo registro de tb_contribuyente, tal como se muestra.



El **ActionResult Nuevo** de tipo POST, recibe los datos de la vista, los valida; si está OK, procederá a ejecutar el procedure, retornando un mensaje y los datos del registro agregado, tal como se muestra

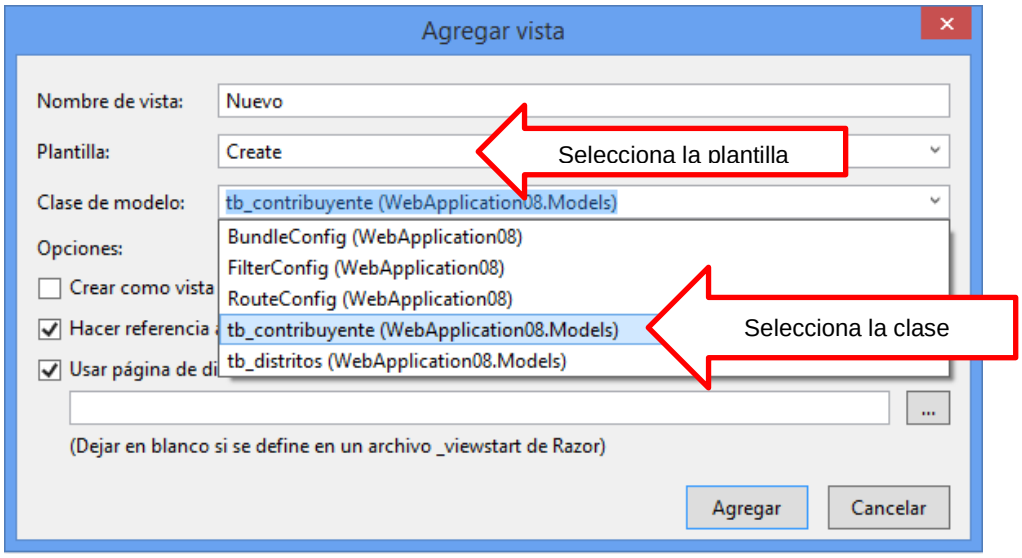


Proceso del **ActionResult Nuevo** tipo POST, ejecutando un procedimiento almacenando y controlando el proceso por un **Transaction**, tal como se muestra

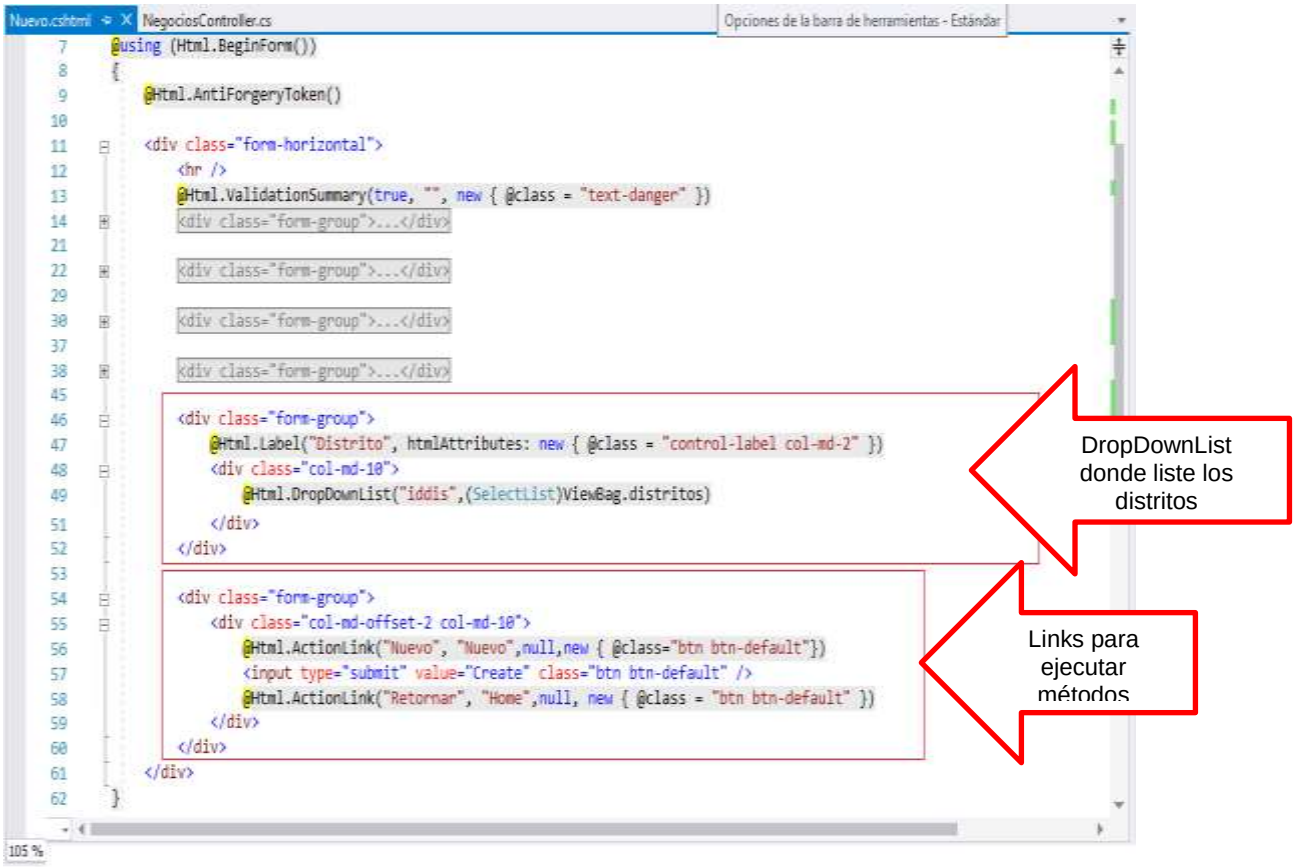


Agregando la Vista.

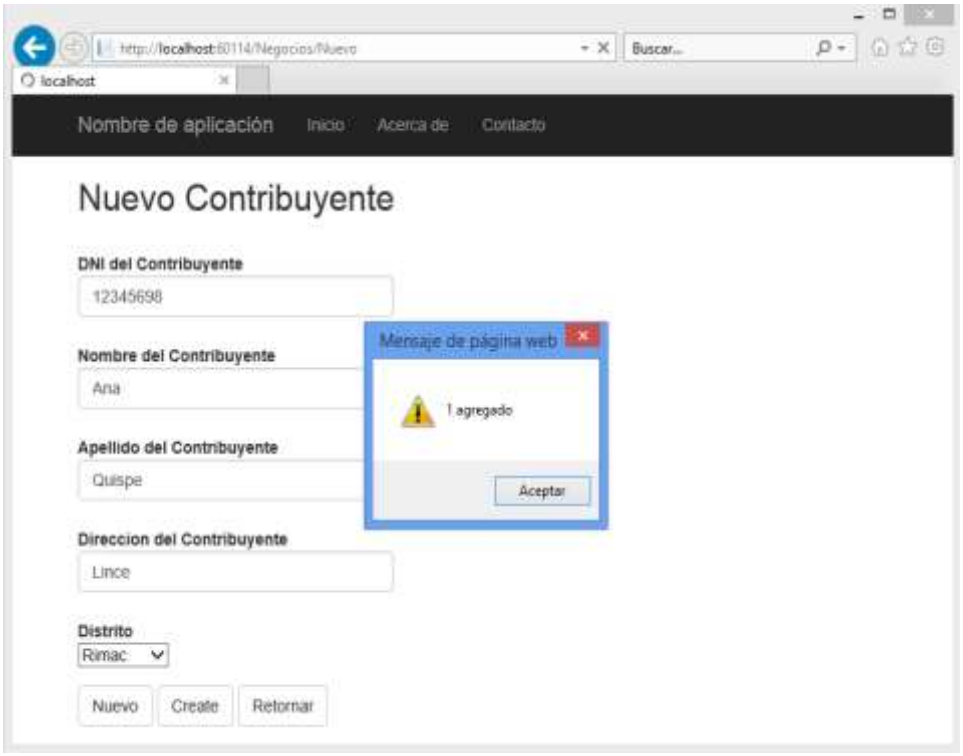
En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **Create**; y la **clase de modelo** la cual es **tb_contribuyente**, tal como se muestra



Modifique el diseño de la vista: en el campo iddis, cambiar por DropDownList, para listar los distritos; agrupe y diseñe los Links de los procesos: Nuevo, Create, Retornar, tal como se muestra



Ejecuta la Vista, ingresa los datos, al presionar el botón CREATE, agregamos un registro y visualizamos un mensaje.
Para retornar, presione el botón RETORNAR

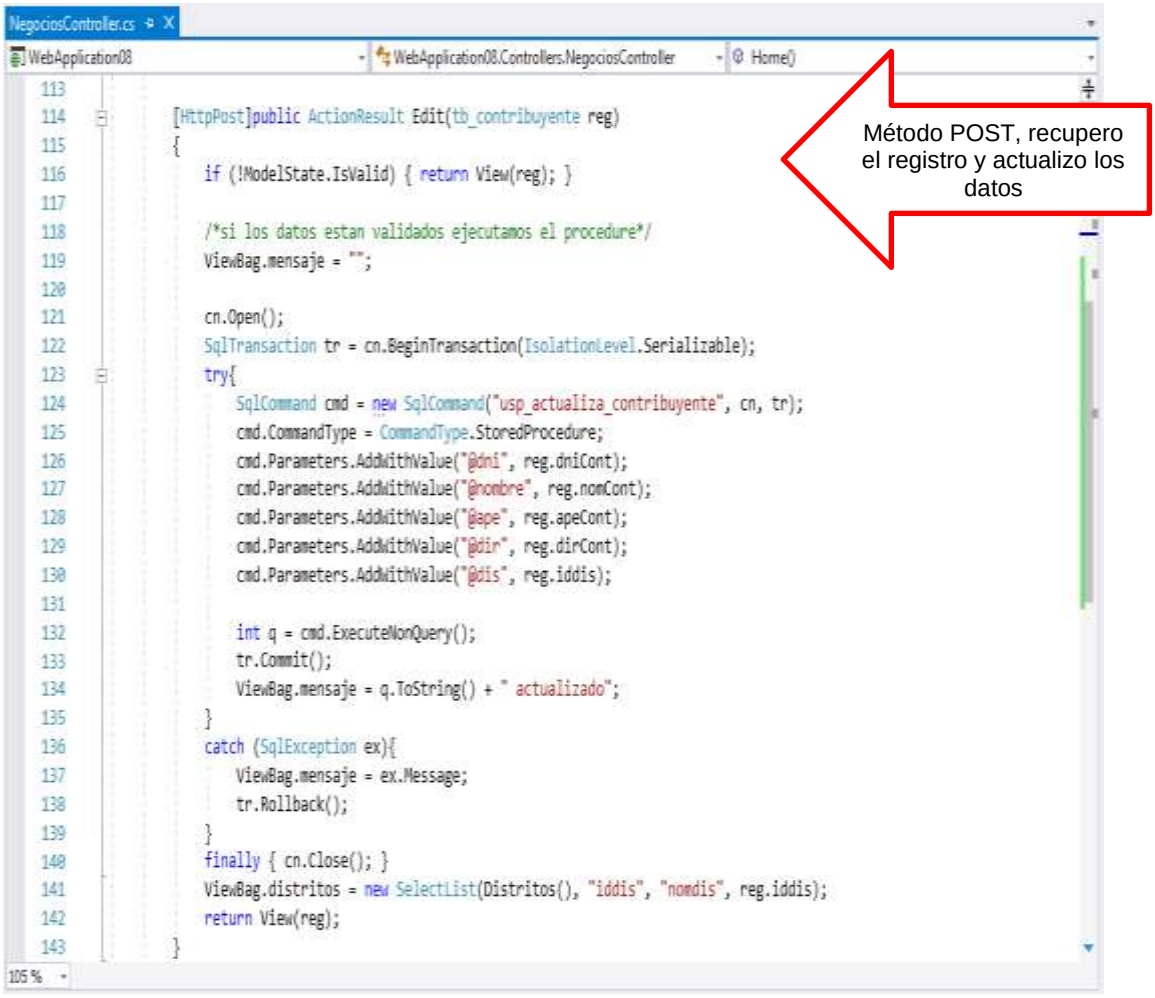


Trabajando con el ActionResult Edit

En el controlador Negocios, defina el ActionResult **Edit** GET y POST, tal como se muestra

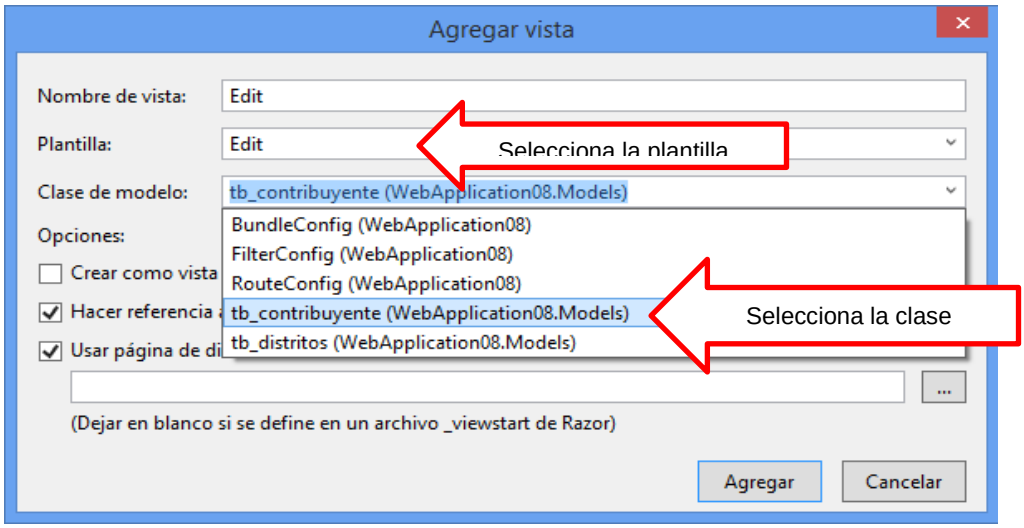
Codifique el **ActionResult Edit** (int id) método GET, donde filtra y envía el registro de la tabla tb_contribuyente filtrado por su campo dniCont; además enviamos a la vista la lista de distritos a través del ViewBag.distritos, tal como se muestra

El **ActionResult** Edit de tipo POST, recibe los datos de la vista, los valida; si está OK, procederá a ejecutar el comando UPDATE, retornando un mensaje y los datos del registro agregado, tal como se muestra



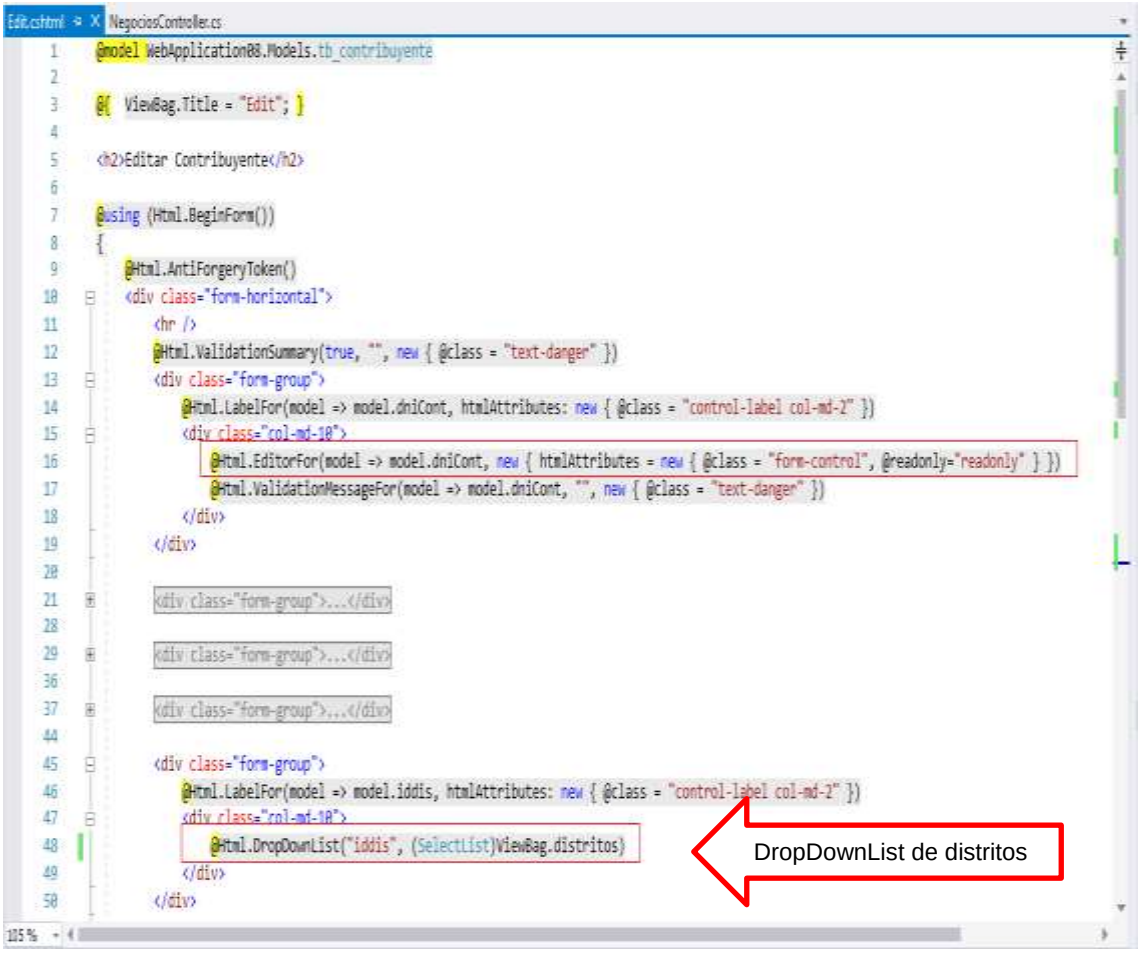
Agregando la Vista.

En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **Edit**; y la **clase de modelo** la cual es **tb_contribuyente**, tal como se muestra



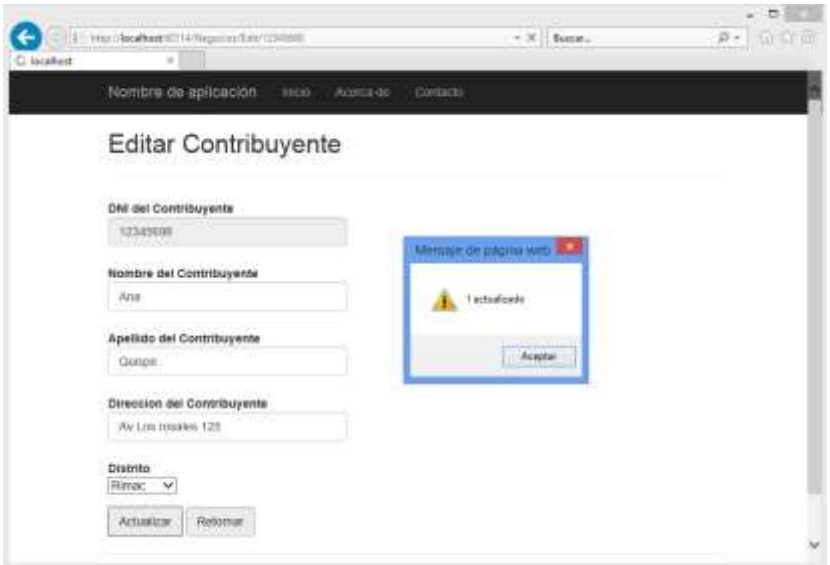
Modifique el diseño de la Vista

Para que el control EditFor del dniCont sea solo lectura, agregar la propiedad @readonly="readonly". Modifique el campo iddis por un DropDownList para listar los registros de tb_distritos, tal como se muestra



Modifique el diseño del ActionLink Retornar, tal como se muestra

Ejecuta la vista, selecciona el Contribuyente, donde visualizamos los datos del registro en la vista Edit.
Modifique los datos, al presionar el botón SAVE, se visualiza un mensaje indicando que se ha actualizado el registro.



Resumen

- Una transacción consiste en un comando único o en un grupo de comandos que se ejecutan como un paquete. Las transacciones permiten combinar varias operaciones en una sola unidad de trabajo. Si en un punto de la transacción se produjera un error, todas las actualizaciones podrían revertirse y devolverse al estado que tenían antes de la transacción.
- Una transacción debe ajustarse a las propiedades: atomicidad, coherencia, aislamiento y durabilidad para poder garantizar la coherencia de datos. La mayoría de los sistemas de bases de datos relacionales, como Microsoft SQL Server, admiten transacciones, al proporcionar funciones de bloqueo, registro y administración de transacciones cada vez que una aplicación cliente realiza una operación de actualización, inserción o eliminación
- Una transacción se considera local cuando consta de una única fase y es controlada directamente por la base de datos. Cada proveedor de datos de .NET Framework tiene su propio objeto Transaction para realizar transacciones locales. Si necesita que se realice una transacción en una base de datos de SQL Server, seleccione una transacción de **System**.**Data**.**SqlClient**. En transacciones de Oracle, utilice el proveedor **System**.**Data**.**OracleClient**. Además, existe una nueva clase **DbTransaction** disponible para la escritura de código independiente del proveedor que requiere transacciones.
- En ADO.NET, las transacciones se controlan con el objeto Connection. Puede iniciar una transacción local con el método **BeginTransaction**. Una vez iniciada una transacción, puede inscribir un comando en esa transacción con la propiedad Transaction de un objeto **Command**. Luego, puede confirmar o revertir las modificaciones realizadas en el origen de datos según el resultado correcto o incorrecto de los componentes de la transacción. Las operaciones para confirmar una transacción es **Commit** y la operación para revertir o deshacer una transacción es **RollBack**.
- La clase **TransactionScope** crea un bloque de código transaccional al inscribir implícitamente las conexiones en una transacción distribuida. Debe llamar al método **Complete** al final del bloque **TransactionScope** antes de abandonarlo. Al salir del bloque se invoca el método **Dispose**. Si se ha producido una excepción que ocasiona que el código salga del ámbito, la transacción se considera anulada
- La clase **TransactionScope** crea una transacción con un **IsolationLevel** predeterminado de **Serializable**. Dependiendo de la aplicación, podría estudiar la posibilidad de reducir el nivel de aislamiento para evitar una elevada contención en la aplicación.
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.

  - http://msdn.microsoft.com/es-es/library/777e5ebh(v=vs.110).aspx
  - http://msdn.microsoft.com/es-es/library/2k2hy99x(v=vs.110).aspx
  - http://msdn.microsoft.com/es-es/library/ms254973(v=vs.110).aspx

UNIDAD

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 6: Arquitectura de Capas Orientadas al Dominio (5 horas)

6.1 Introducción
6.2 Capas de la arquitectura DDD
6.3 Implementando una capa de Dominio en un proceso CRUD.

ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando el patrón ASP.NET MVC para realizar operaciones de actualización de datos.
- Los alumnos desarrollan los laboratorios de esta semana

6. Arquitectura “N” capas orientadas al Dominio

6.1 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.

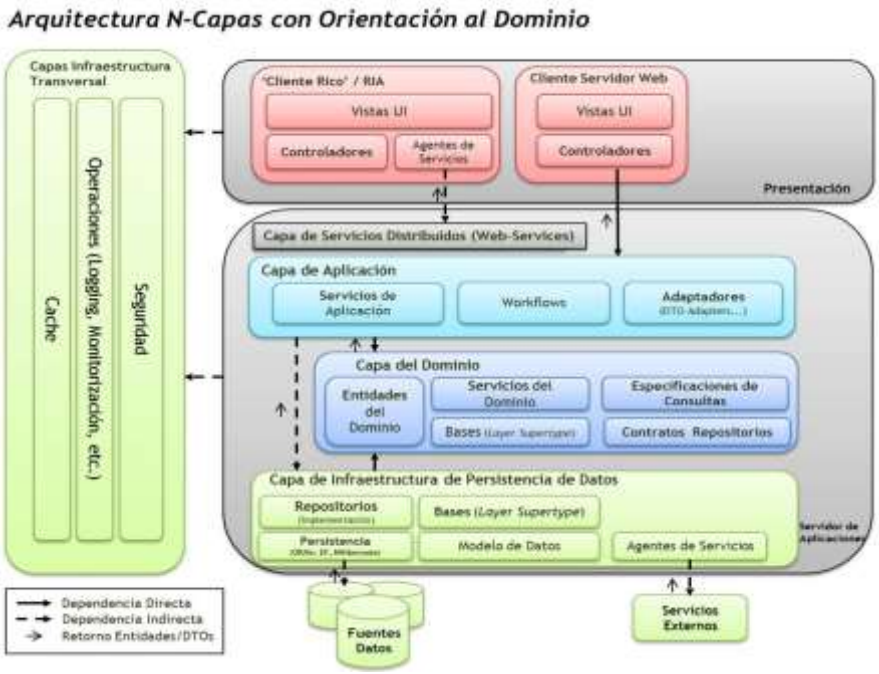


Figura 1: Arquitectura de capas orientadas al dominio
http://es.stackoverflow.com/questions/41889/asp-net-mvc-arquitectura-ddddomain-driven-design

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 Ubicuo). 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

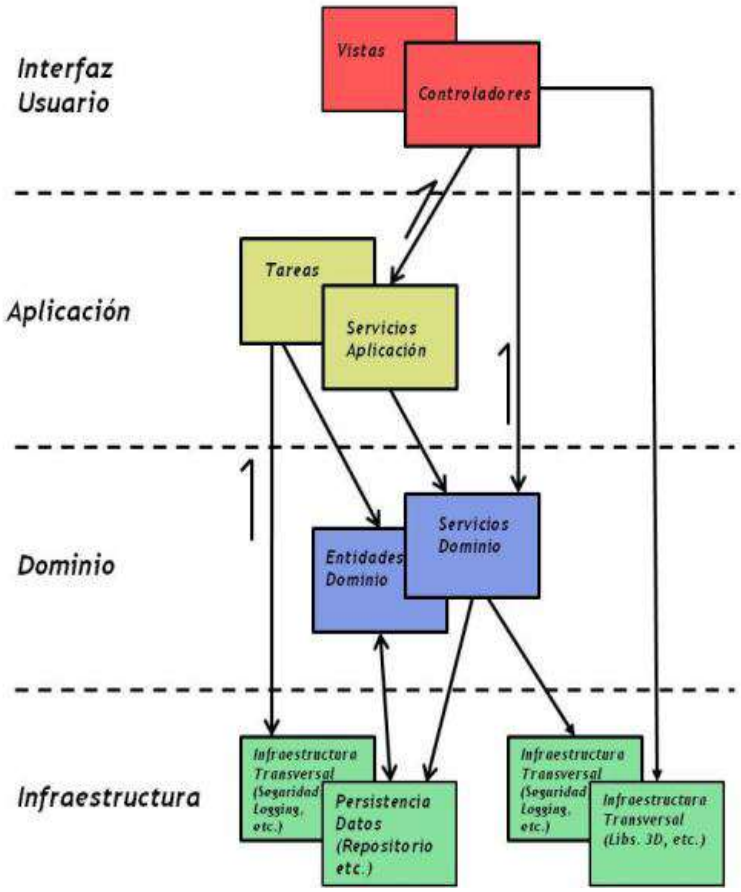


Figura 2: Interaccción de la capas

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.

6.2 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 aplication. 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.

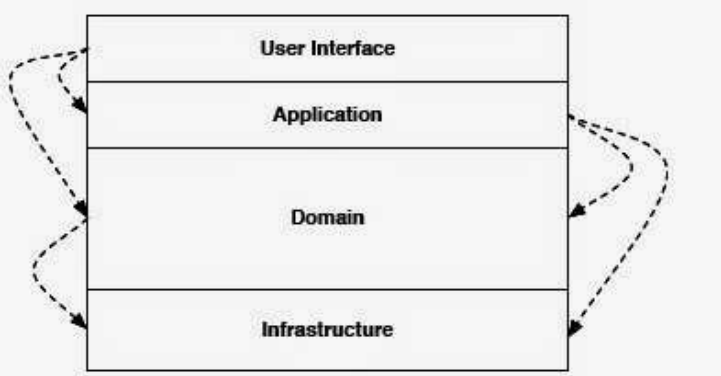


Figura 3: Capas en una aplicación MVC
http://nfanjul.blogspot.pe/2014/09/arquitectura-ddd-y-sus-capas.html

6.3 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.

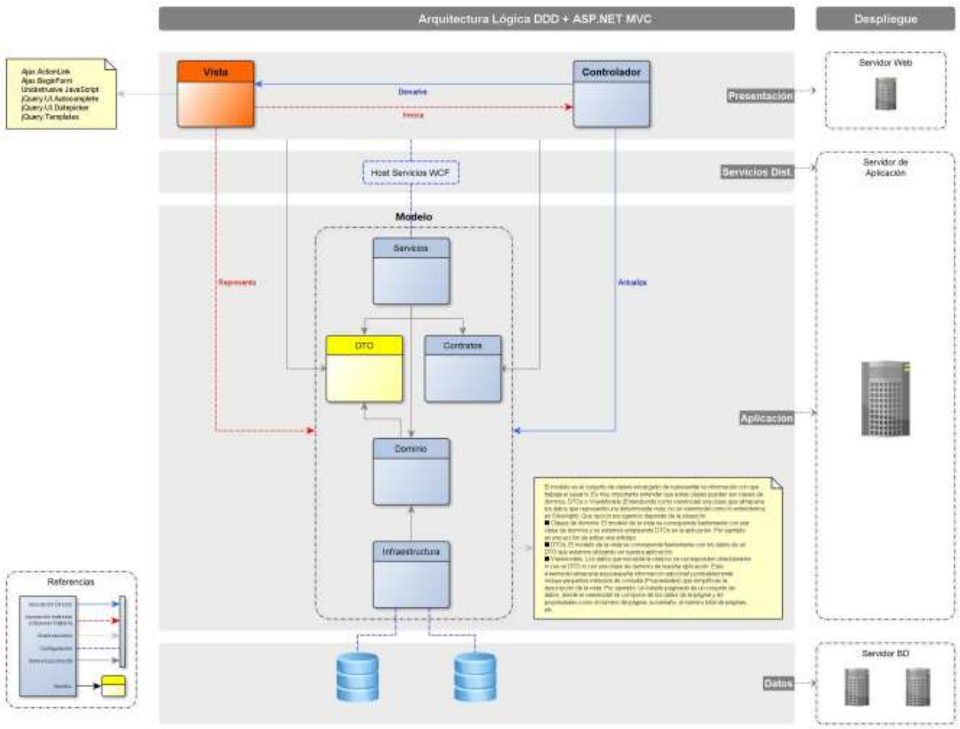


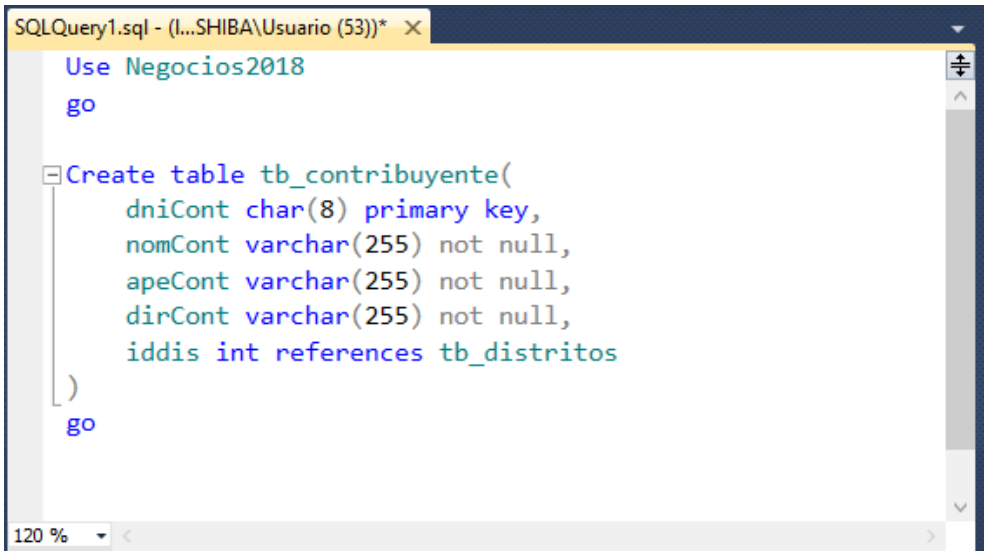
Figura4: Implementando Capas + aplicación MVC
https://jjestruch.wordpress.com/2012/02/21/arquitectura-ddd-domain-driven-design-asp-net-mvc/

Laboratorio 6.1
Manipulación de Datos - Capas de Dominio

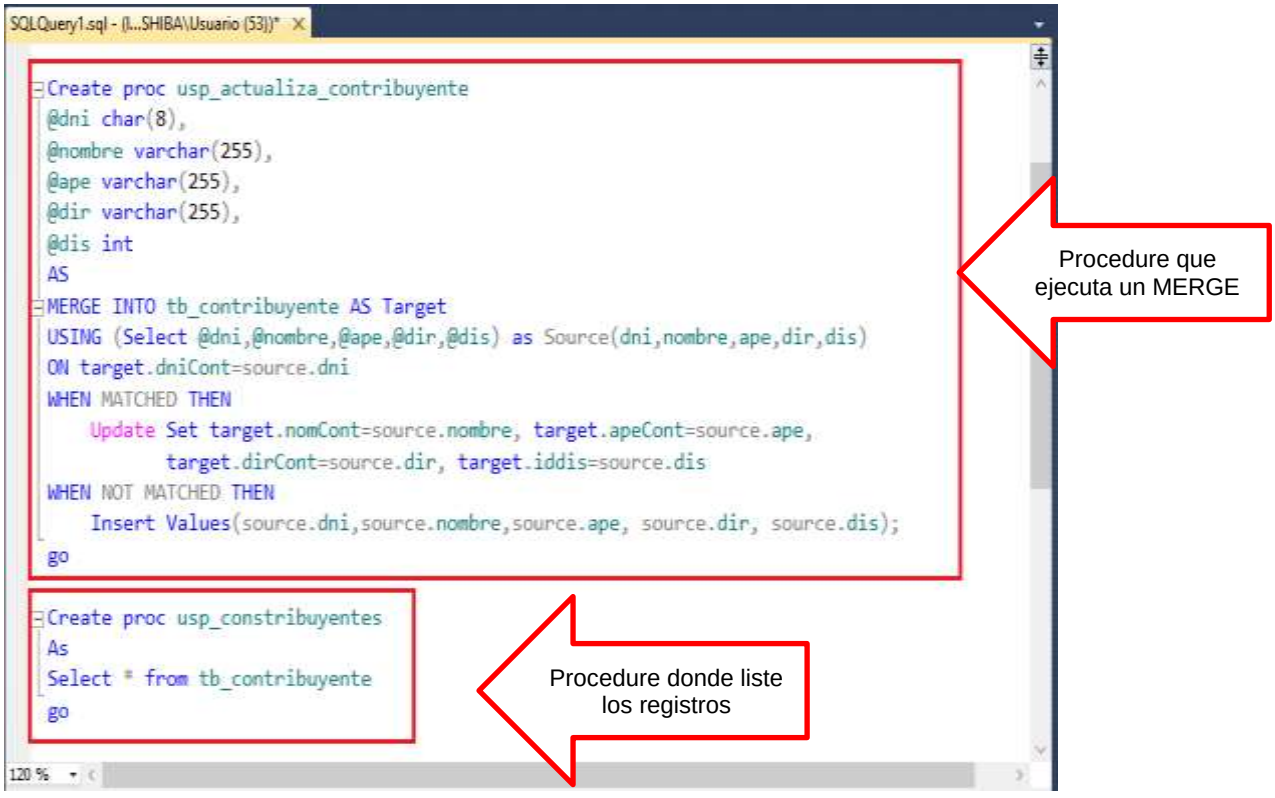
Implemente un proyecto ASP.NET MVC, donde permita INSERTAR, CONSULTAR y ACTUALIZAR los datos de la tabla tb_contribuyente.

Creando la tabla tb_contribuyente

En el administrador del SQL Server, defina la tabla tb_contribuyente, tal como se muestra.

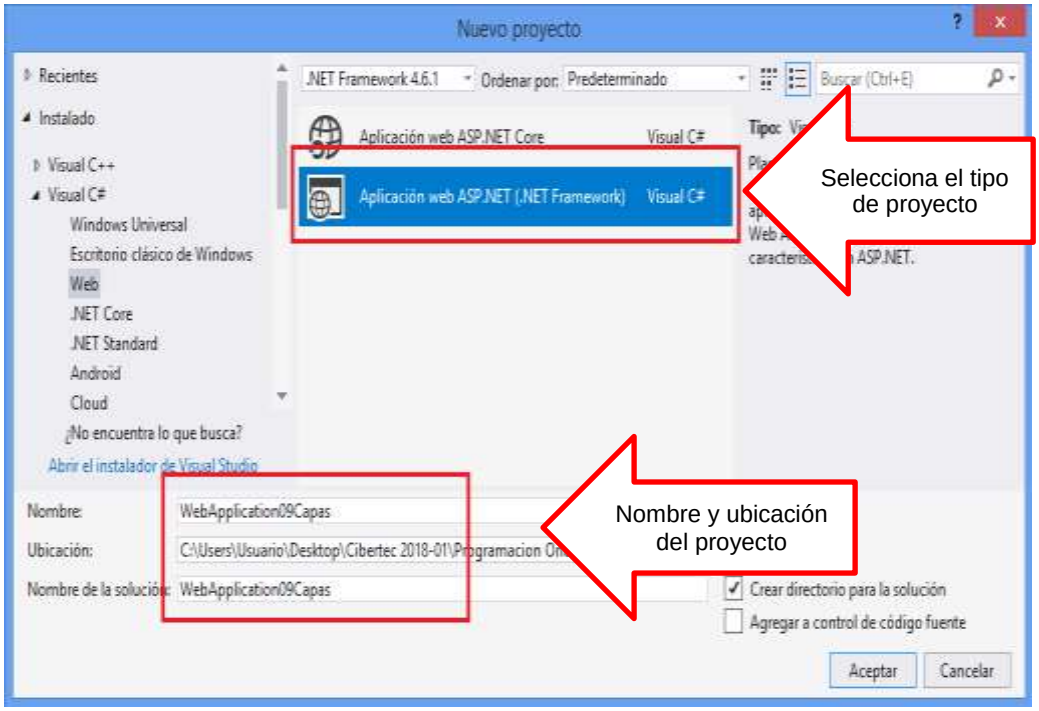


Crea los procedimientos almacenados: usp_actualiza_contribuyente, donde inserta o actualiza un registro (uso del MERGE), usp_contribuyentes, lista los registros

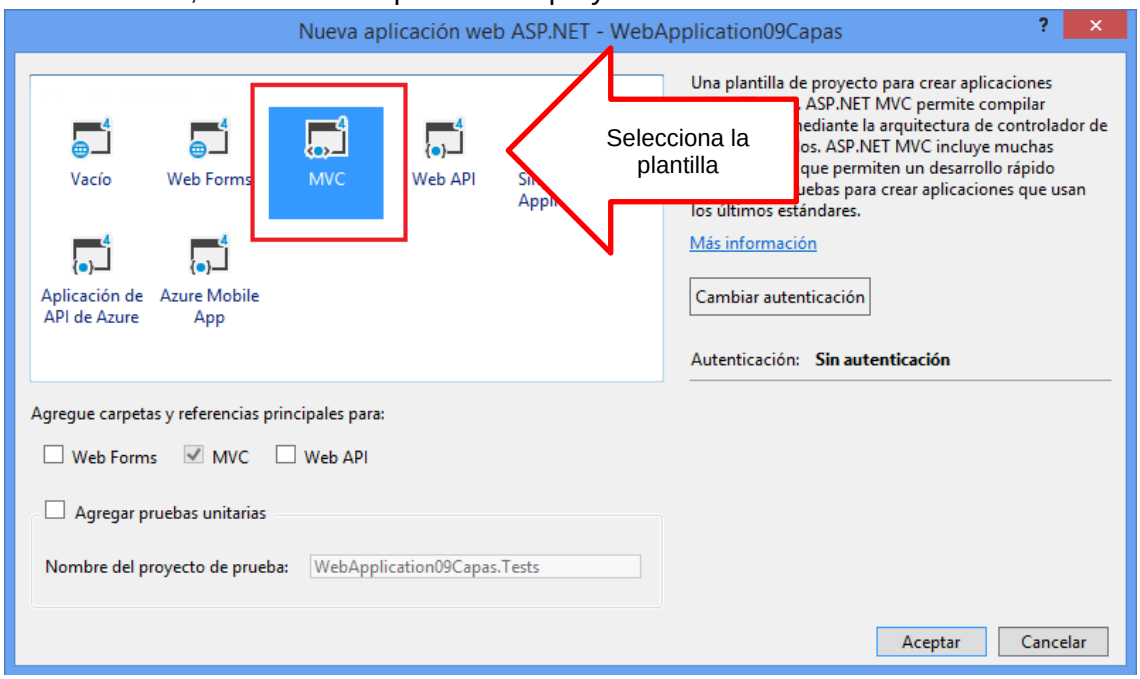


1. Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR

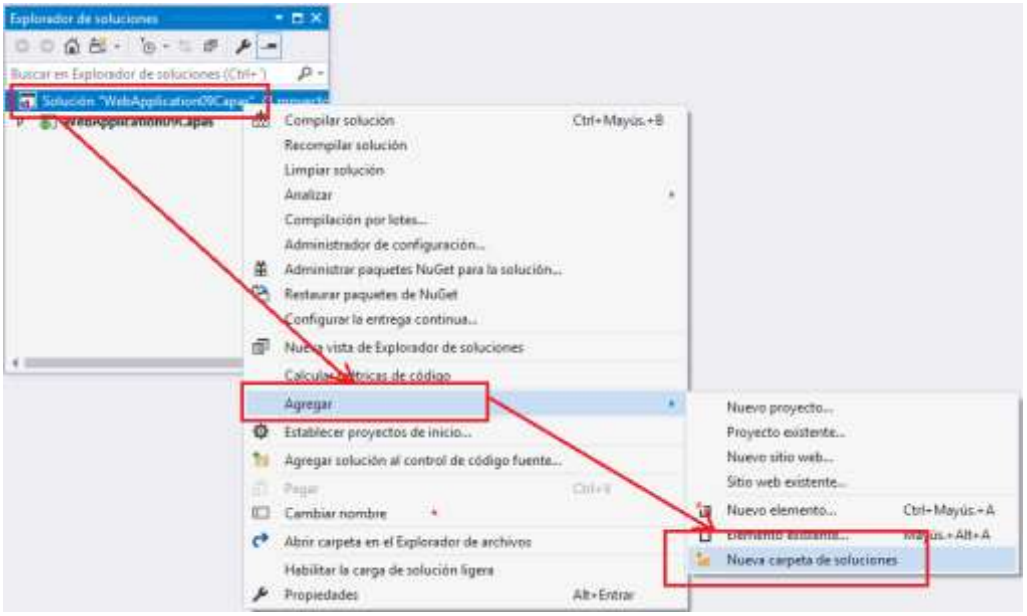


A continuación, seleccionar la plantilla del proyecto MVC.

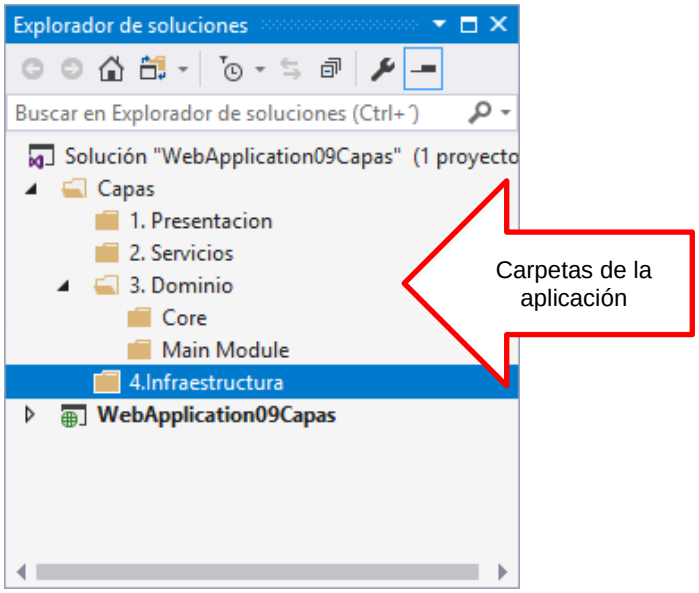


2. Estructura de la aplicación

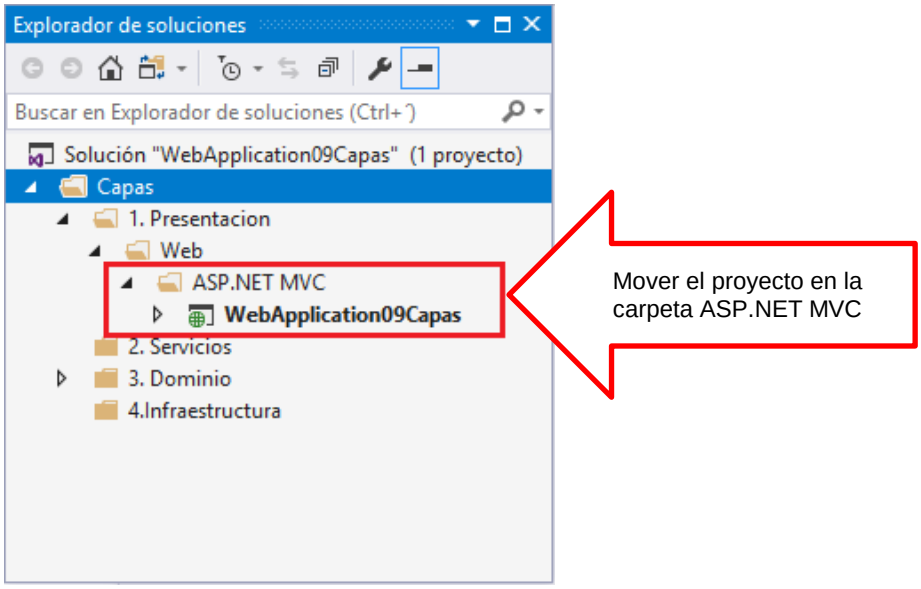
Una vez creado el proyecto base procedemos a armar la estructura de la aplicación, la cual será básicamente carpetas en donde irán los proyectos.
Clic derecho al archivo de solución → Agregar → Nueva carpeta de soluciones (haremos esta operación varias veces)



Crearemos las siguientes carpetas con la siguiente jerarquía, tal como se muestra en la figura

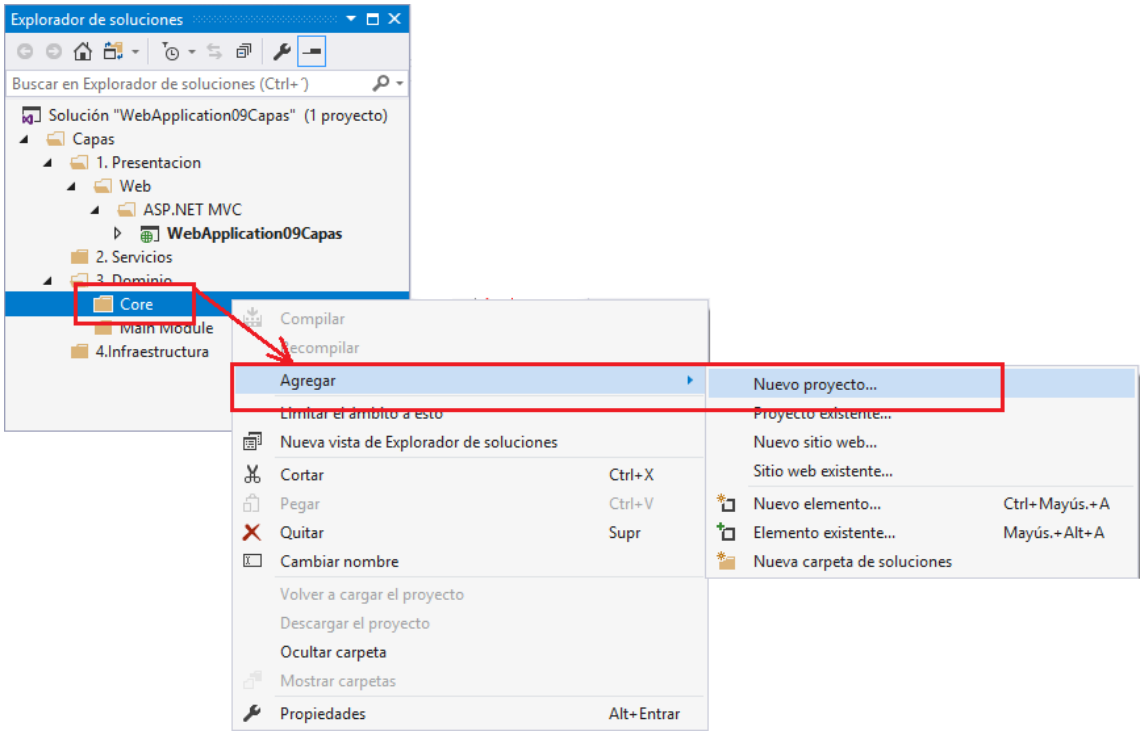


A continuación, movemos el proyecto WebApplication09Capas a la carpeta ASP.NET MVC, tal como se muestra en la figura.

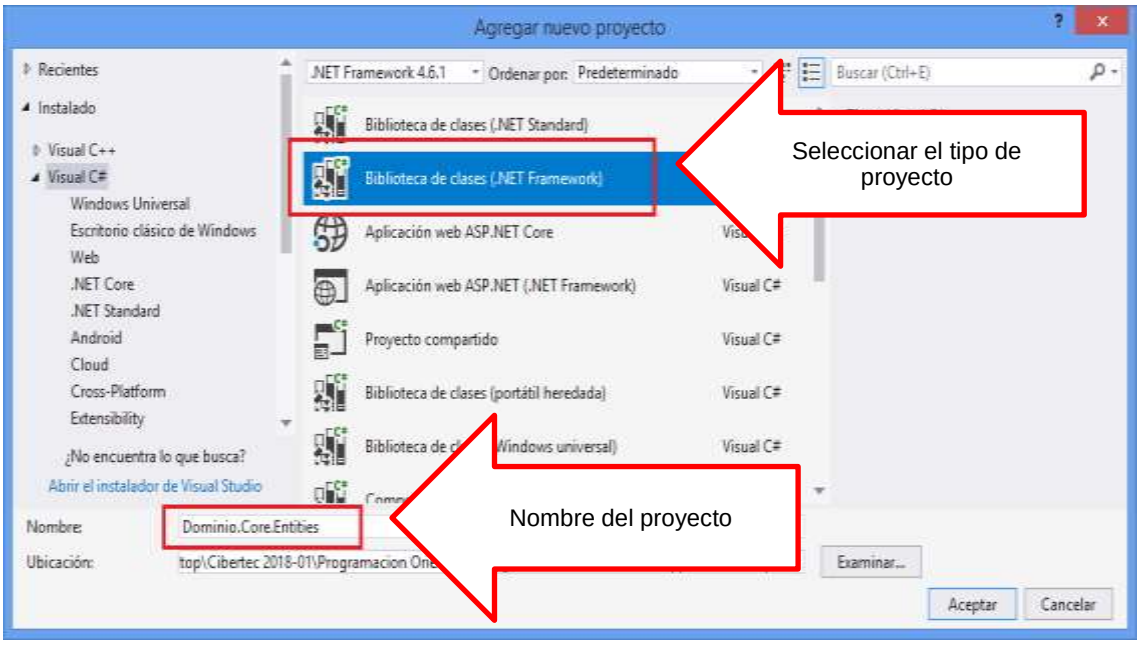


Creando el proyecto Entities

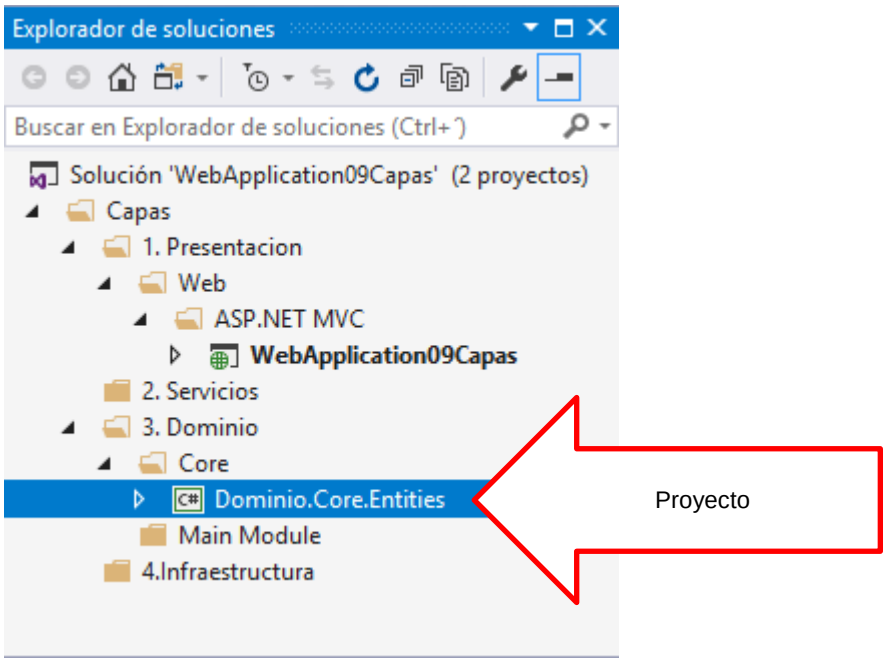
A continuación agregamos en la capeta Core un proyecto: Clic derecho a la carpeta Core → Agregar → Nuevo Proyecto



El primer proyecto se llamará **Dominio.Core.Entities**. Si lo relacionamos con la estructura tradicional de N Capas vendría a ser el proyecto Entities.

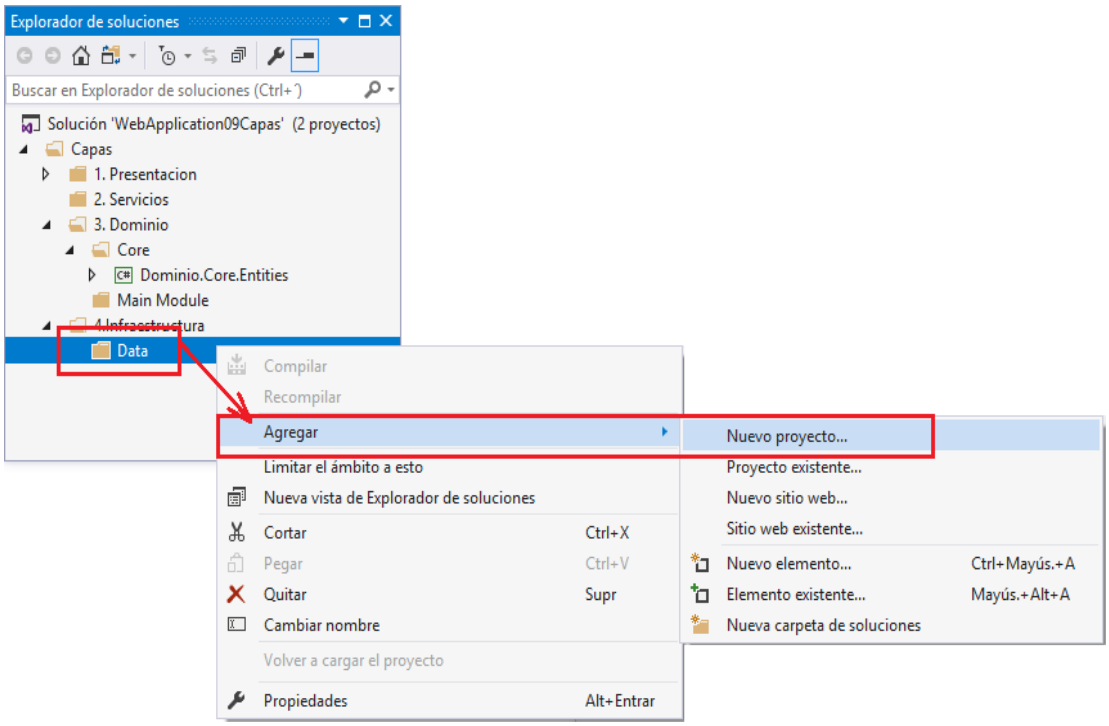


Proyecto Dominio.Core.Entities, ubicado en la carpeta Core, tal como se muestra.

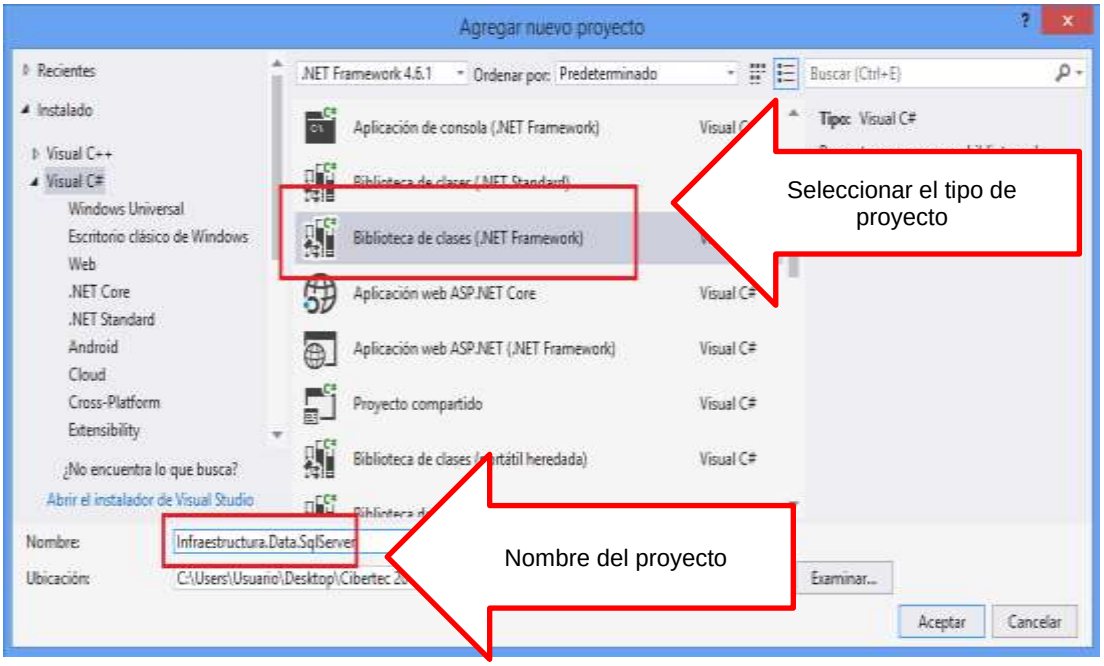


Creando el proyecto DataAccess

A continuación agregamos en la capeta Data un proyecto: Clic derecho a la carpeta Core → Agregar → Nuevo Proyecto

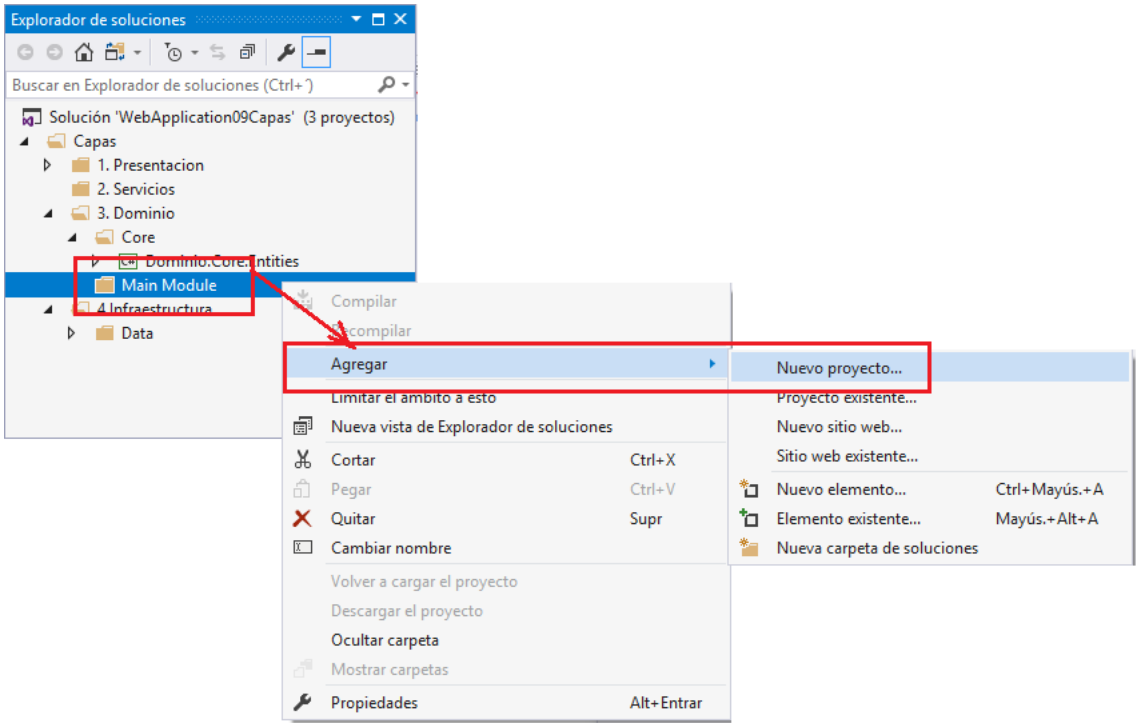


El proyecto se llamará **Infraestructura.Data.SqlServer**. Si lo relacionamos con la estructura tradicional de N Capas vendría a ser el proyecto DataAccess.

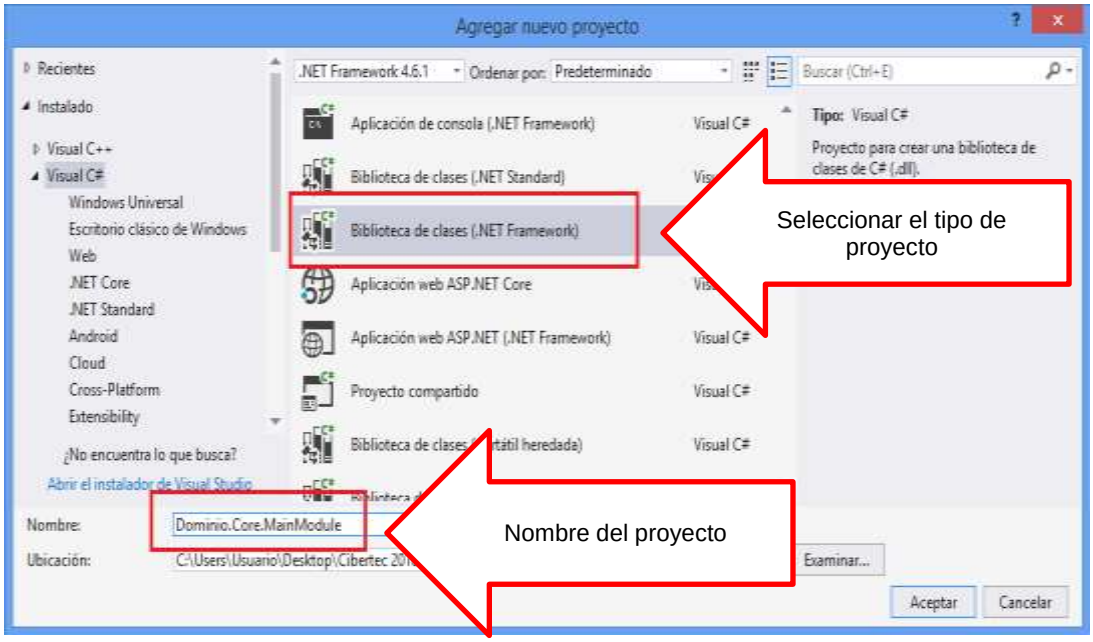


Creando el proyecto BusinessLayer

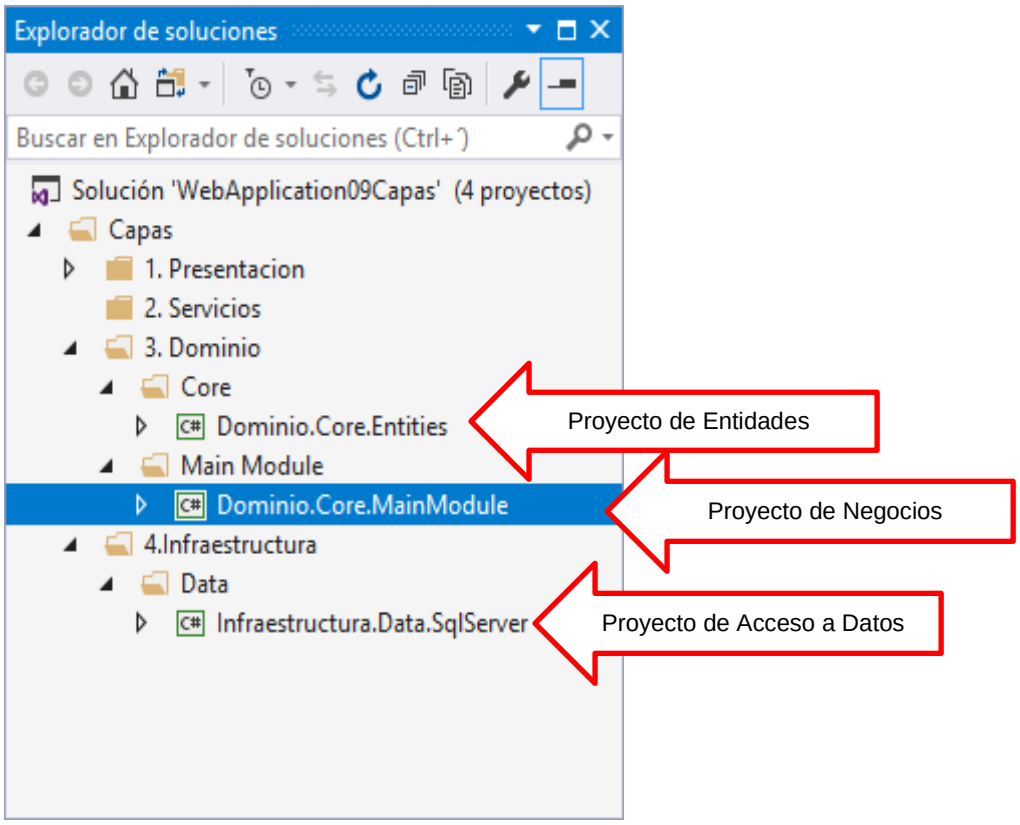
A continuación, creamos el proyecto de Negocios: Clic derecho a la carpeta Main Module → Agregar → Nuevo Proyecto
Nombramos al proyecto de Biblioteca de clases Dominio.MainModule



El proyecto se llamará **Dominio.Core.MainModule**. Si lo relacionamos con la estructura tradicional de N Capas vendría a ser el proyecto BusinessLayer



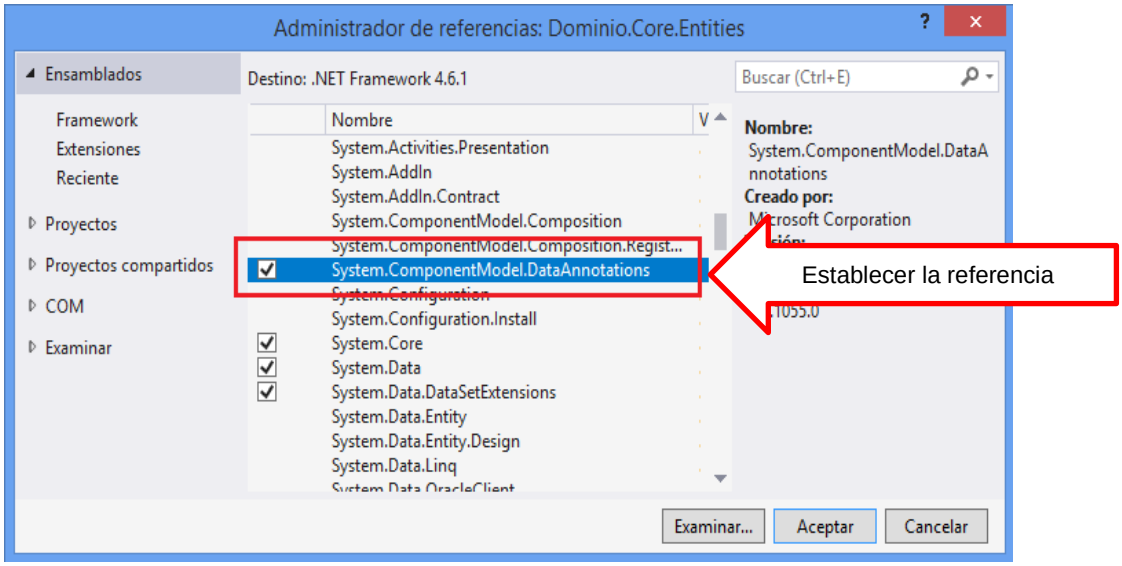
A concluir con este proceso, hemos agregado 3 proyectos a la solución, tal como se muestra.



3. Estableciendo las referencias

a) Dominio.Core.Entities

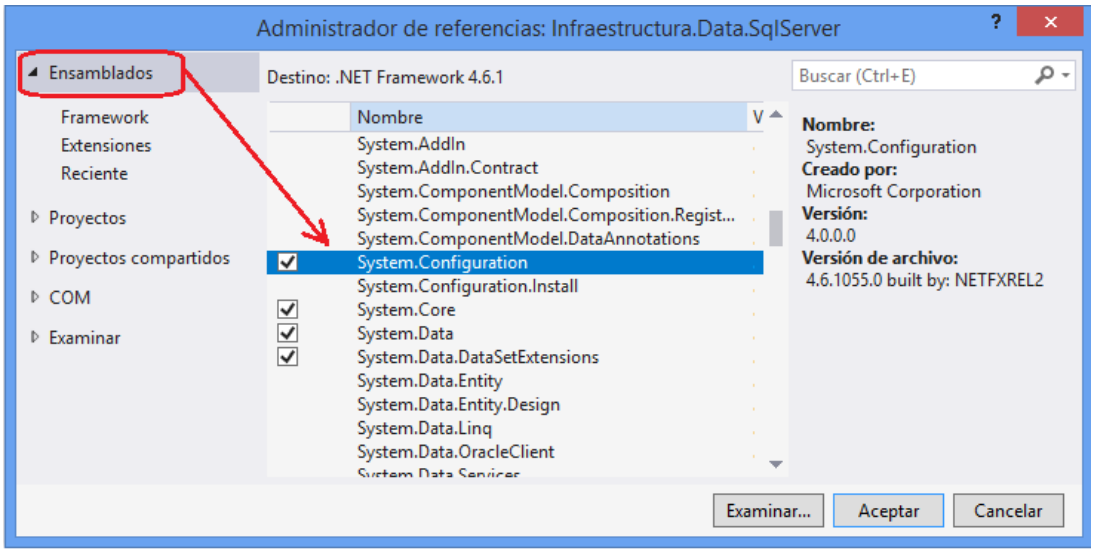
Establecer la referencia a System.ComponentModel.DataAnnotations, tal como se muestra.

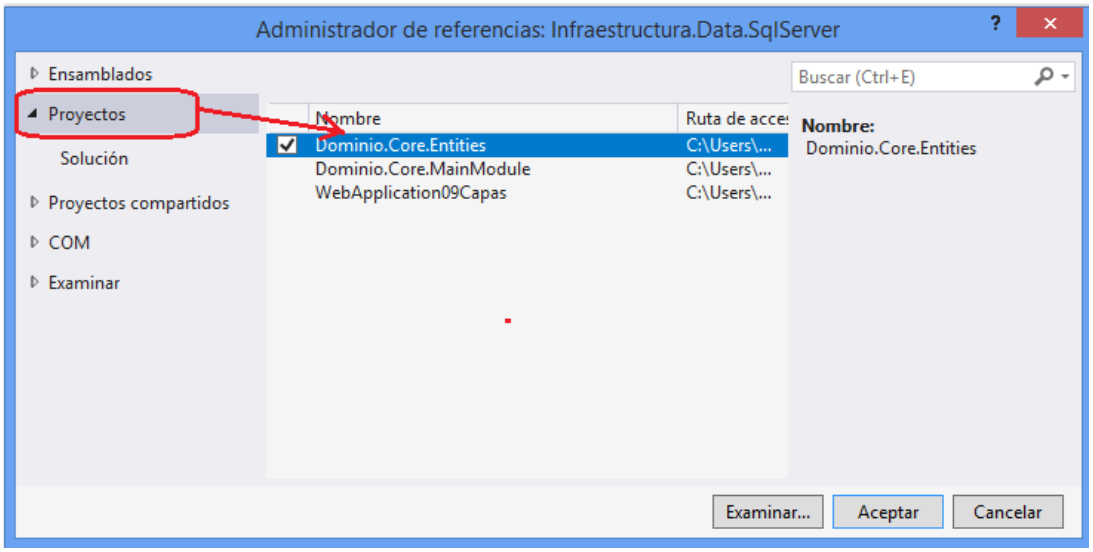


b) Infraestructura.Data.SqlServer

Establecer la referencia a System.Configuration, la cual esta ubicación en la opción Ensamblados.

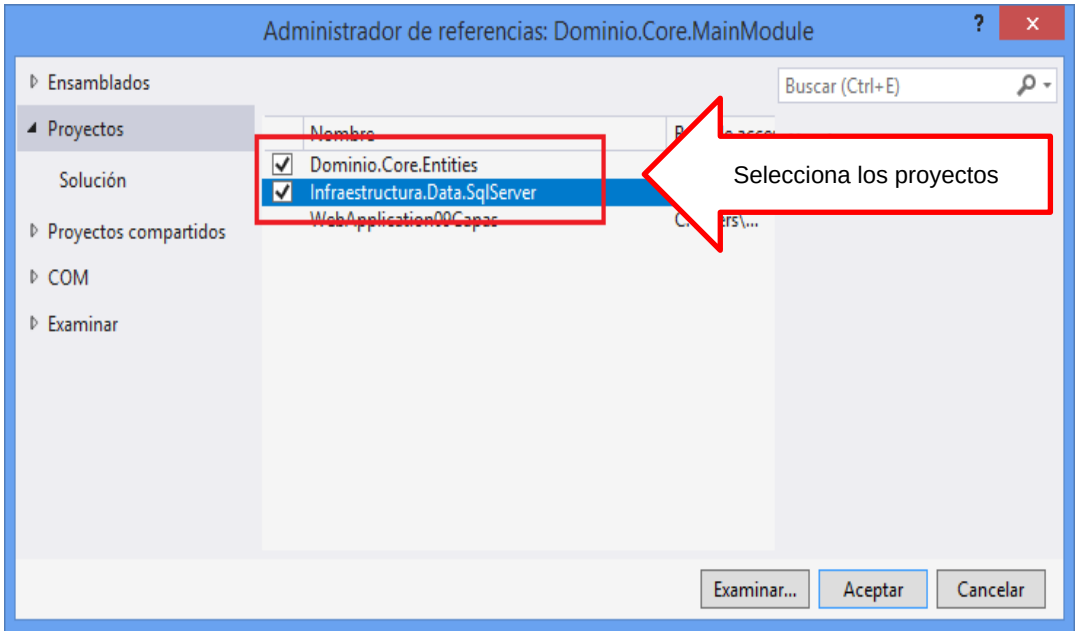
A continuación, selecciona la opción Proyectos, marcar el proyecto Dominio.Core.Entities, tal como se muestra





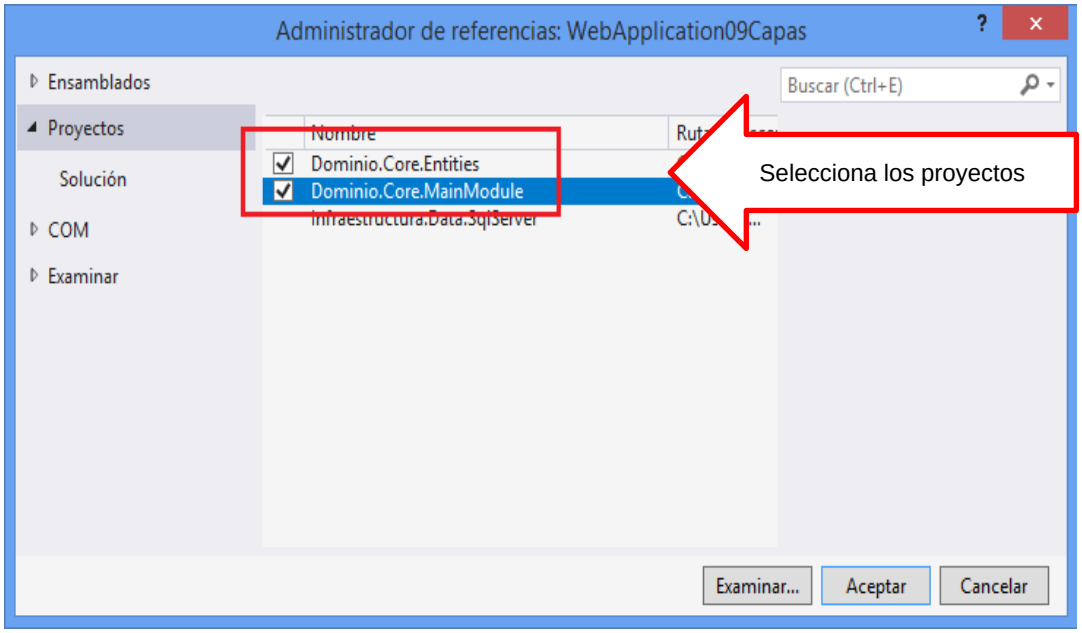
c) Dominio.Core.MainModule

Establecer la referencia: selecciona la opción Proyectos, marcar el proyecto Dominio.Core.Entities e infraestructura.Data.SqlServer, tal como se muestra



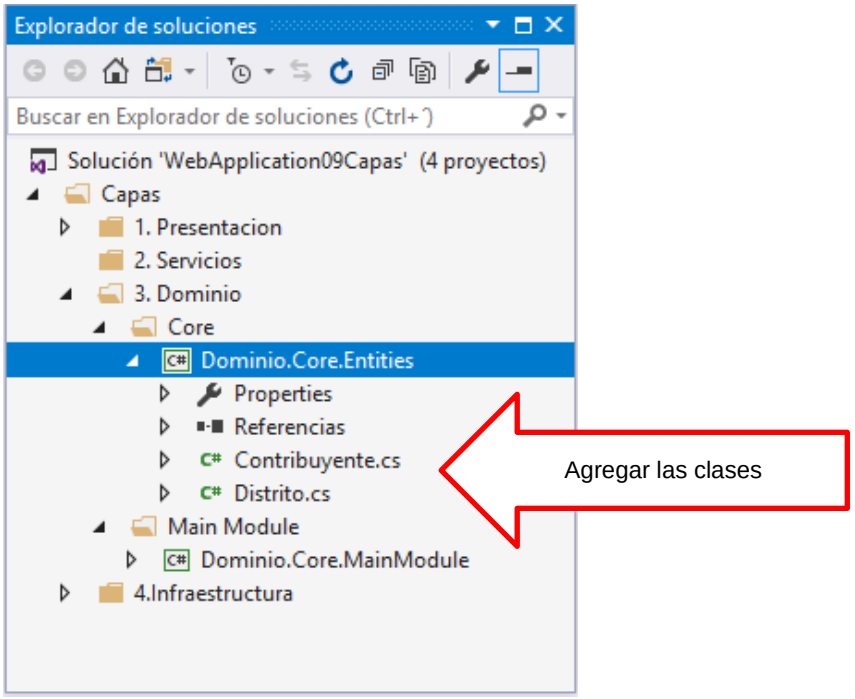
d) Proyecto WebApplication09Capas

Establecer la referencia: selecciona la opción Proyectos, marcar el proyecto Dominio.Core.Entities y Dominio.Core.MainModule, tal como se muestra

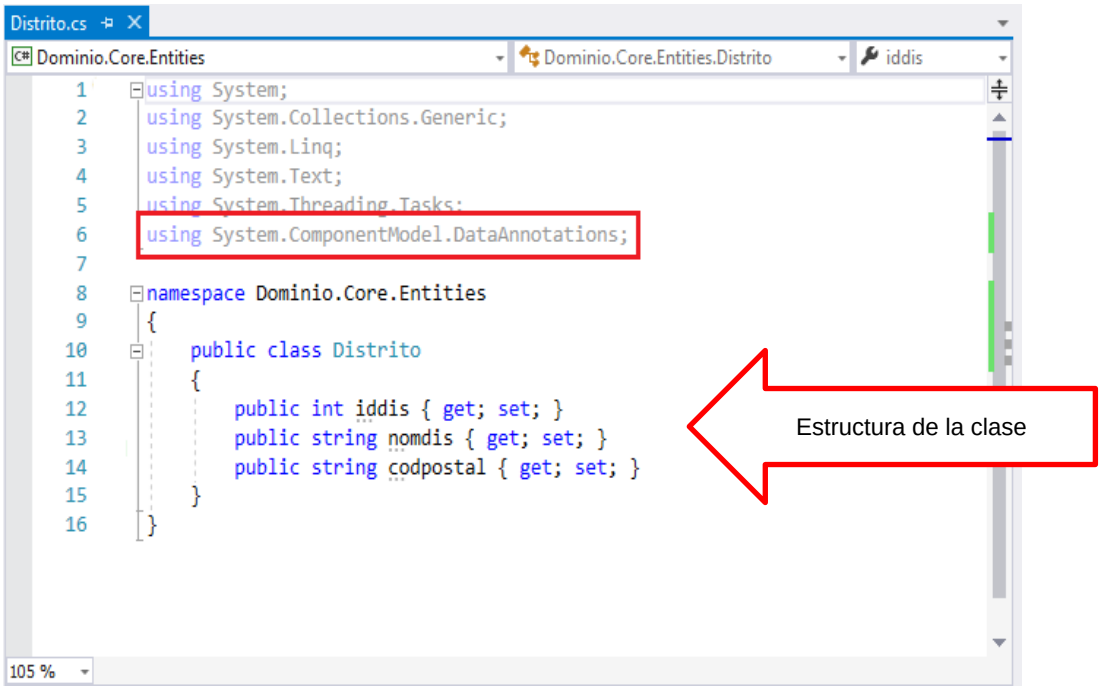


Trabajando con el Proyecto Dominio.Core.Entities

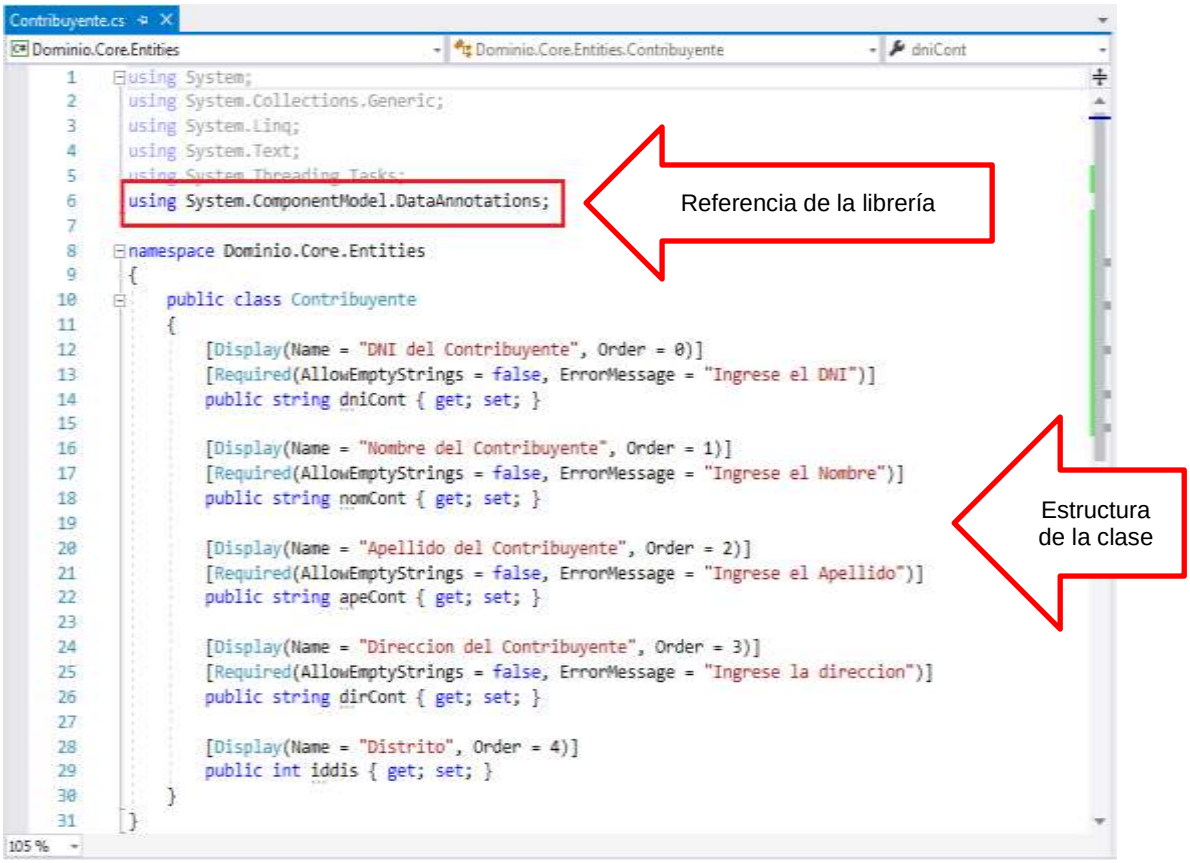
En el proyecto Dominio.Core.Entities agregamos las clases Contribuyente.cs y Distrito.cs, tal como se muestra.



Defina la estructura Distrito.cs, tal como se muestra.



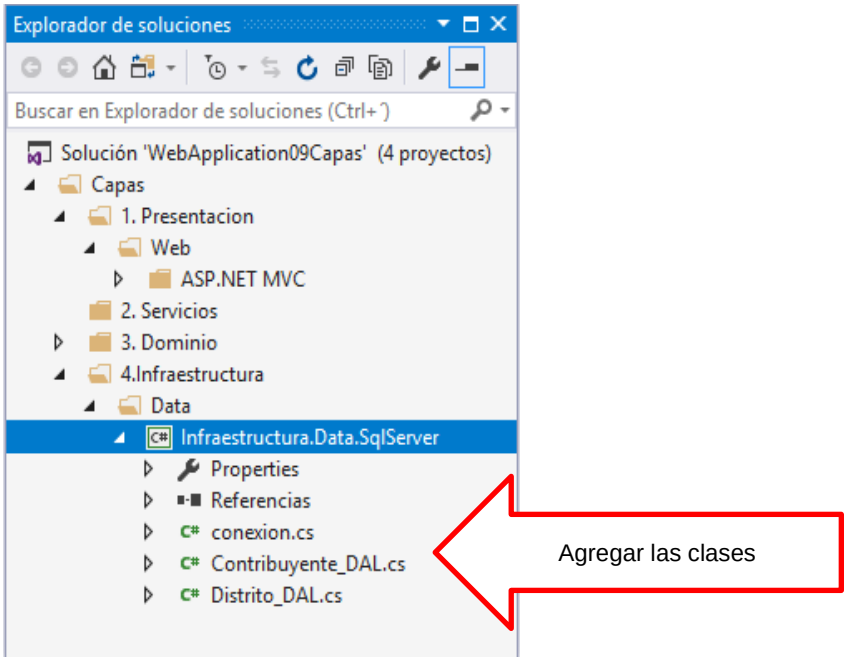
Defina la estructura Contribuyente.cs, la ventaja de este esquema es que al crear la clase podemos poner los atributos de validación de la misma



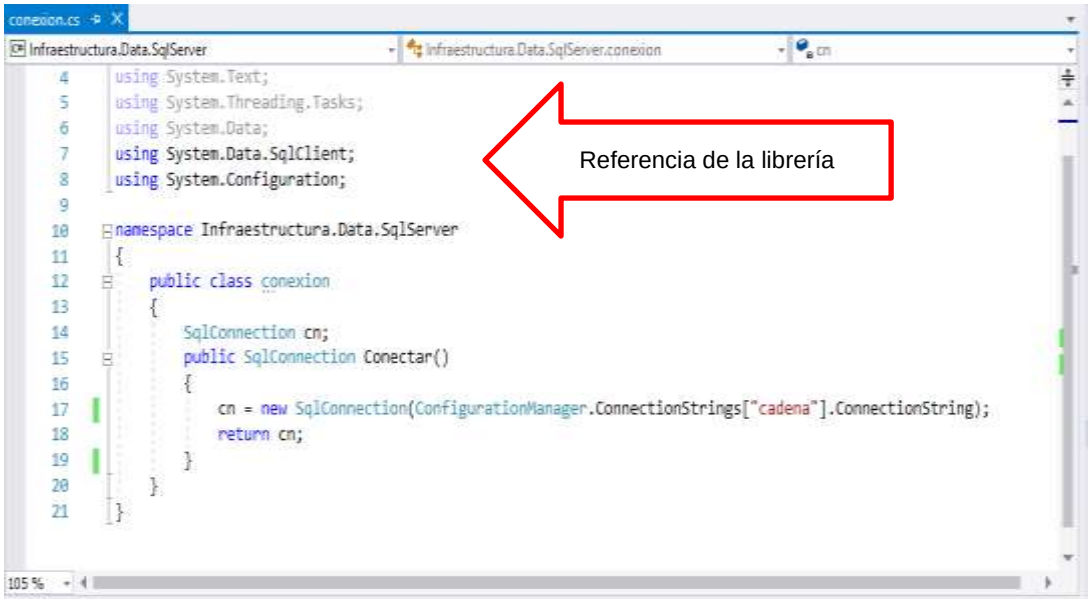

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace Dominio.Core.Entities
{
    public class Contribuyente
    {
        [Display(Name = "DNI del Contribuyente", Order = 0)]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el DNI")]
        public string dniCont { get; set; }

        [Display(Name = "Nombre del Contribuyente", Order = 1)]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Nombre")]
        public string nomCont { get; set; }

        [Display(Name = "Apellido del Contribuyente", Order = 2)]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese el Apellido")]
        public string apeCont { get; set; }

        [Display(Name = "Direccion del Contribuyente", Order = 3)]
        [Required(AllowEmptyStrings = false, ErrorMessage = "Ingrese la direccion")]
        public string dirCont { get; set; }

        [Display(Name = "Distrito", Order = 4)]
        public int iddis { get; set; }
    }
}
```

Trabajando con el Proyecto Infraestructura.Data.SqlServer

En el proyecto Infraestructura.Data.SqlServer agregamos las clases conexión.cs, Contribuyente_DAL.cs y Distrito_DAL.cs, tal como se muestra

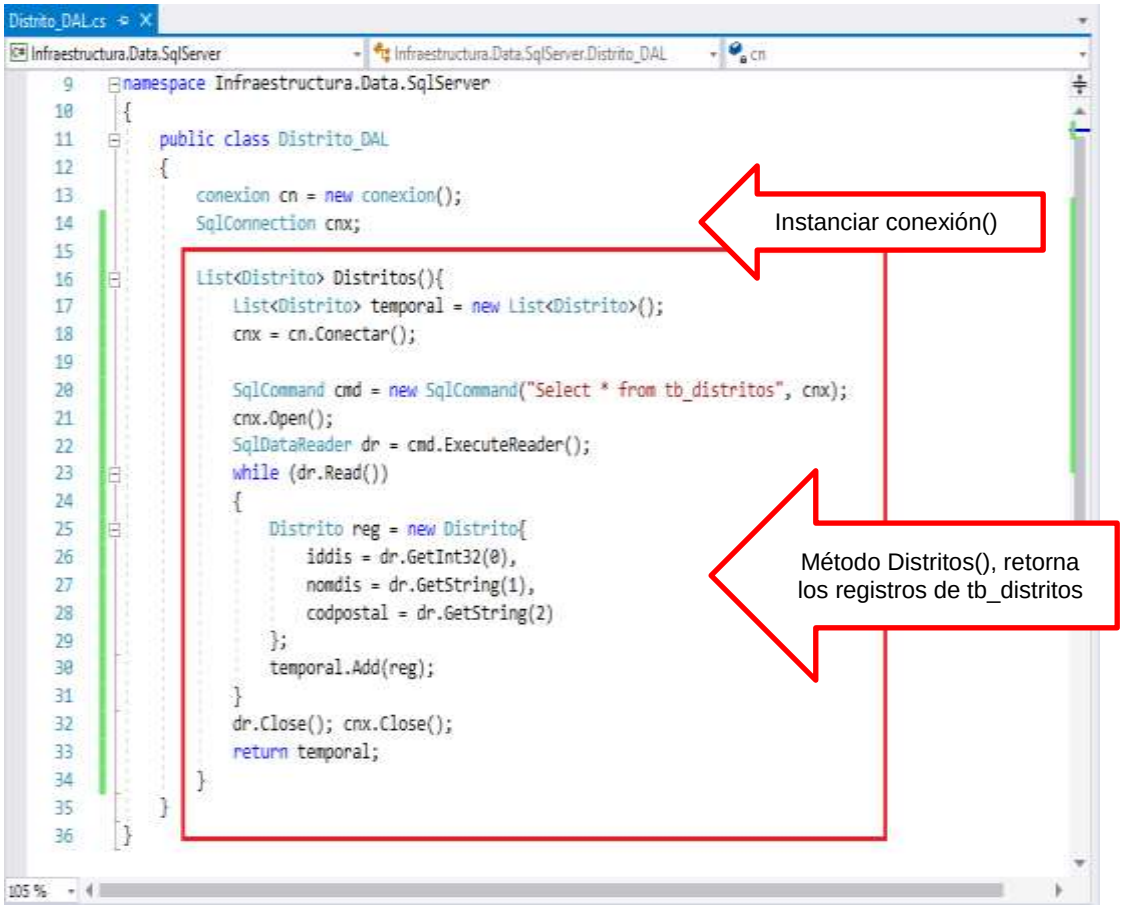


Defina la clase **conexión.cs**, tal como se muestra. Agrega el método Conectar(), el cual retorna la conexión de la base de datos Negocios2018. En el Web.config publicar la conexión a la base de datos.



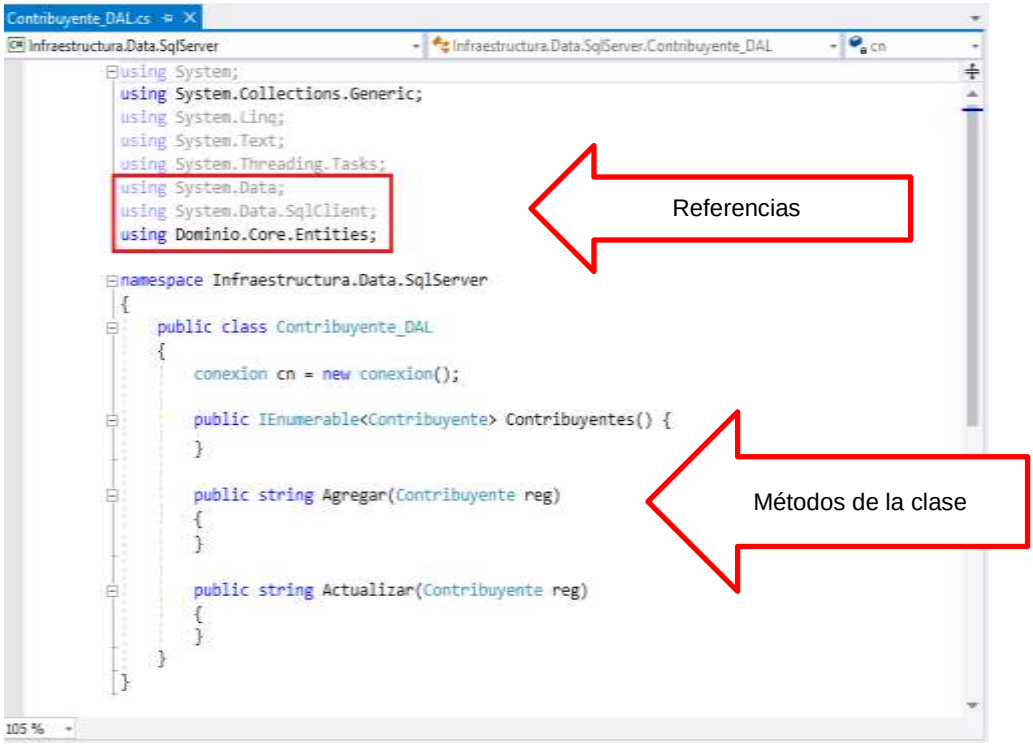

```
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace Infraestructura.Data.SqlServer
{
    public class conexion
    {
        SqlConnection cn;
        public SqlConnection Conectar()
        {
            cn = new SqlConnection(ConfigurationManager.ConnectionStrings["cadena"].ConnectionString);
            return cn;
        }
    }
}
```

Defina la clase **Distrito_DAL.cs**, tal como se muestra. Instancia la clase conexión llamada **cn**. Agrega el método **Distritos**(), el cual retorna los registros de la tabla tb_distritos



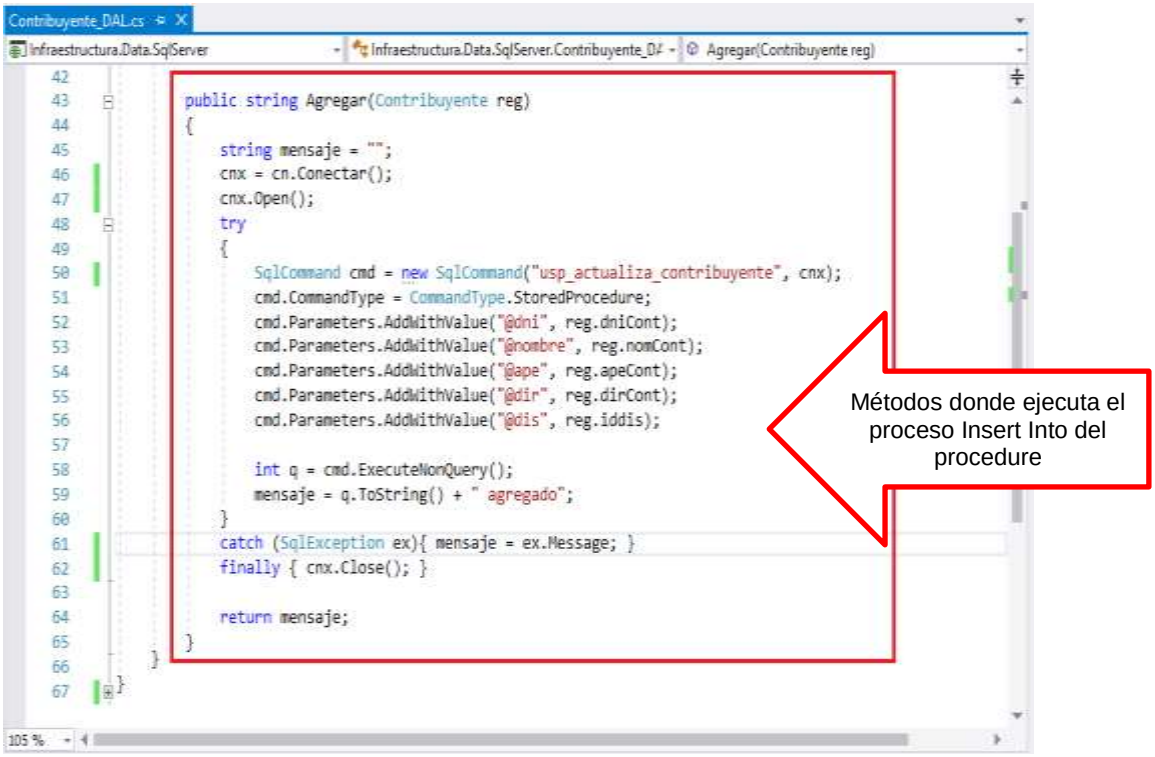

```
namespace Infraestructura.Data.SqlServer
{
    public class Distrito_DAL
    {
        conexion cn = new conexion();
        SqlConnection cnx;

        List<Distrito> Distritos(){
            List<Distrito> temporal = new List<Distrito>();
            cnx = cn.Conectar();

            SqlCommand cmd = new SqlCommand("Select * from tb_distritos", cnx);
            cnx.Open();
            SqlDataReader dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                Distrito reg = new Distrito{
                    iddis = dr.GetInt32(0),
                    nomdis = dr.GetString(1),
                    codpostal = dr.GetString(2)
                };
                temporal.Add(reg);
            }
            dr.Close(); cnx.Close();
            return temporal;
        }
    }
}
```

Defina la clase **Contribuyente_DAL.cs**, tal como se muestra.
Instancia la clase conexión llamada **cn**. Agrega los método **Contribuyentes**(), donde retorna los registros de tb_contribuyentes; Agregar(Contribuyente reg) y Actualizar (Contribuyente reg) los cuales ejecutan el procedure del Merge.



Codifique el método Contribuyentes(), ejecutando el procedimiento almacenado, tal como se muestra

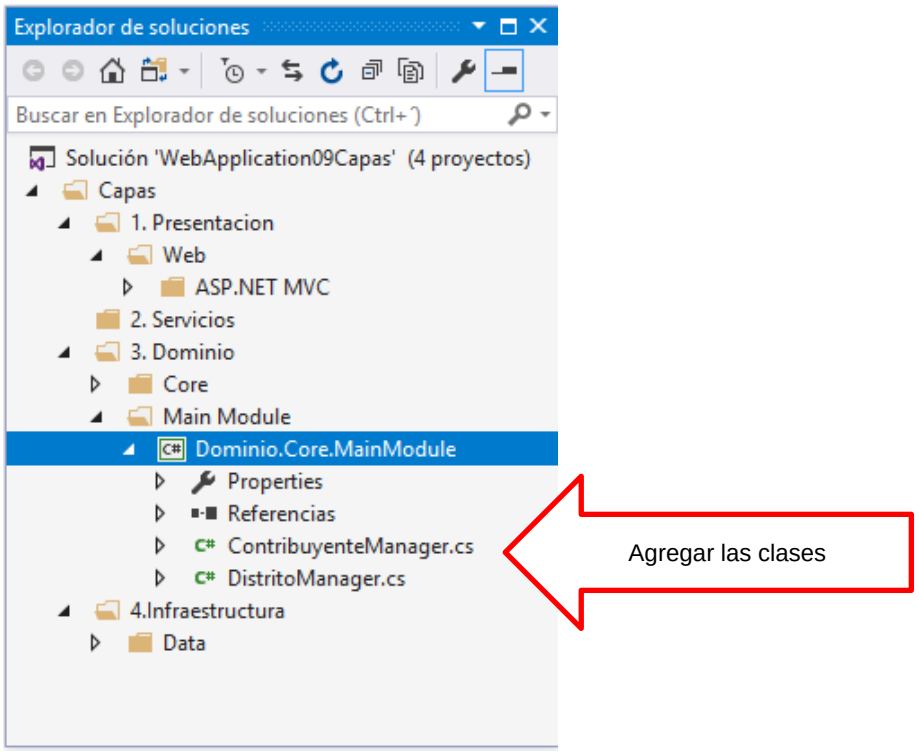
Codifique el método Agregar(), tal como se muestra



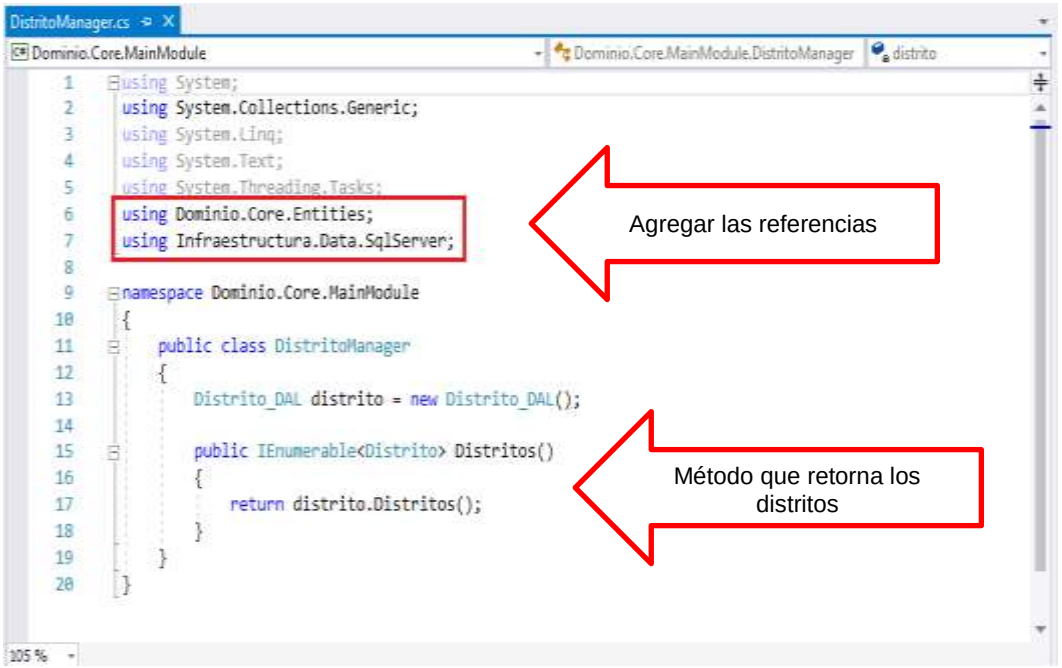
Codifique el método Actualizar(), tal como se muestra

Trabajando con el Proyecto Dominio.Core.MainModule

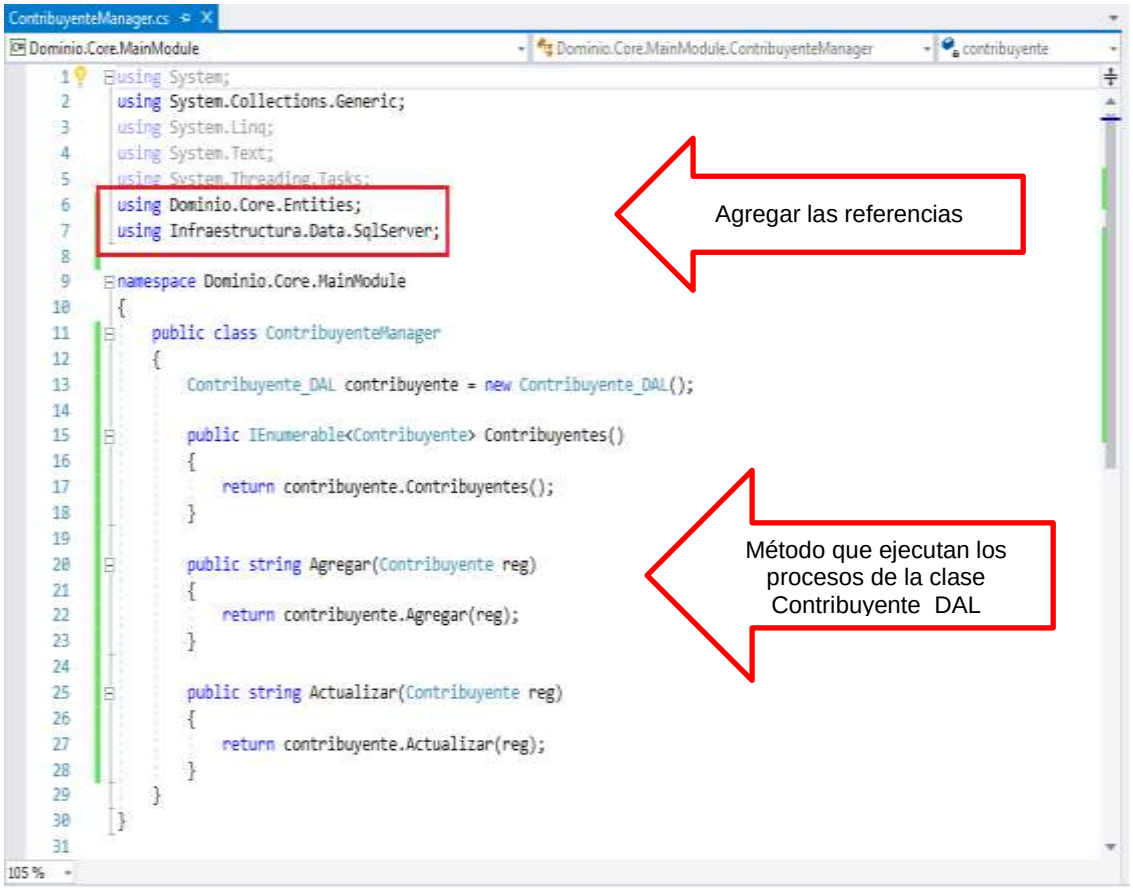
En el proyecto agregamos las clases ContribuyenteManager.cs y DistritoManager.cs, tal como se muestra



Defina la clase **DistritoManager.cs**, tal como se muestra. Instancia la clase Distrito_DAL llamada **distrito**. Agrega el método **Distritos**(), el cual ejecuta el método Distritos() de la instancia de Distrito_DAL, tal como se muestra



Defina la clase **ContribuyenteManager.cs**, tal como se muestra.
Instancia la clase Contribuyente_DAL llamada **contribuyente**.
Agrega los métodos definimos en la clase, los cuales ejecutarán los métodos de la clase Contribuyente_DAL, retornando los resultados



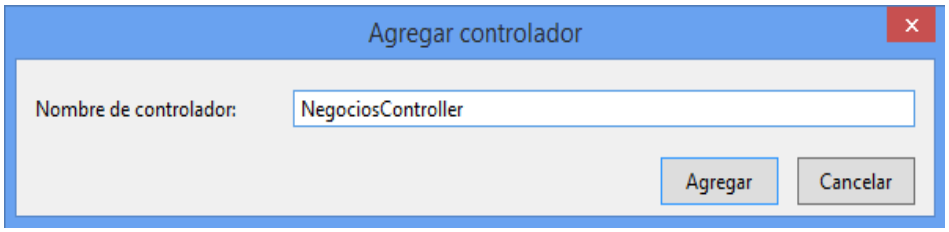
Trabajando con la Aplicación Web ASP.NET MVC

Creando un Controlador

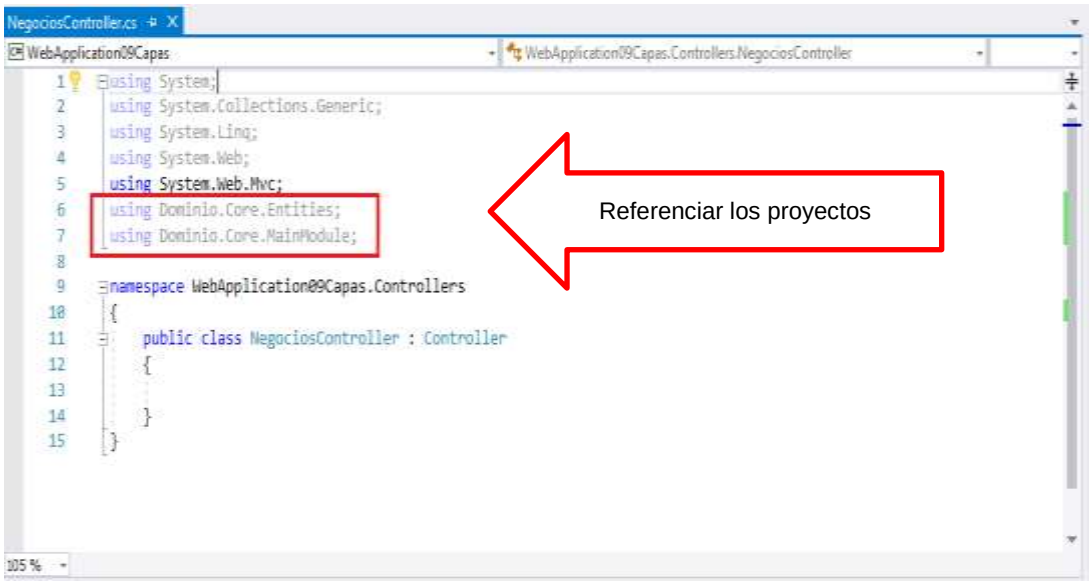
A continuación, creamos un controlador: Desde la carpeta **Controllers**, selecciona la opción **Agregar**, **Controlador**.

Selecciona el scaffold, Controlador de MVC5: en blanco, presiona el botón AGREGAR

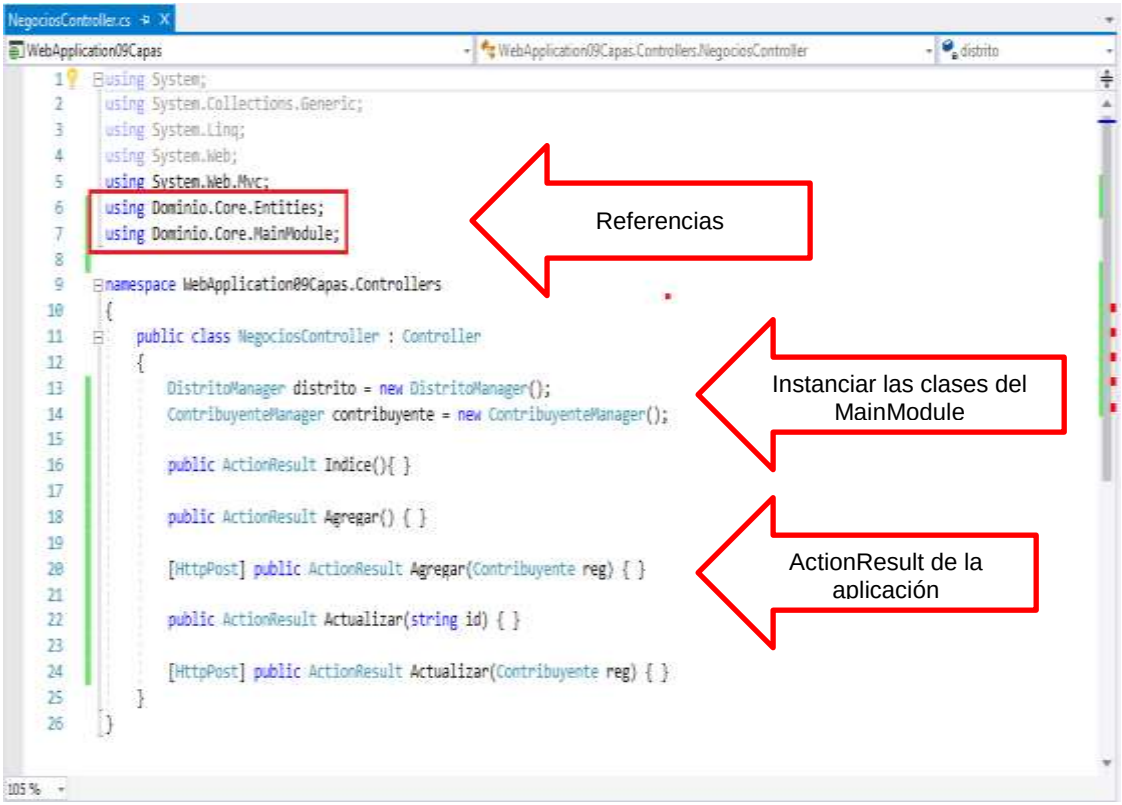
Asigne el nombre del controlador: NegociosController, tal como se muestra



Referenciar los proyectos, tal como se muestra.



En el controlador, instanciar las clases del proyecto MainModule, para ejecutar los procesos.
Defina los ActionResult de la aplicación, tal como se muestra.

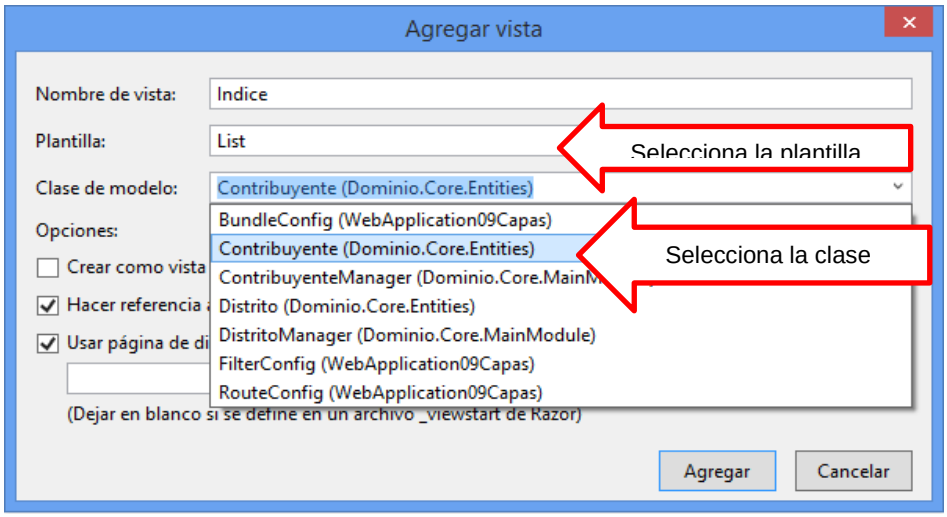


Trabajando con el ActionResult Indice

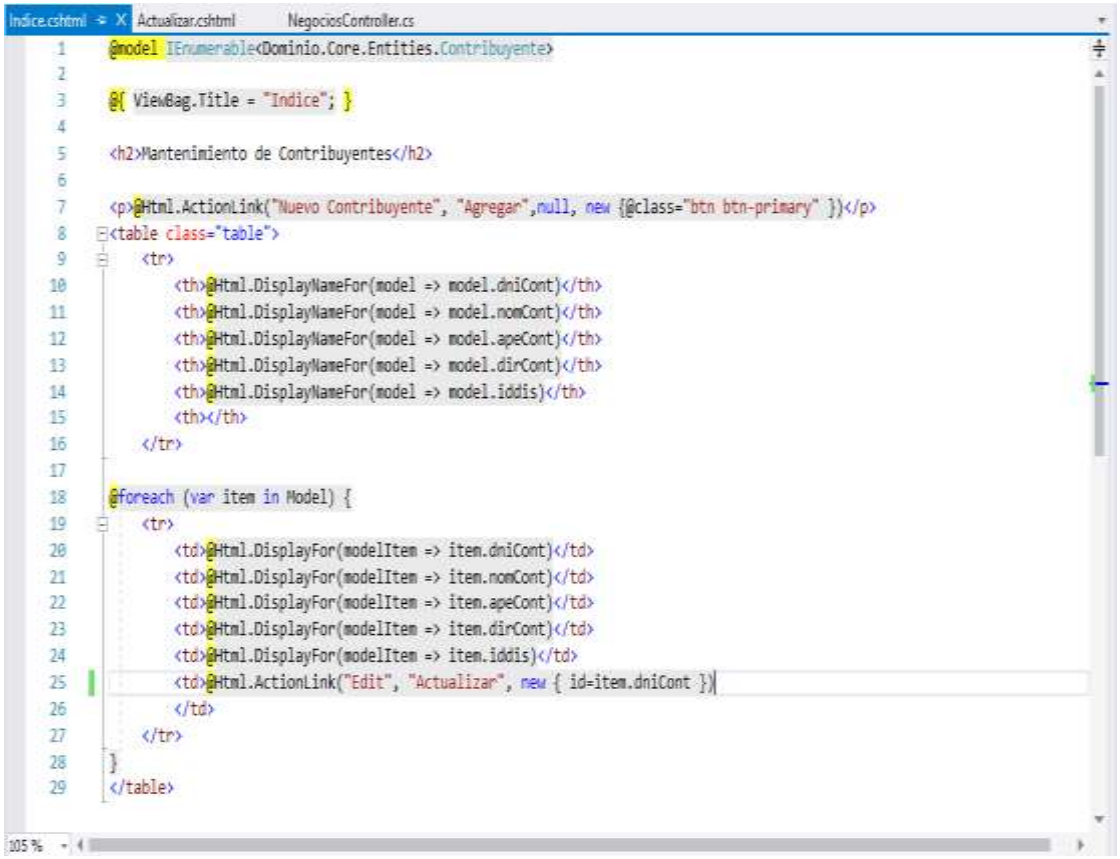
Defina el ActionResult **Indice**(), el cual ejecuta el método Contribuyentes(), enviando a la vista la lista de los registros de tb_contribuyentes.

Agregando la Vista.

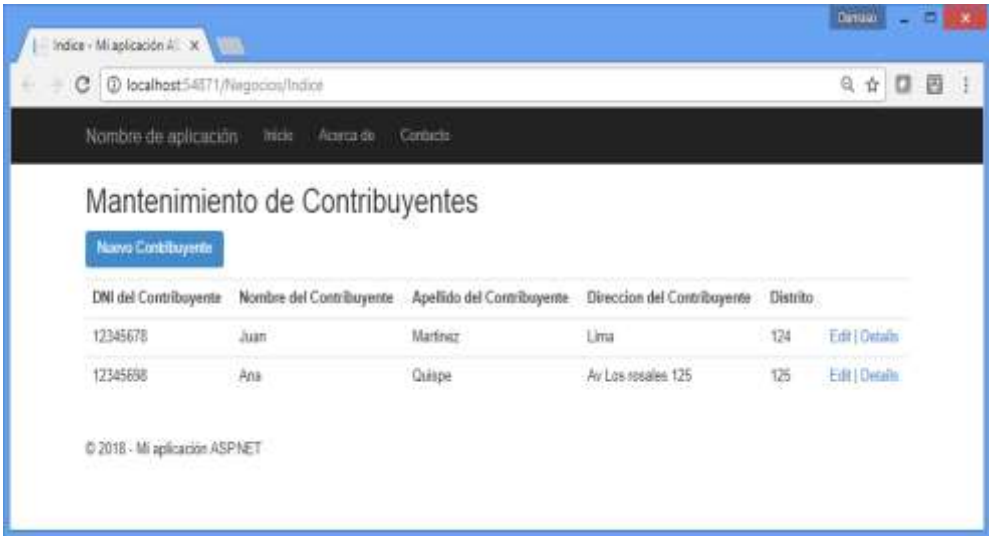
En el ActionResult, agrega la vista.
En dicha ventana, selecciona la **plantilla**, la cual será **List**; y la **clase de modelo** la cual es **Contribuyente (Core.Entities)**, tal como se muestra



Modifica la Vista:
1. Agregar un ActionLink el cual ejecutará el ActionResult Agregar.
2. Modifica la estructura de la página, tal como se muestra



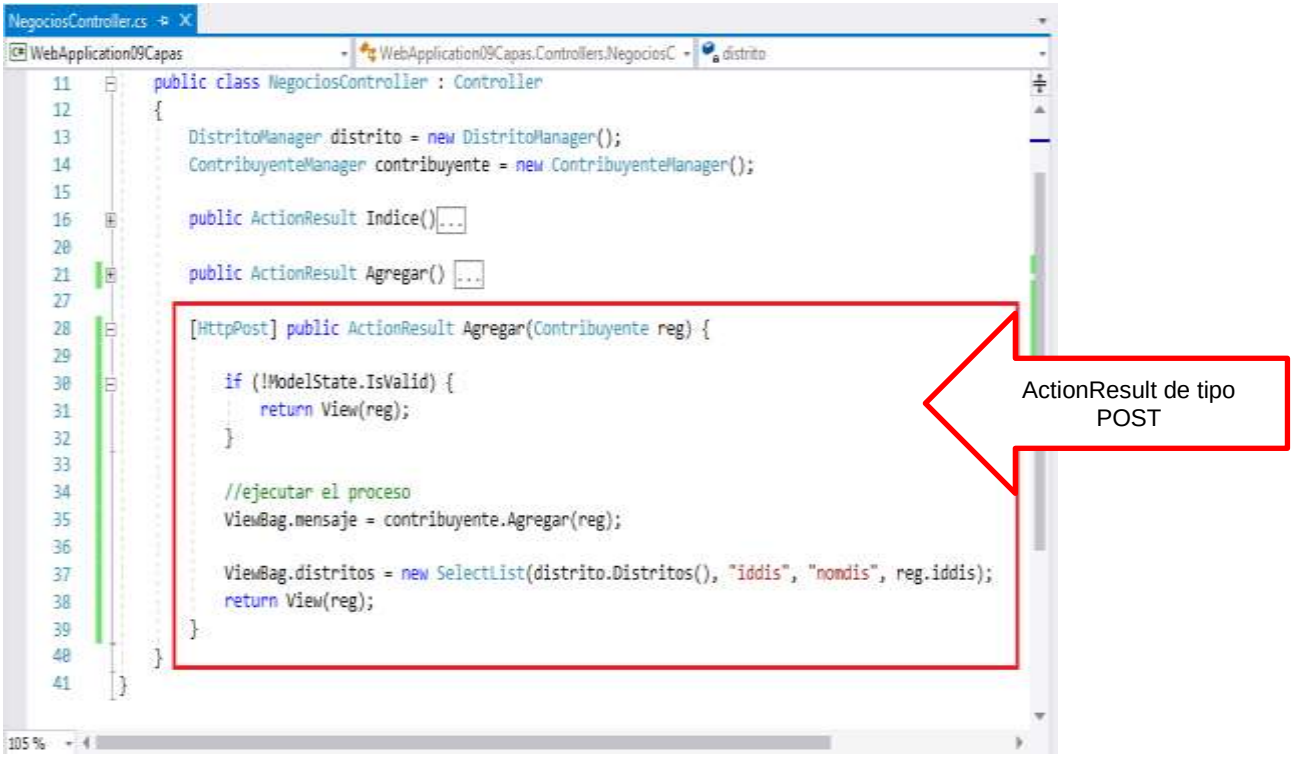
Ejecutamos la Vista, visualizamos los registros de la tabla tb_contribuyentes.



Trabajando con el ActionResult Agregar

El **ActionResult Agregar** de tipo GET, almaceno los distritos en el ViewBag.distritos; envía a la vista un nuevo registro de tb_contribuyente, tal como se muestra.

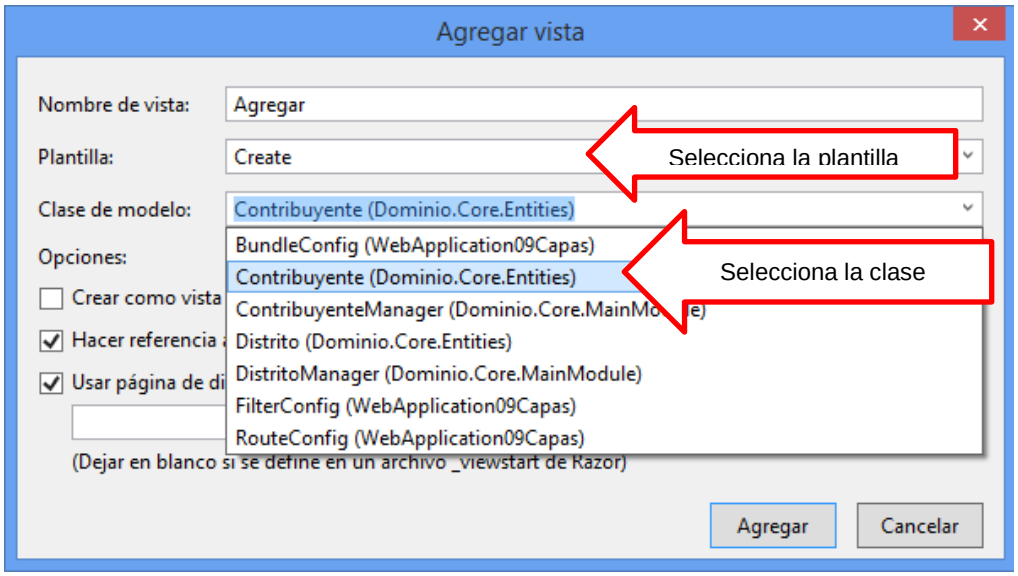
El **ActionResult Agregar** de tipo POST, recibe los datos de la vista, los valida; si está OK, procederá a ejecutar el método AGREGAR, retornando un mensaje y los datos del registro agregado, tal como se muestra



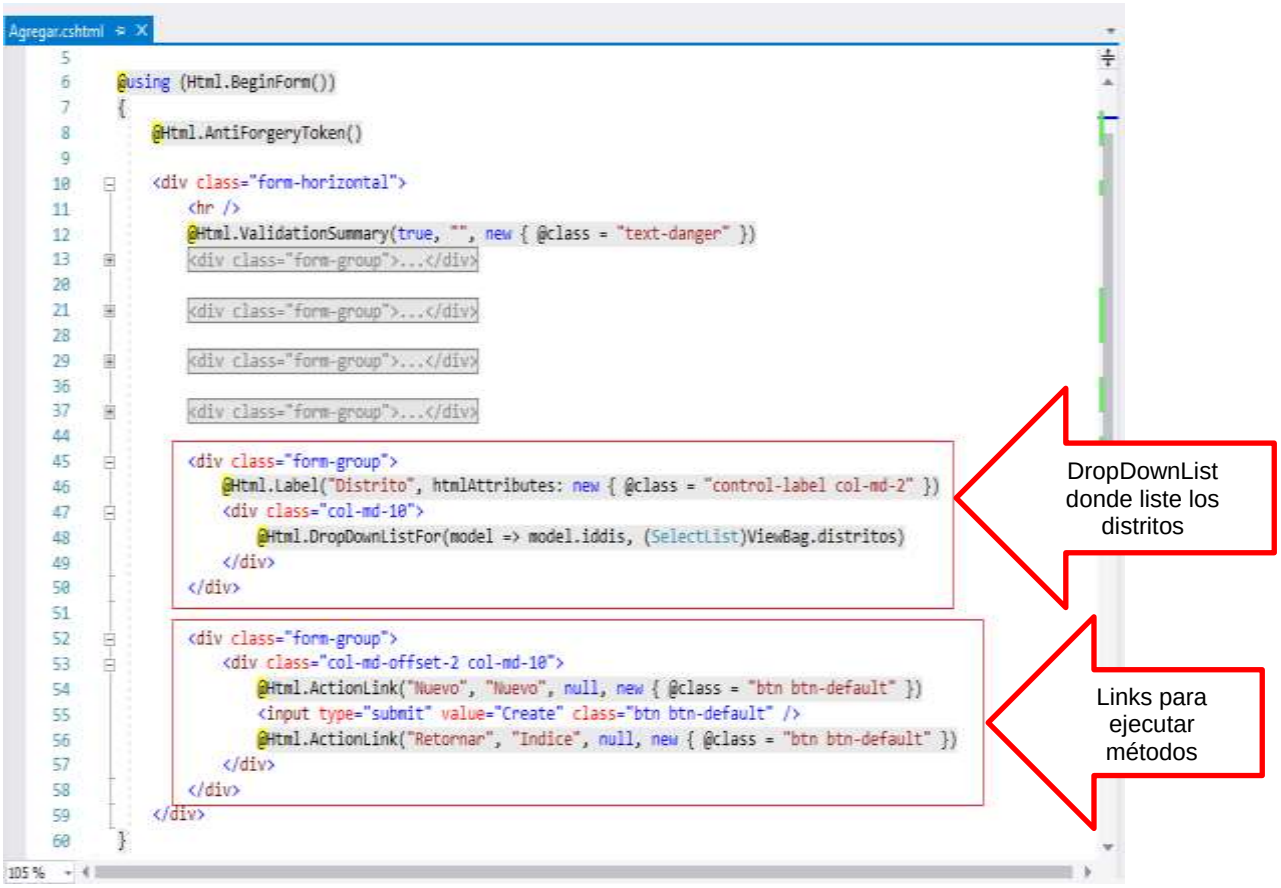
Agregando la Vista.

En el ActionResult, agrega la vista.

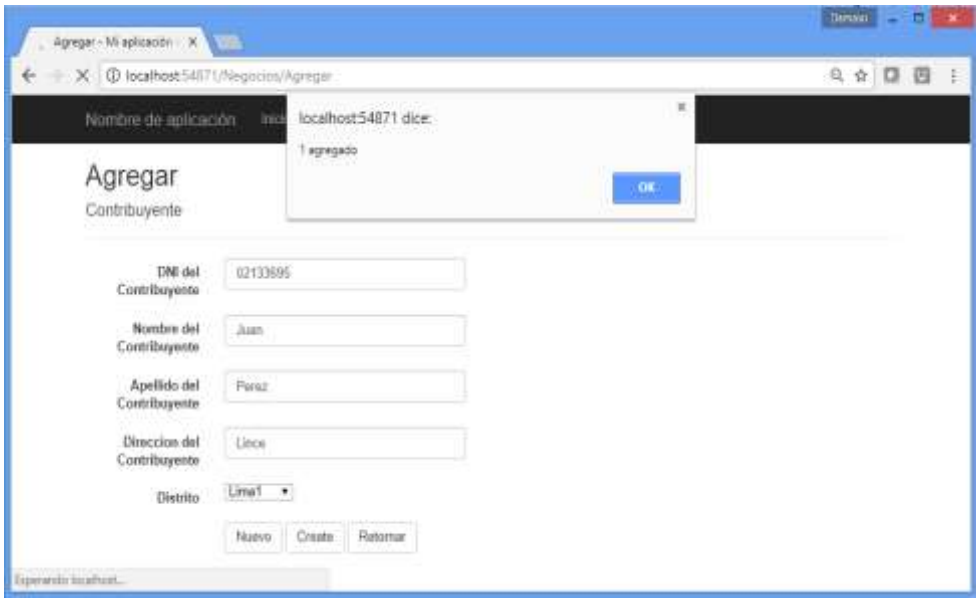
En dicha ventana, selecciona la **plantilla**, la cual será **Create**; y la **clase de modelo** la cual es **Contribuyente**, tal como se muestra



Modifique el diseño de la vista: en el campo iddis, cambiar por DropDownList, para listar los distritos; agrupe y diseñe los Links de los procesos: Nuevo, Create, Retornar, tal como se muestra

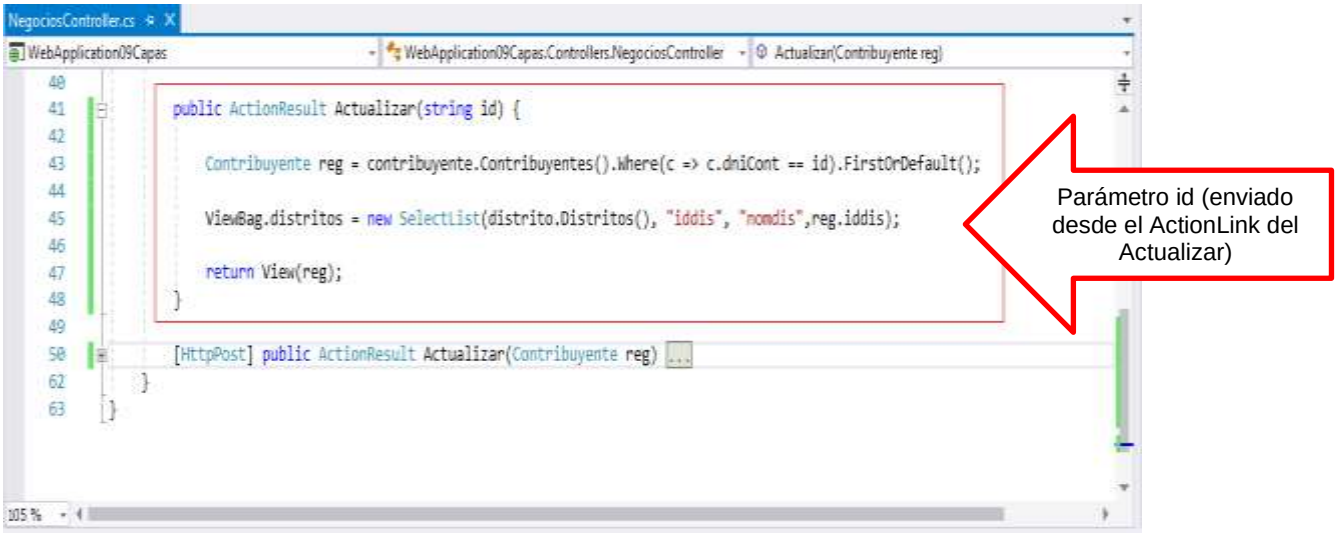


Ejecuta la Vista, ingresa los datos, al presionar el botón CREATE, agregamos un registro y visualizamos un mensaje.
Para retornar, presione el botón RETORNAR



Trabajando con el ActionResult Actualizar

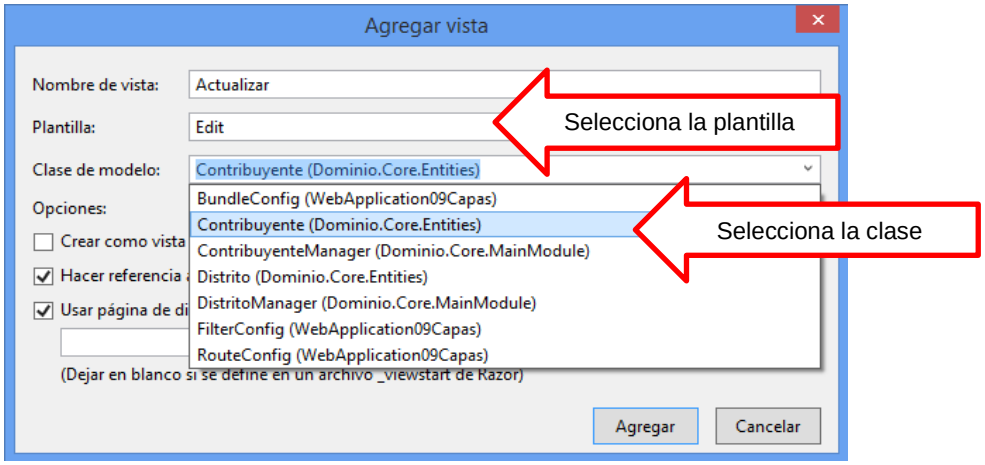
Codifique el **ActionResult Actualizar** (string id) método GET, donde filtra y envía el registro de la tabla tb_contribuyente filtrado por su campo dniCont; además enviamos a la vista la lista de distritos a través del ViewBag.distritos, tal como se muestra



El **ActionResult** Edit de tipo POST, recibe los datos de la vista, los valida; si está OK, procederá a ejecutar el comando UPDATE, retornando un mensaje y los datos del registro agregado, tal como se muestra

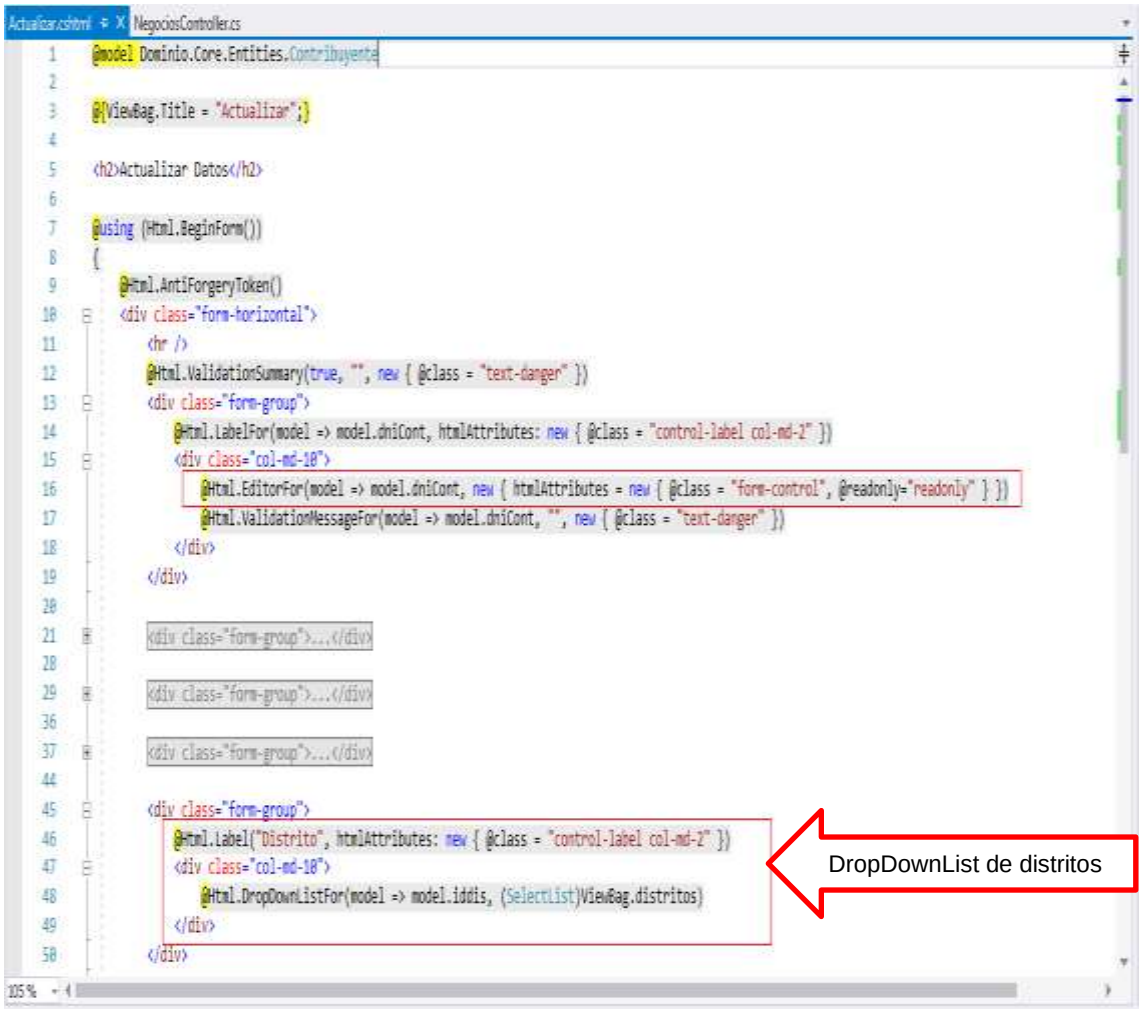
Agregando la Vista.

En el ActionResult, agrega la vista. Selecciona la **plantilla**, la cual será **Edit**; y la **clase de modelo** la cual es **Contribuyente**, tal como se muestra

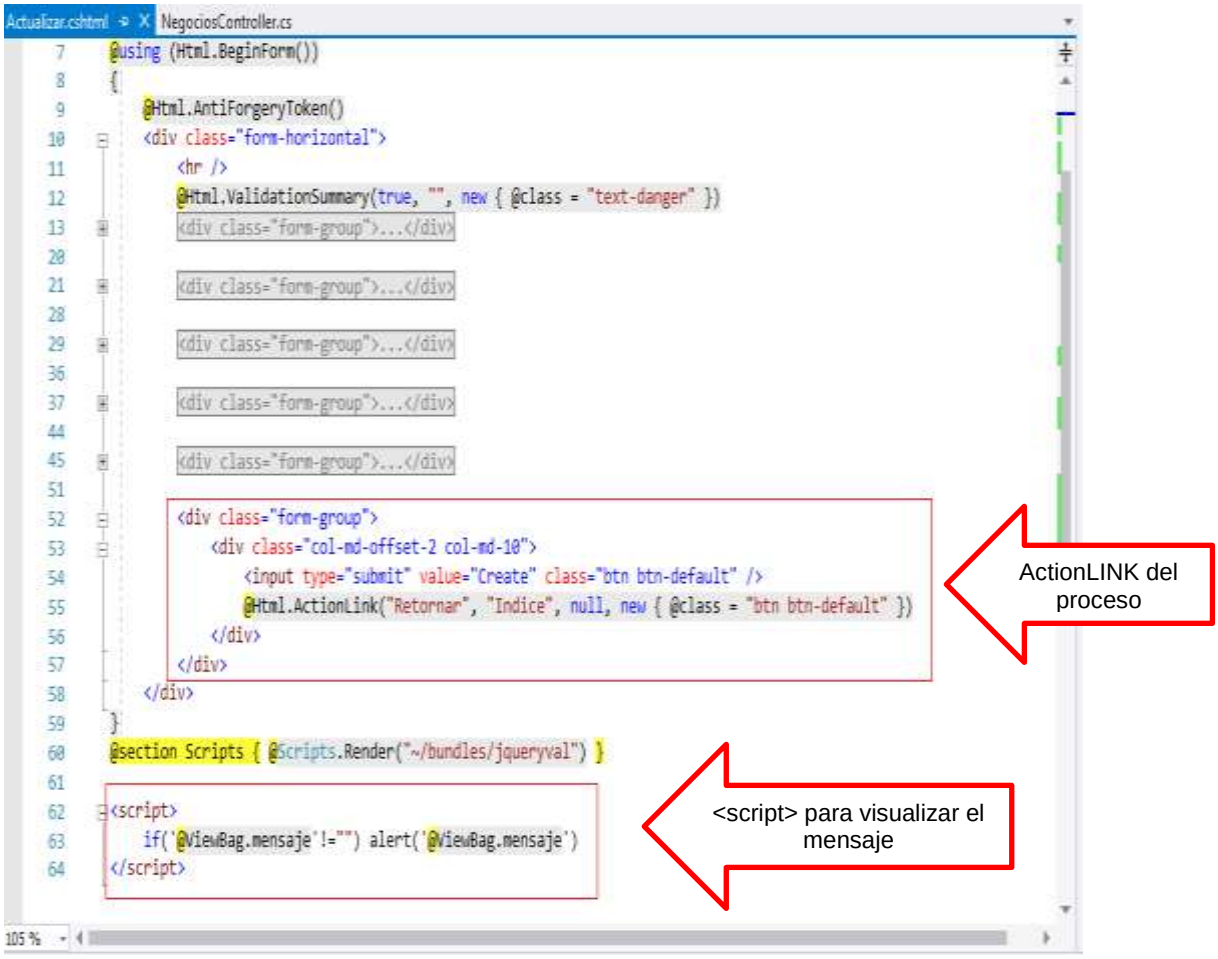


Modifique el diseño de la Vista

Para que el control EditFor del dniCont sea solo lectura, agregar la propiedad @readonly="readonly". Modifique el campo iddis por un DropDownList para listar los registros de tb_distritos, tal como se muestra



Modifique el diseño del ActionLink Retornar, tal como se muestra



Ejecuta la vista, selecciona el Contribuyente, donde visualizamos los datos del registro en la vista Edit. Modifique los datos, al presionar el botón CREATE, se visualiza un mensaje indicando que se ha actualizado el registro.

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

UNIDAD DE APRENDIZAJE

3

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 7: Implementando una aplicación e-commerce (06 horas)

7.1 Introducción
7.2 Proceso del e-commerce
7.3 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

7. Implementando una aplicación e-commerce “N”

7.1 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.

7.2 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.

7.3 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 carito del súper el de nosotros no es necesario que se tome (crearlo) al principio; no mas llegue a nuestro sitio el cliente. Sino que el cliente puede revisar el catalogo de productos y hasta que este listo a comprar se le asignara un carrito donde introducir su compra.

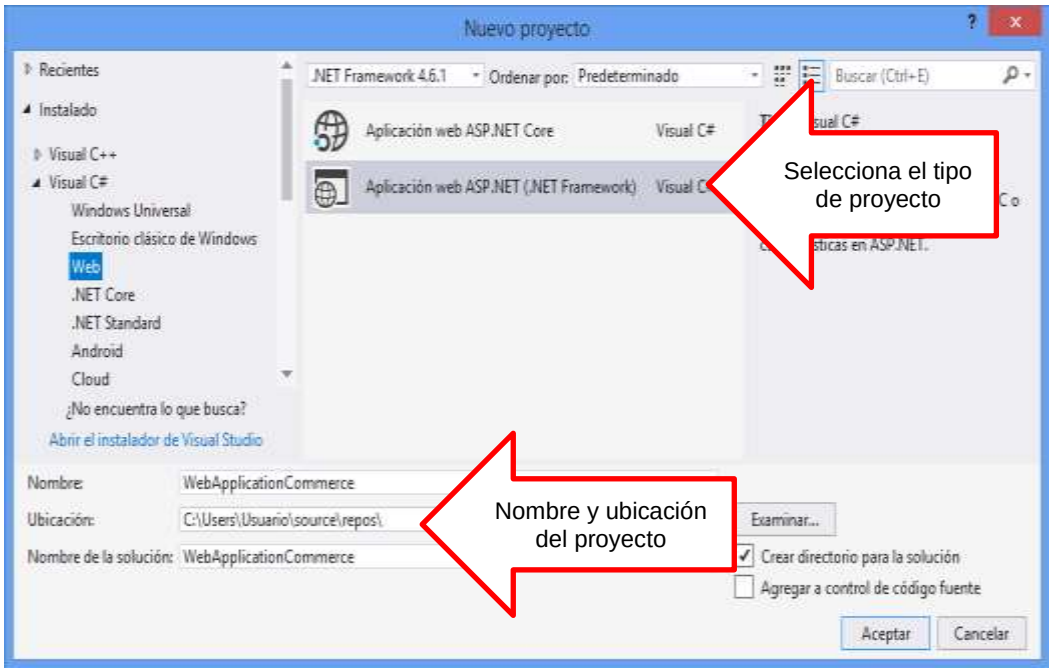
2. Segundo en el carro de supermercado es capaz de contener una gran cantidad de productos a la vez, nuestro carro de compras debe ser capaz de hacer lo mismo.
3. Tercero el carro de supermercado me permite introducir de un productos varios del mismo, el que programaremos debe ser capaz de hacerlo.
4. 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 7.1
E-Commerce

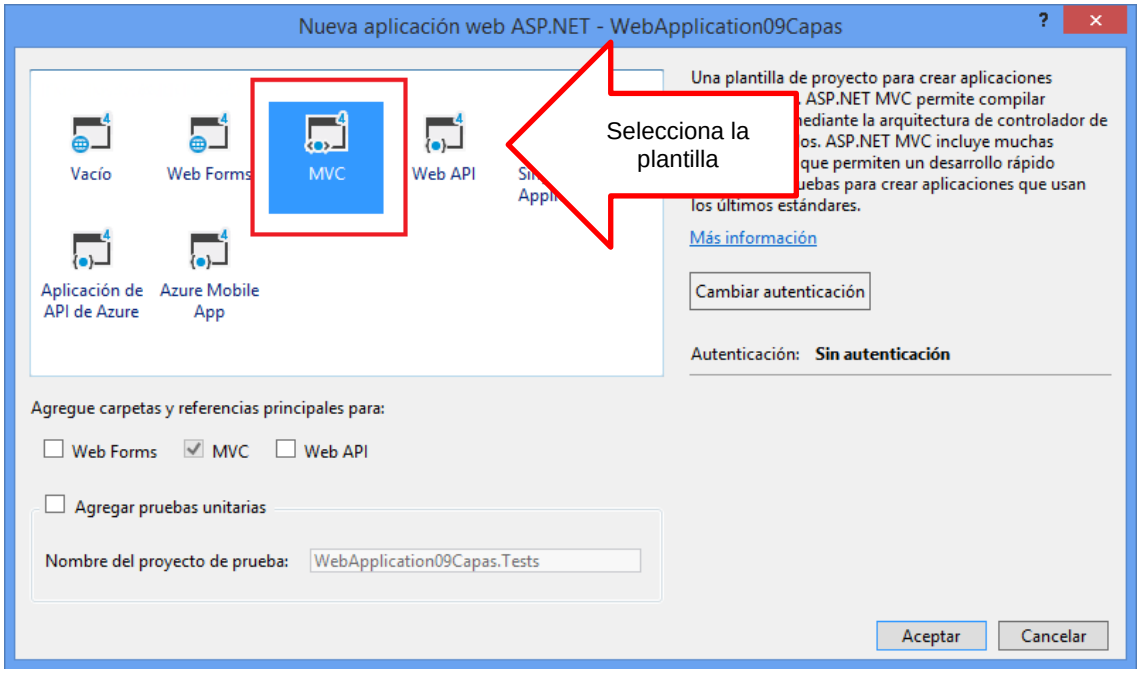
Implemente un proyecto ASP.NET MVC, donde permita implementar un carrito de compras para el registro de los productos seleccionados y el proceso transaccional de la venta.

1. Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR

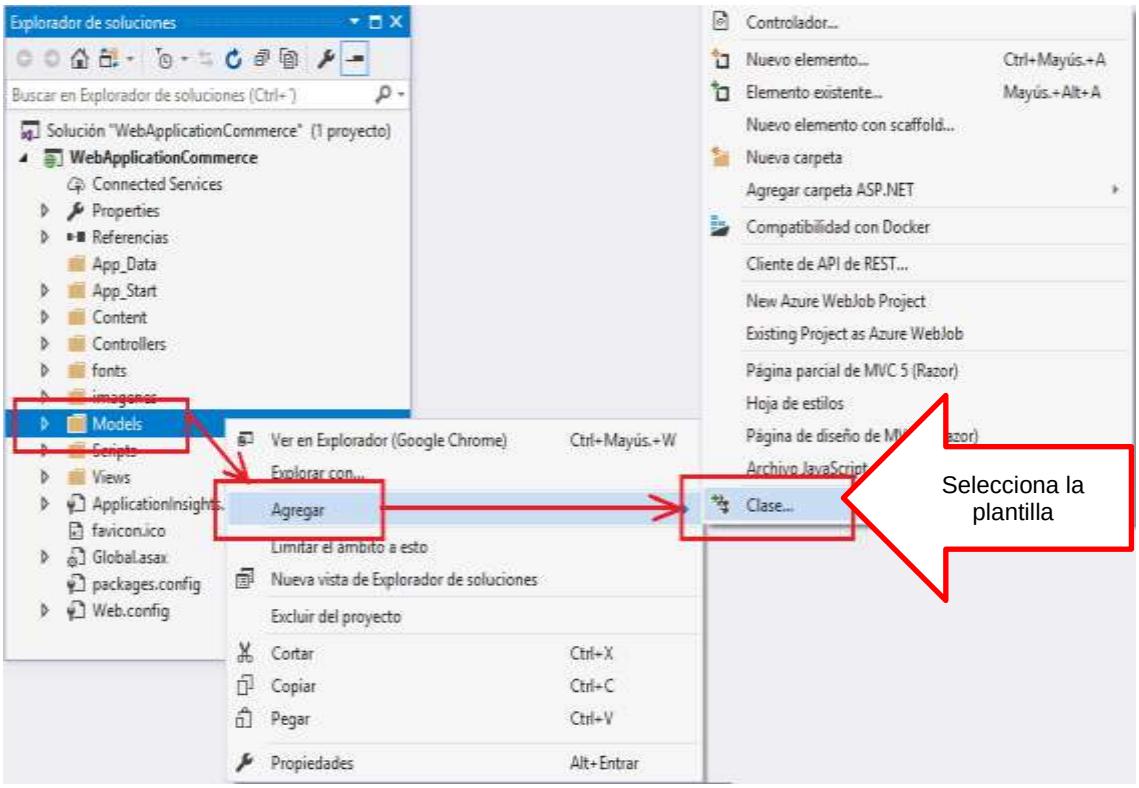


A continuación, seleccionar la plantilla del proyecto MVC.

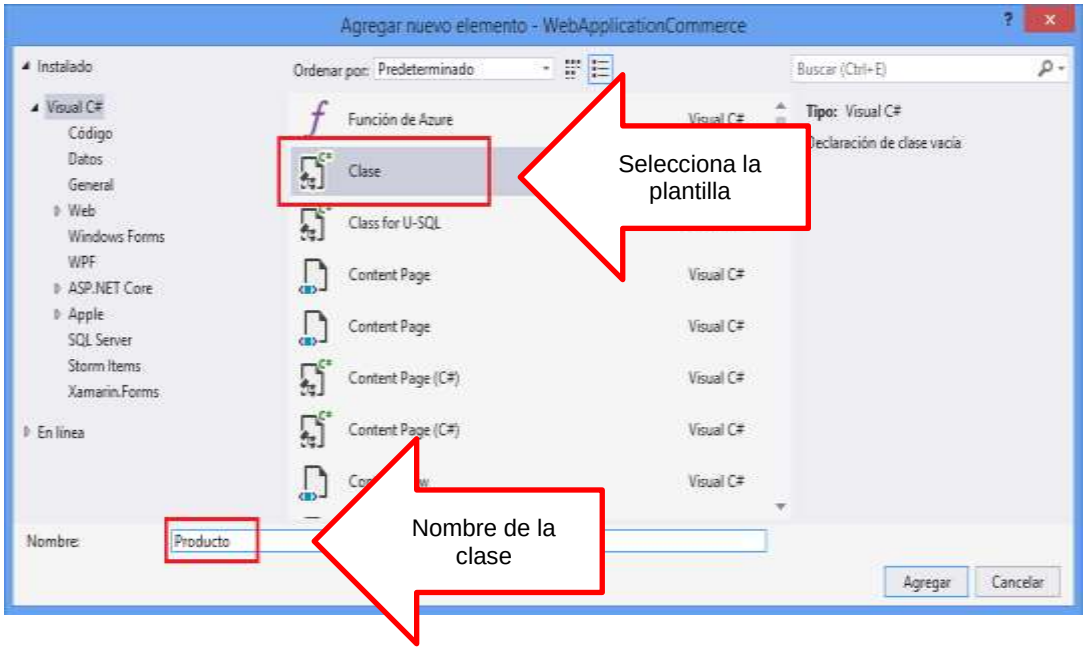


2. Trabajando con la carpeta Models

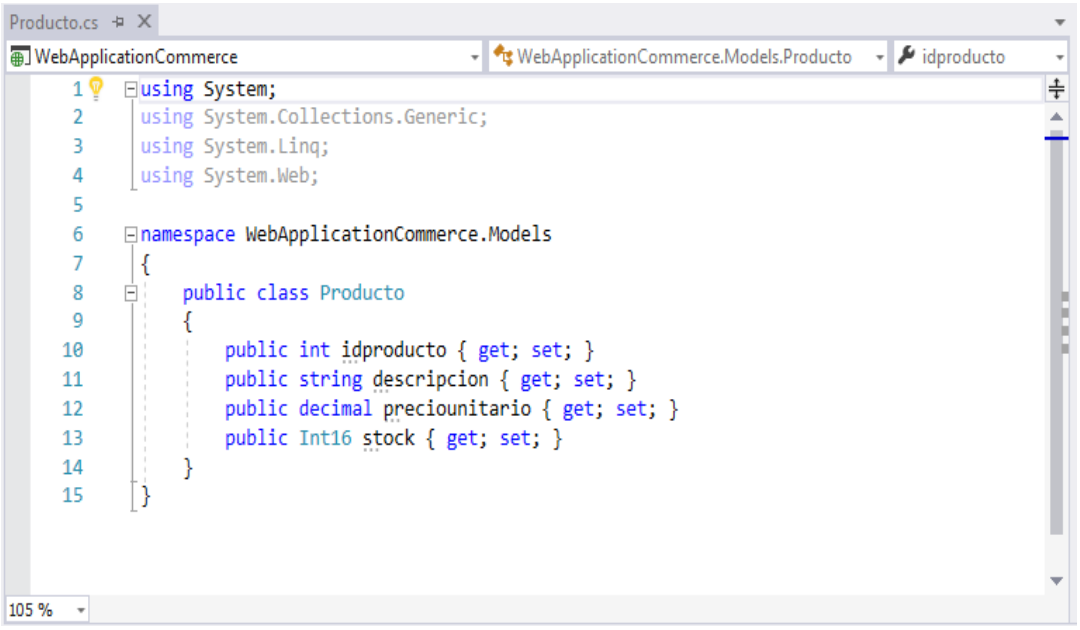
Para iniciar el proceso, defina las clases del proyecto para el manejo de los productos en la página. En la carpeta Models, agrega una clase, tal como se muestra



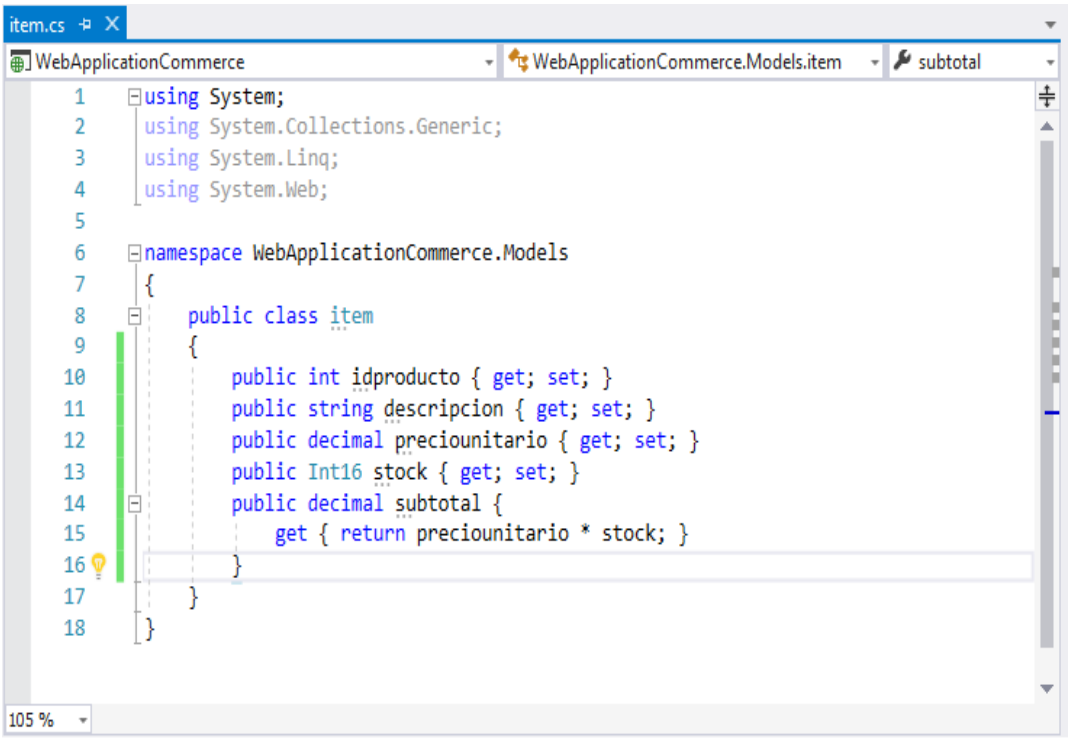
Selecciona el elemento, y asigna el nombre de la clase, tal como se muestra



Defina la estructura de la clase, tal como se muestra

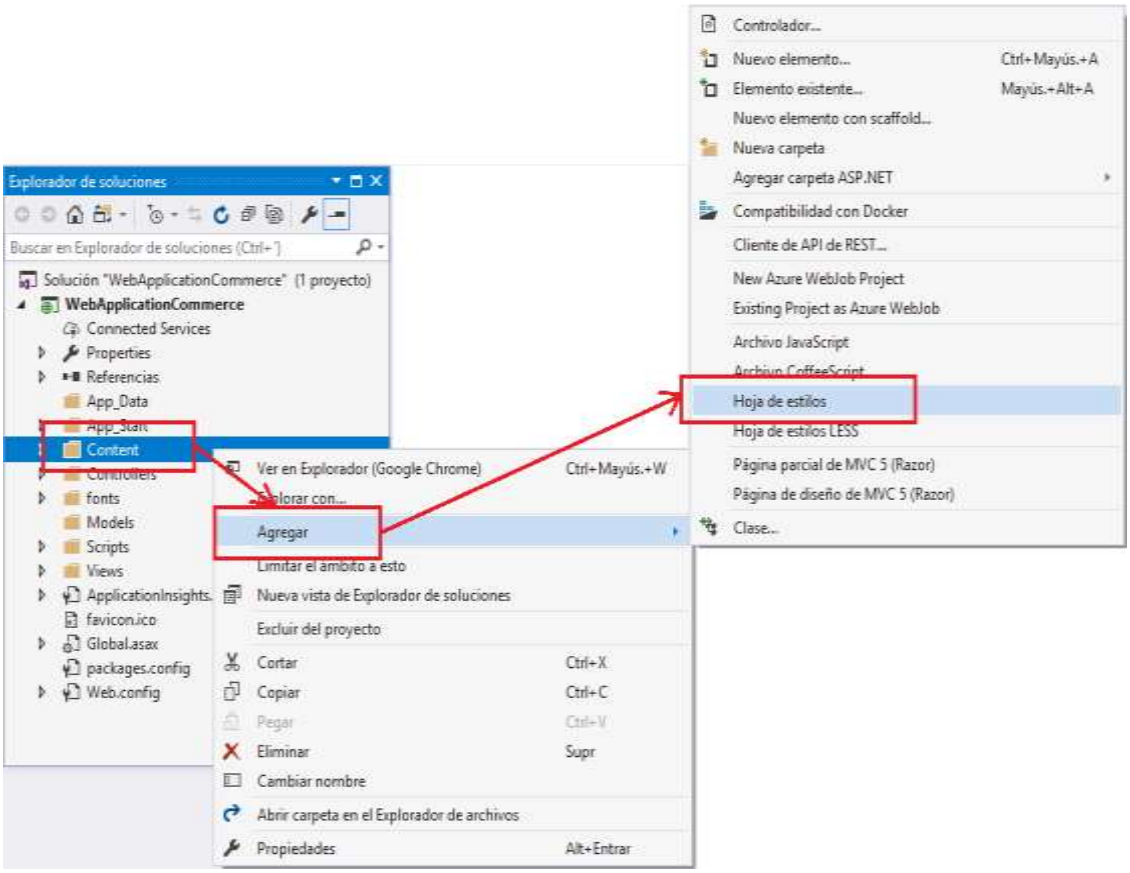


Defina la clase item, la cual almacenara, en una Session, los registros seleccionados del proceso, tal como se muestra

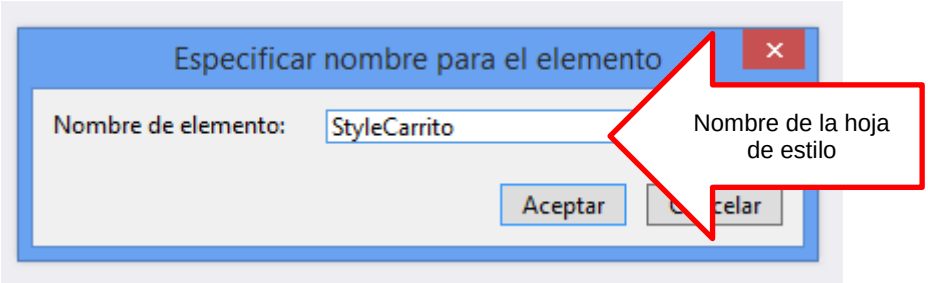


3. Trabajando con el Content

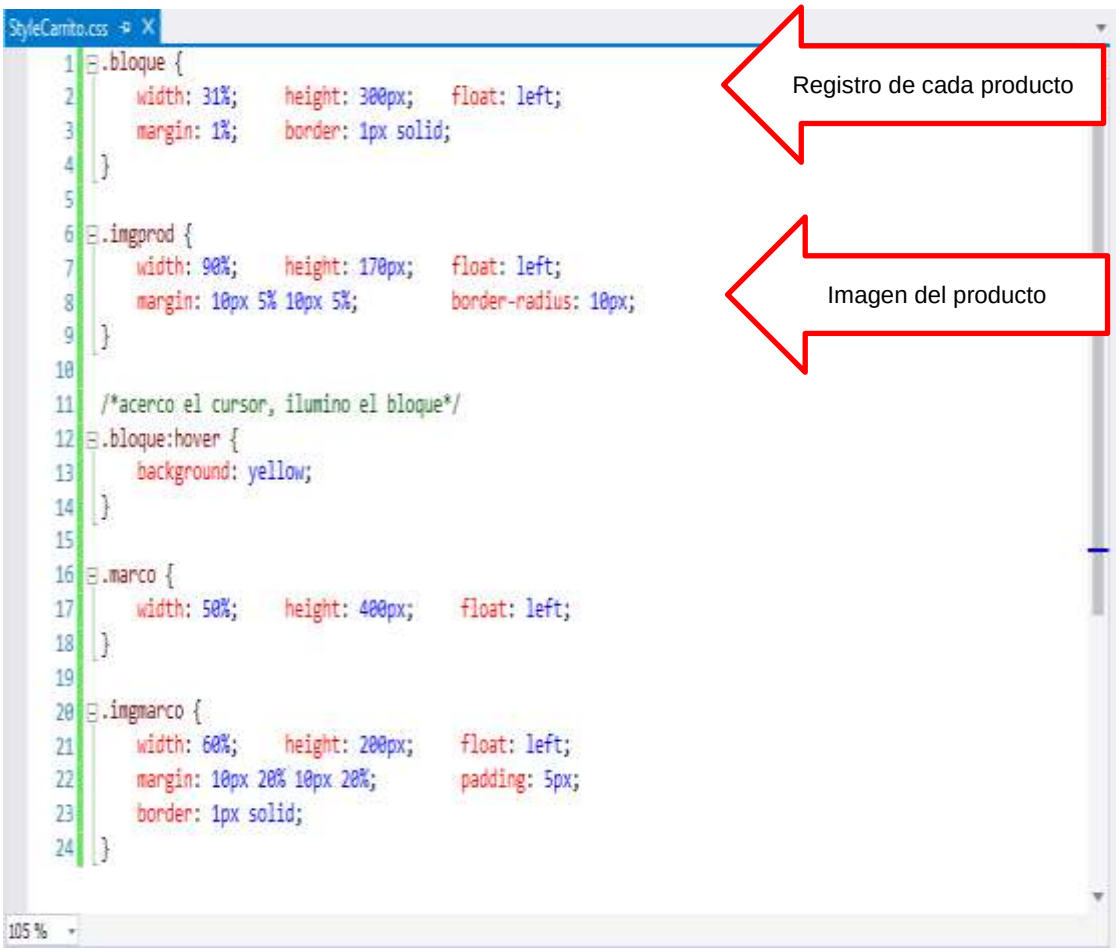
En la carpeta Content agregamos un hoja de estilo, para dar estilo a las etiquetas y selectores de la paginas html.



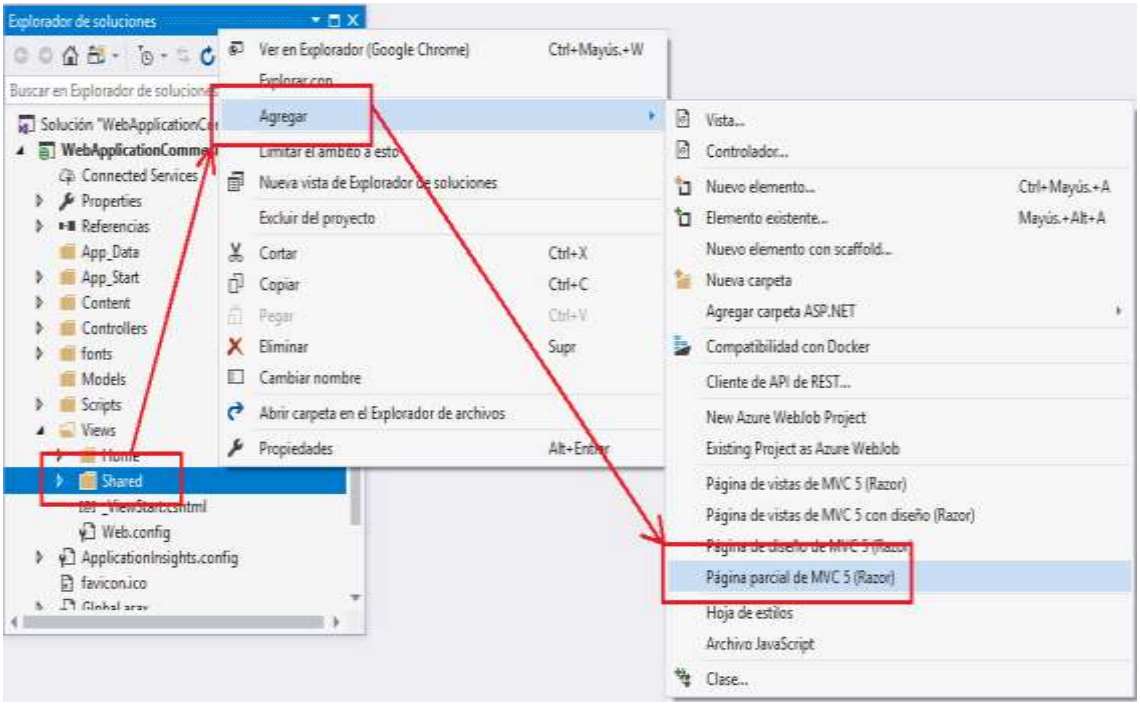
Selecciona la plantilla hoja de estilo y le asignarás el nombre styleCarrito.css, tal como se muestra



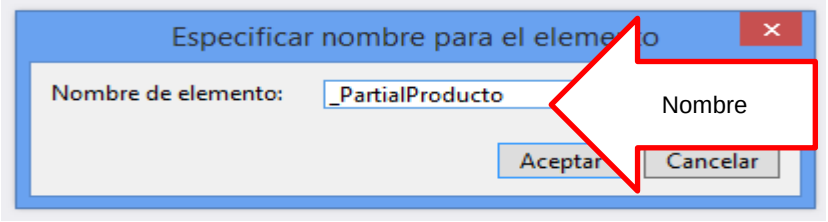
Defina las clases en la hoja de estilo tal como se muestra.



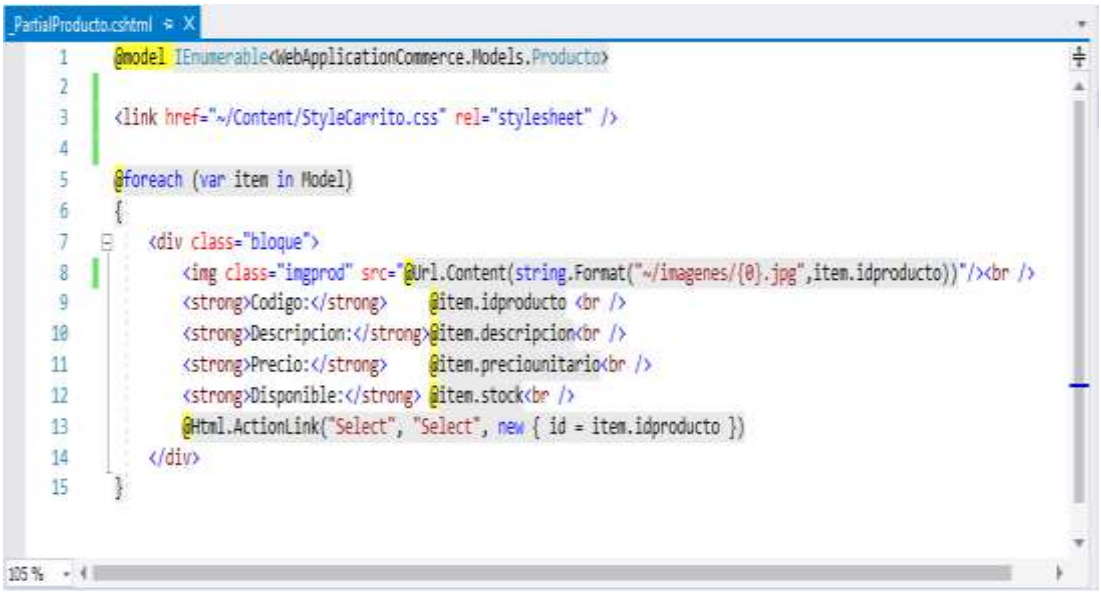
En la carpeta Shared, agrega una página Parcial de MVC5, llamada _PartialProducto, tal como se muestra



Asigne su nombre, y presiona el botón ACEPTAR



Defina el diseño de la página para listar los productos, tal como se muestra.



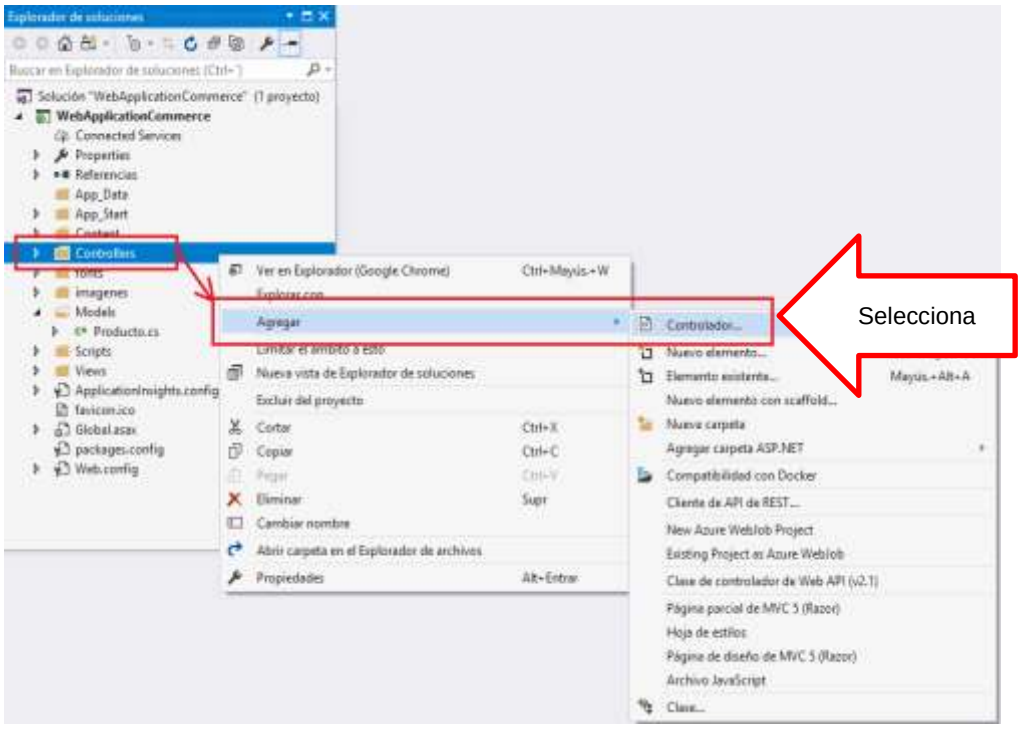
```
_PartialProducto.cshtml
@model IEnumerable<WebApplicationCommerce.Models.Producto>

<link href="~/Content/StyleCarrito.css" rel="stylesheet" />

@foreach (var item in Model)
{
    <div class="bloque">
        <img class="imgprod" src="@Url.Content(string.Format("~/imagenes/{0}.jpg",item.idproducto))"/><br />
        <strong>Codigo:</strong>     @item.idproducto <br />
        <strong>Descripcion:</strong>@item.descripcion<br />
        <strong>Precio:</strong>     @item.preciounitario<br />
        <strong>Disponible:</strong> @item.stock<br />
        @Html.ActionLink("Select", "Select", new { id = item.idproducto })
    </div>
}
```

En la clase partial _PartialRegistro definimos el diseño para visualizar los datos del producto seleccionado, tal como se muestra

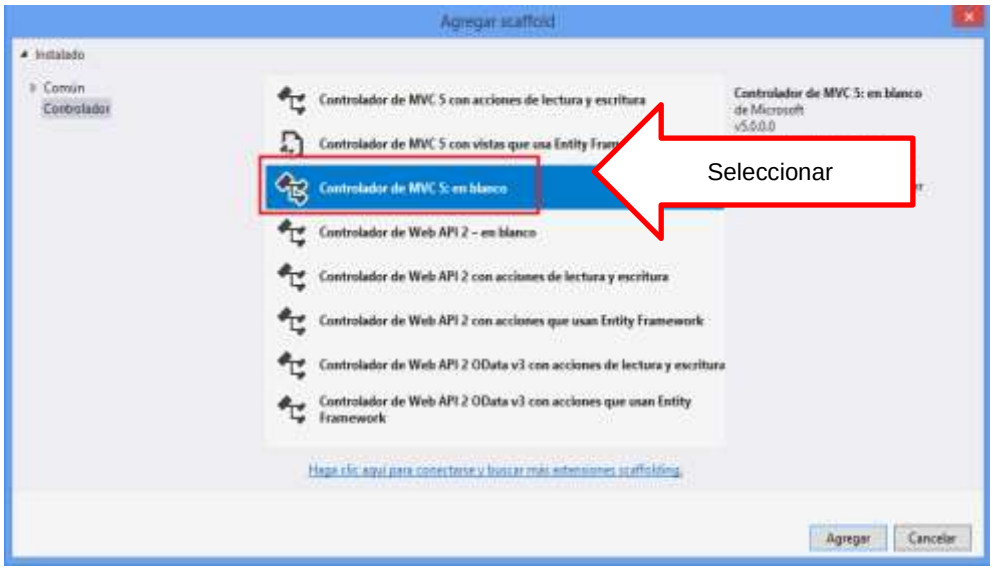
```
_PartialProducto.cshtml  _PartialRegistro.cshtml
@model WebApplicationCommerce.Models.Producto

<link href="~/Content/StyleCarrito.css" rel="stylesheet" />

<div class="marco">
    <dl class="dl-horizontal">
        <dt>@Html.DisplayNameFor(model => model.idproducto)</dt>
        <dd>@Html.DisplayFor(model => model.idproducto)</dd>
        <dt>@Html.DisplayNameFor(model => model.descripcion)</dt>
        <dd>@Html.DisplayFor(model => model.descripcion)</dd>
        <dt>@Html.DisplayNameFor(model => model.preciounitario)</dt>
        <dd>@Html.DisplayFor(model => model.preciounitario)</dd>
        <dt>@Html.DisplayNameFor(model => model.stock)</dt>
        <dd>@Html.DisplayFor(model => model.stock)</dd>
    </dl>
</div>

<div class="marco">
    <img class="imgmarco"
        src="@Url.Content(string.Format("~/imagenes/{0}.jpg",Model.idproducto))" />
</div>
<p>
    @Html.ActionLink("Agregar", "Agregar", new { id = Model.idproducto }) |
    @Html.ActionLink("Retornar", "Tienda")
</p>
```

4. Trabajando con el Controlador

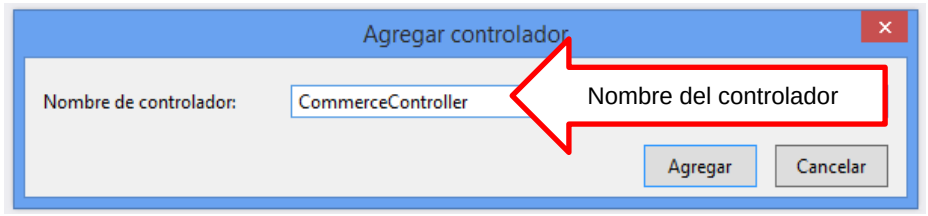
A continuación agrega el controlador llamado CarritoController, tal como se muestra



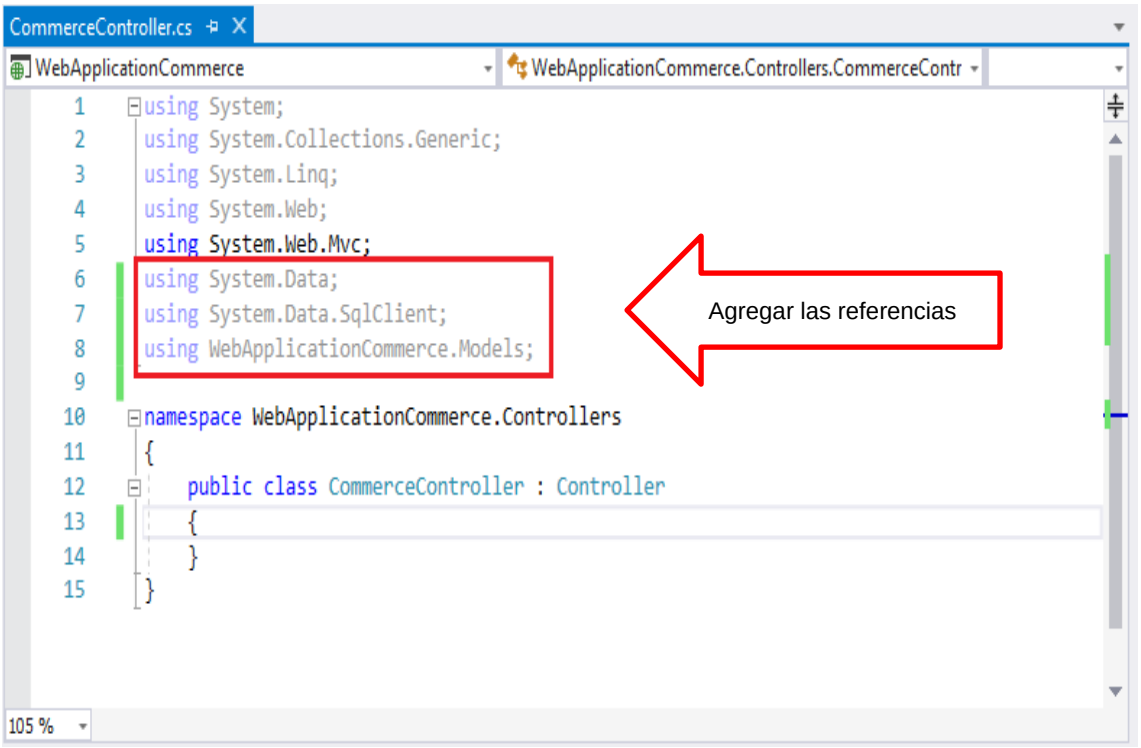
Selecciona el scaffold del controlador



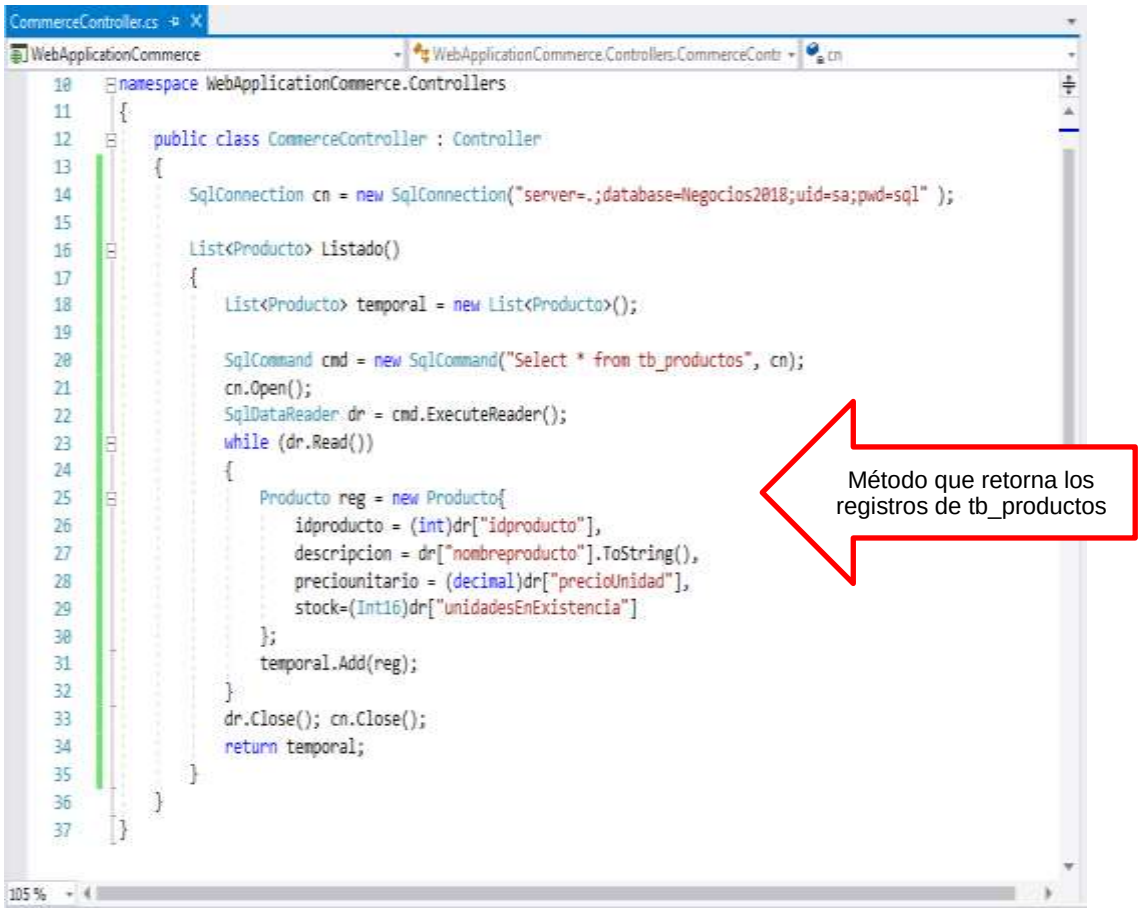
Defina el nombre del controlador y presiona el botón AGREGAR



En el controlador, agregar las referencias: System.Data.SqlClient, carpeta Models

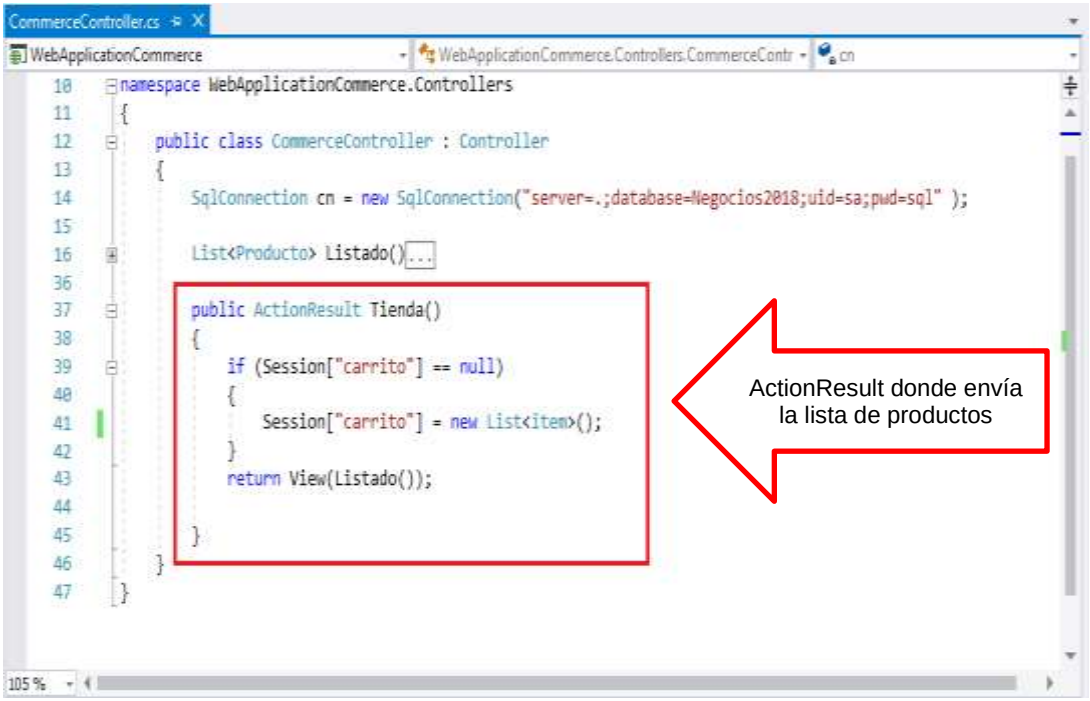


En el controlador, defina la conexión a la base de datos llamada “cn”. A continuación, defina una lista de productos, la cual recupera los registros de tb_productos.

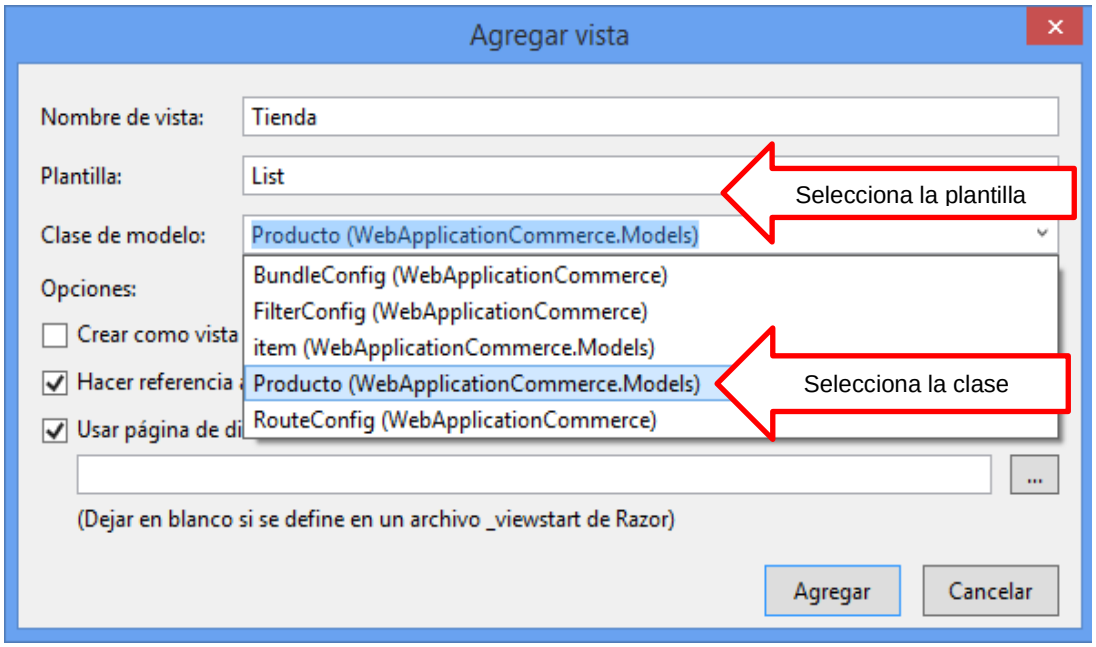


ActionResult Tienda

Defina el ActionResult Tienda(), la cual evalúa si existe el Session["carrito"], luego envío a la Vista la lista de Productos, tal como se muestra.



A continuación, agregar la vista del ActionResult: la plantilla será de tipo List, y la clase de Modelo es "Producto", presiona el botón AGREGAR

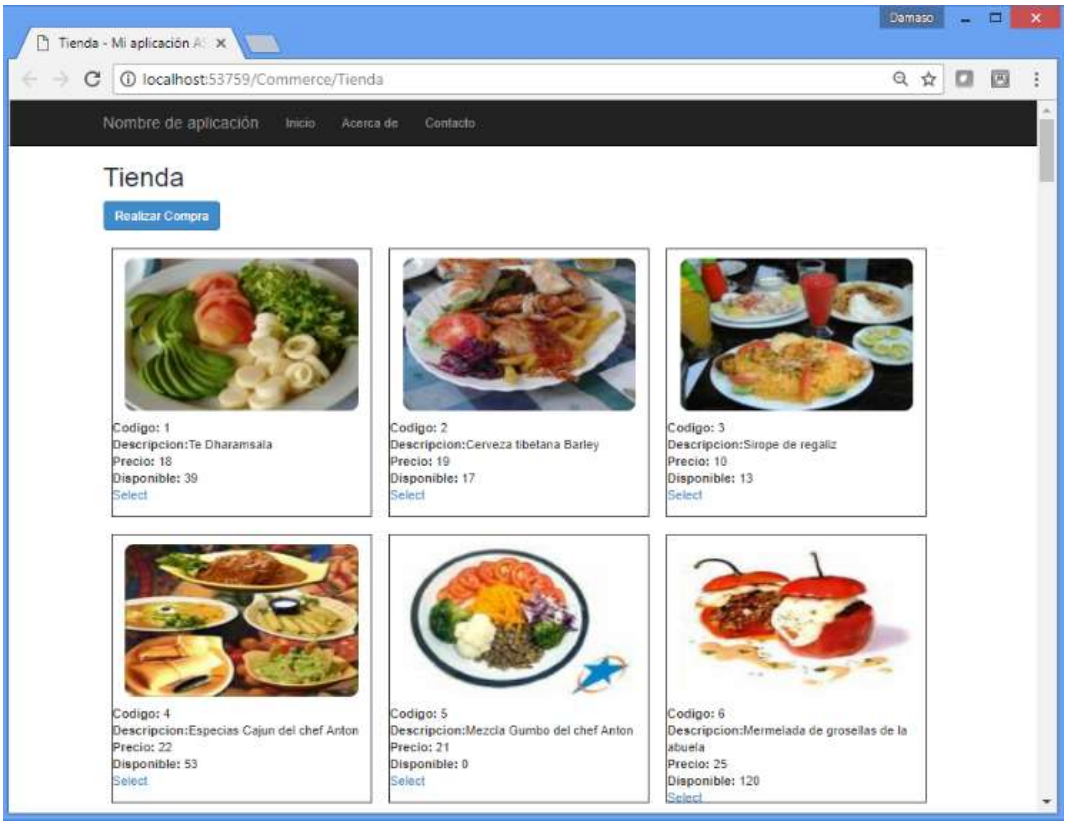


Modificar la estructura de la Vista, agregar la vista Parcial “_PartialProducto”, tal como se muestra

Tienda.cshtml CommerceController.cs

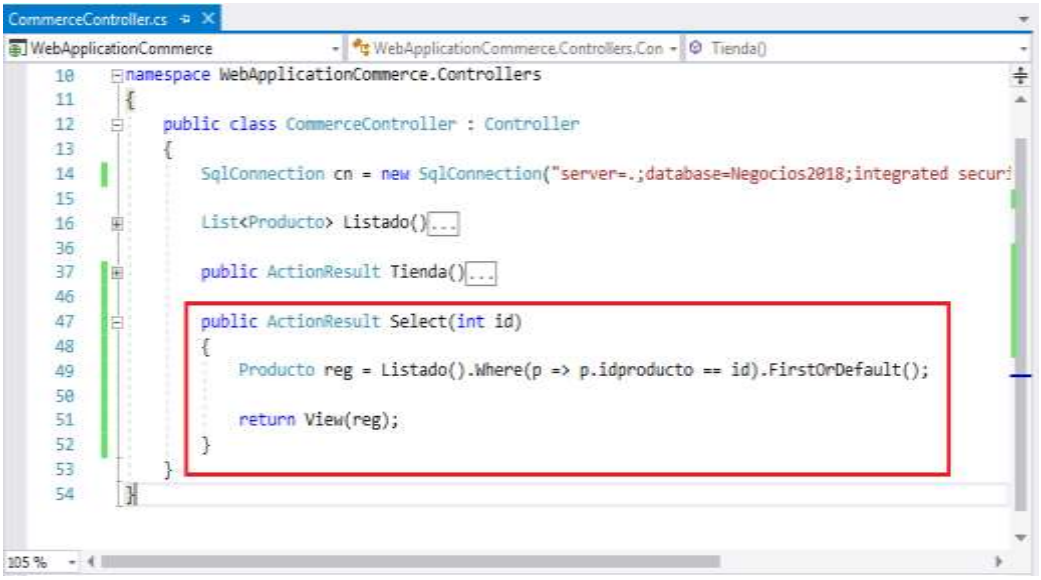
```
@model IEnumerable<WebApplicationCommerce.Models.Producto>

@{
    ViewBag.Title = "Tienda";
}

<h2>Tienda</h2>

<p>
    @Html.ActionLink("Realizar Compra", "Comprar", null, new {@class="btn btn-primary" })
</p>

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

Ejecuta la Vista, donde se visualiza los productos almacenados en la Base de datos.

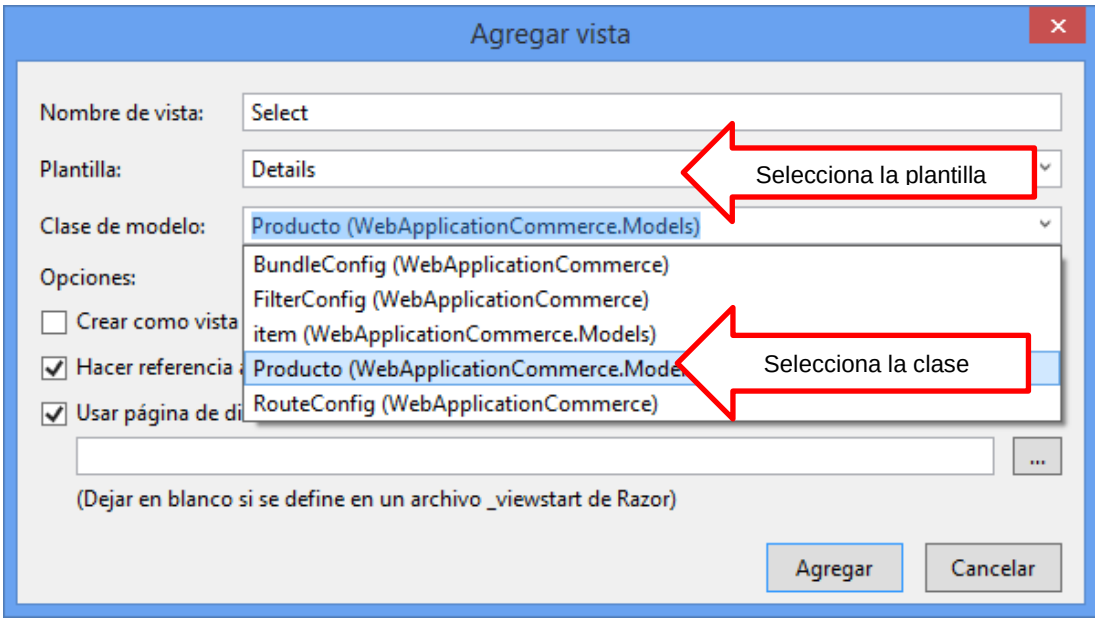


ActionResult Select

En el controlador Carrito, defina el método **Select**, donde recibe el id, busca el registro en la tabla tb_productos, enviando el registro seleccionado.



Agregar la Vista del ActionResult, donde la clase de modelo es Producto.

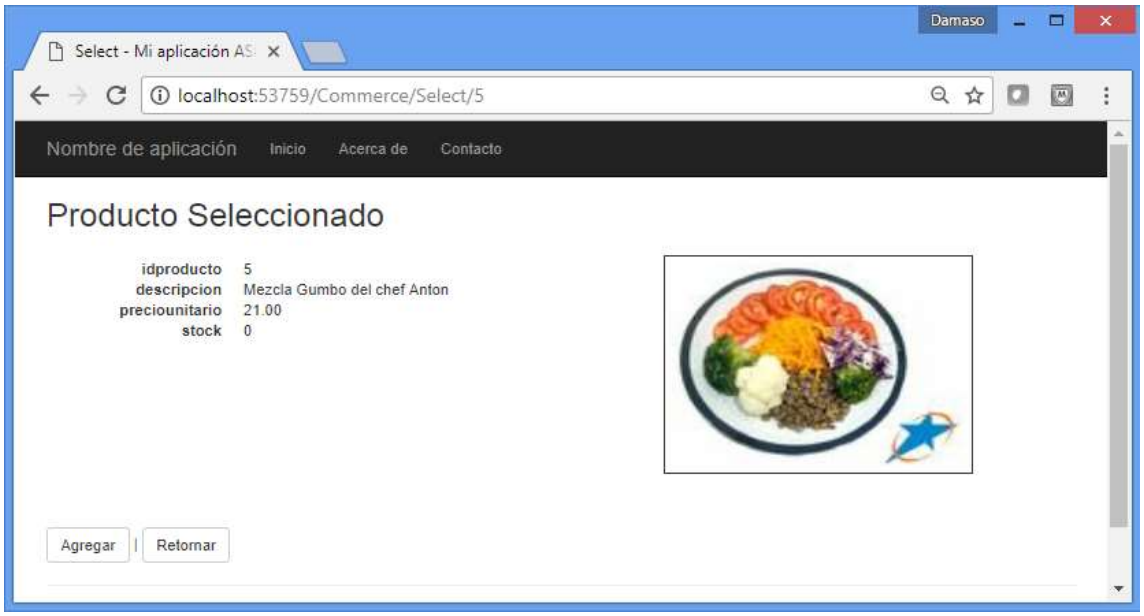


Diseña la vista Select, agregar la vista parcial "_PartialRegistro", tal como se muestra.

Select.cshtml | CommerceController.cs

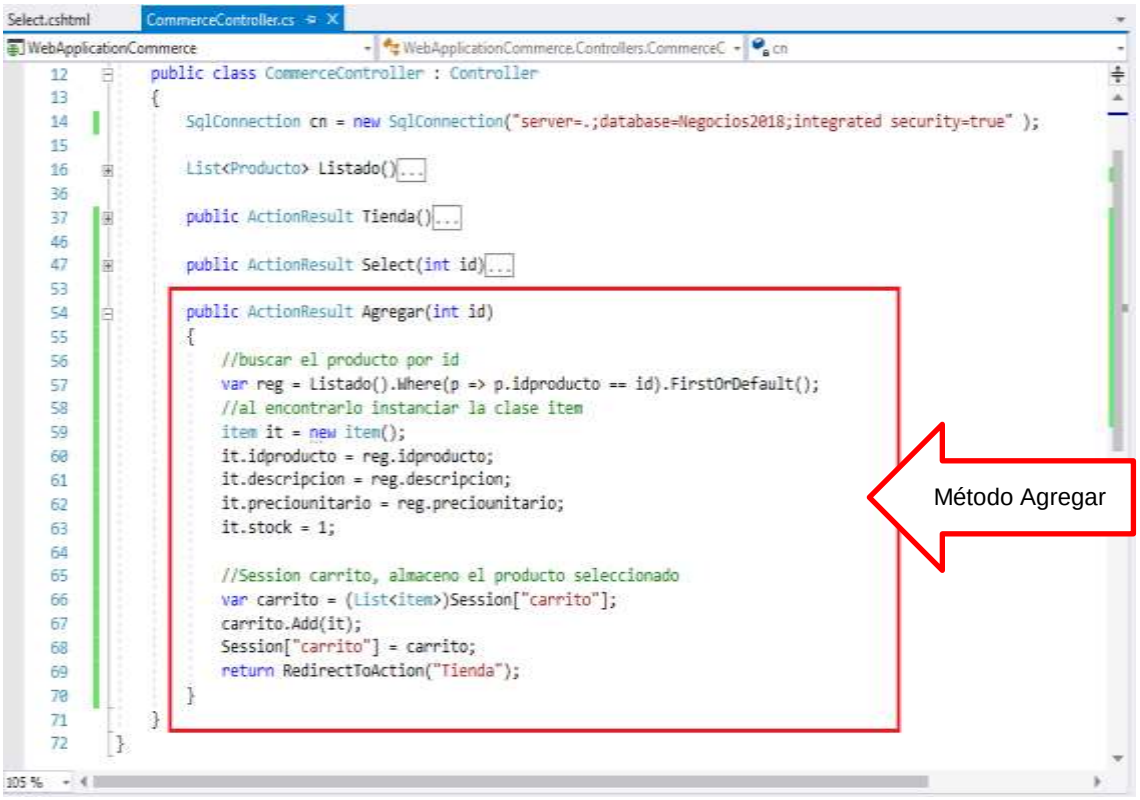
```
@model WebApplicationCommerce.Models.Producto

@{
    ViewBag.Title = "Select";
}

<h2>Producto Seleccionado</h2>

@Html.Partial("_PartialRegistro")

<div>
    @Html.ActionLink("Agregar", "Agregar", new { id = Model.idproducto }, new { @class="btn btn-default"}) |
    @Html.ActionLink("Retornar", "Tienda",null, new { @class = "btn btn-default" })
</div>
```

A continuación ejecuta el proyecto, al seleccionar el producto, mostraremos sus datos, tal como se muestra



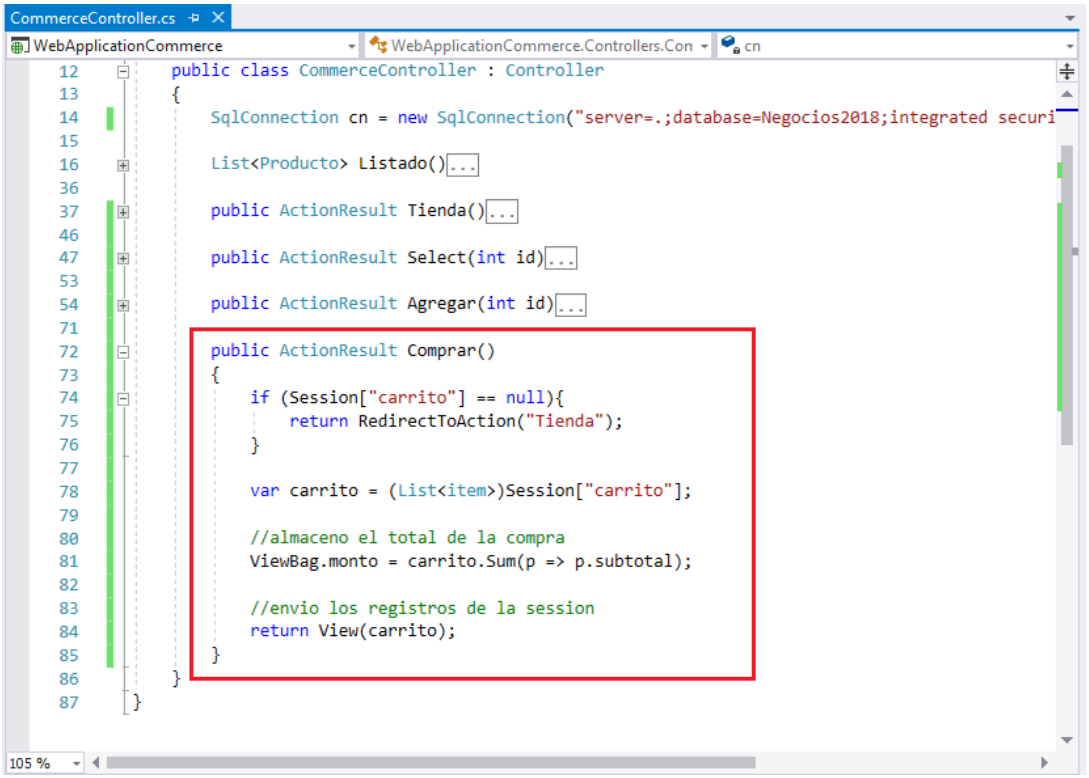
ActionResult Agregar

En este proceso, agrego el registro seleccionado desde la página y es añadido a la Session[“carrito”], tal como se muestra.

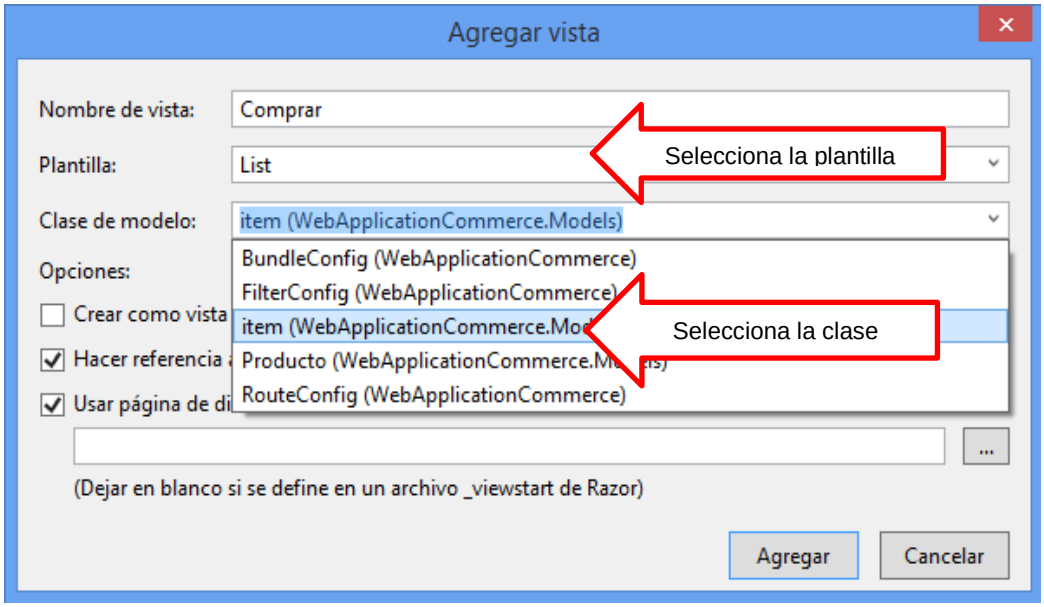


ActionResult Comprar

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.monto, tal como se muestra



A continuación, agrega la Vista, donde la plantilla es de tipo **List** y la clase de modelo es **item**, tal como se muestra

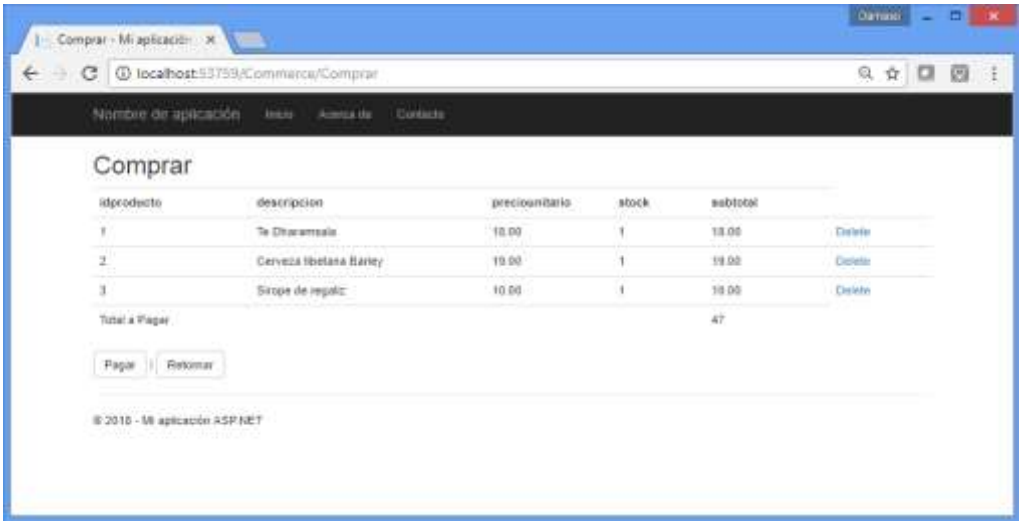


Diseña la Vista Comprar, agregando el ViewBag.monto. Agregar dos ActionLink, para ejecutar los procesos de Pago, Retornar a la Tienda, tal como se muestra

Comprar.cshtml | CommerceController.cs

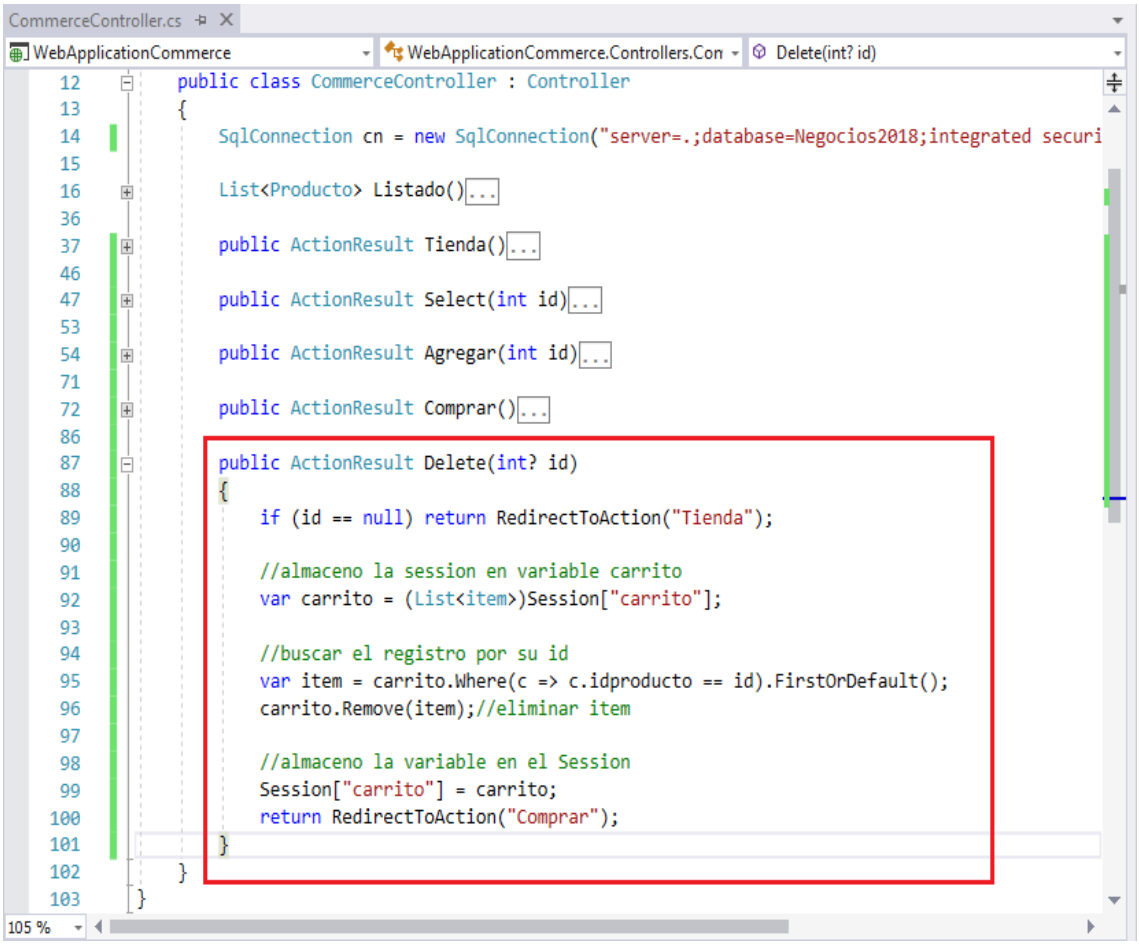
```
@model IEnumerable<WebApplicationCommerce.Models.item>

@{ ViewBag.Title = "Comprar"; }

<h2>Comprar</h2>
<table class="table">
    <tr>
        <th>@Html.DisplayNameFor(model => model.idproducto)</th>
        <th>@Html.DisplayNameFor(model => model.descripcion)</th>
        <th>@Html.DisplayNameFor(model => model.preciounitario)</th>
        <th>@Html.DisplayNameFor(model => model.stock)</th>
        <th>@Html.DisplayNameFor(model => model.subtotal)</th>
    </tr>
@foreach (var item in Model) {
    <tr>
        <td>@Html.DisplayFor(modelItem => item.idproducto)</td>
        <td>@Html.DisplayFor(modelItem => item.descripcion)</td>
        <td>@Html.DisplayFor(modelItem => item.preciounitario)</td>
        <td>@Html.DisplayFor(modelItem => item.stock)</td>
        <td>@Html.DisplayFor(modelItem => item.subtotal)</td>
        <td>@Html.ActionLink("Delete", "Delete", new { id=item.idproducto })</td>
    </tr>
}
    <tr>
        <td colspan="4">Total a Pagar</td>
        <td>@ViewBag.monto</td>
    </tr>
</table>
<div>
    @Html.ActionLink("Pagar", "Pagar", null, new { @class = "btn btn-default" }) |
    @Html.ActionLink("Retornar", "Tienda", null, new { @class = "btn btn-default" })
</div>
```

Al ejecutar el proyecto, agrega algunos productos. Al presionar la opción realizar Compra, se visualiza los registros seleccionados, tal como se muestra



ActionResult Delete

En este proceso procedemos a eliminar un registro del carrito, y re direccionamos a la página Comprar, tal como se muestra



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.

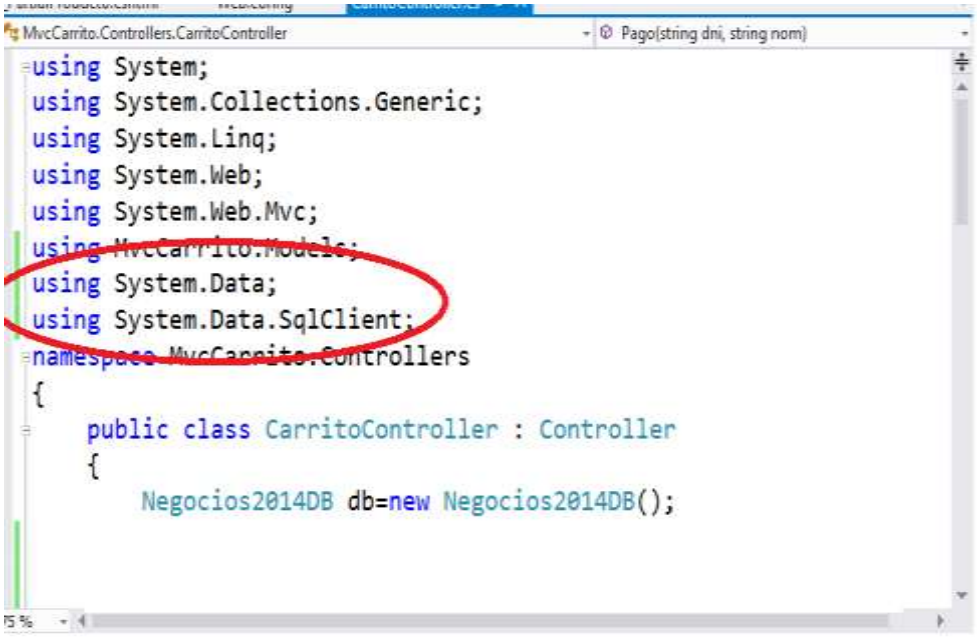
```
public class CarritoController : Controller
{
    Negocios2014DB db=new Negocios2014DB();

    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

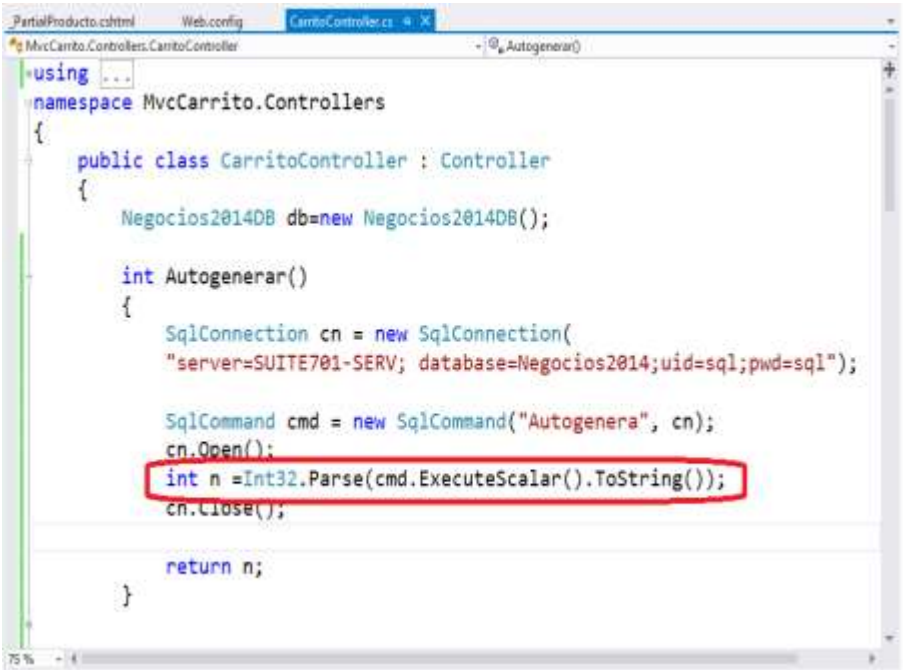
```
@model IEnumerable<MvcCarrito.Models.registro>
@{
    ViewBag.Title = "Pago";
}

<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

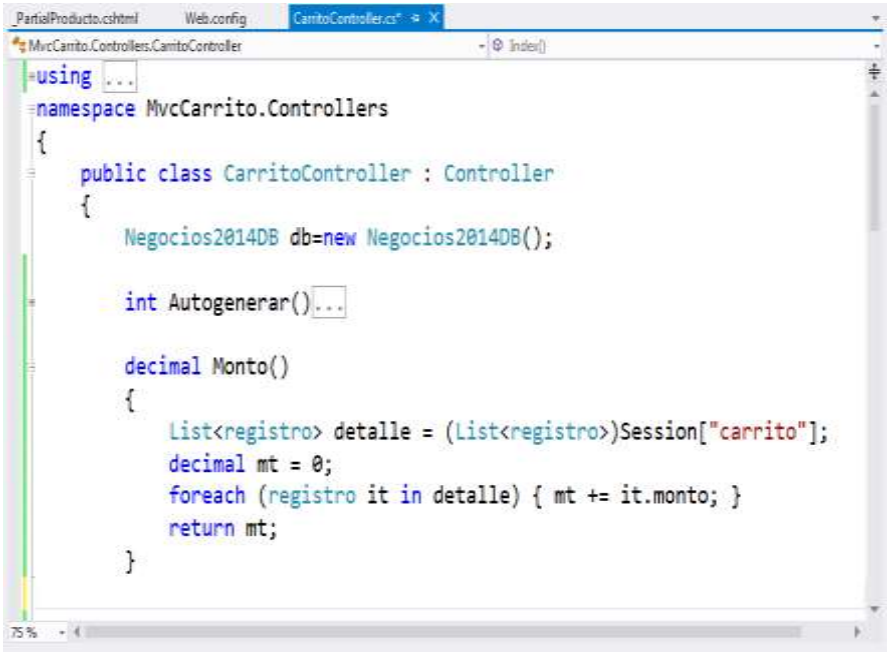
Defina las librerías de trabajo dentro del controlador



Función autogenera(), retorna el siguiente valor del número del pedido. Utilice la función ExecuteScalar en el proceso

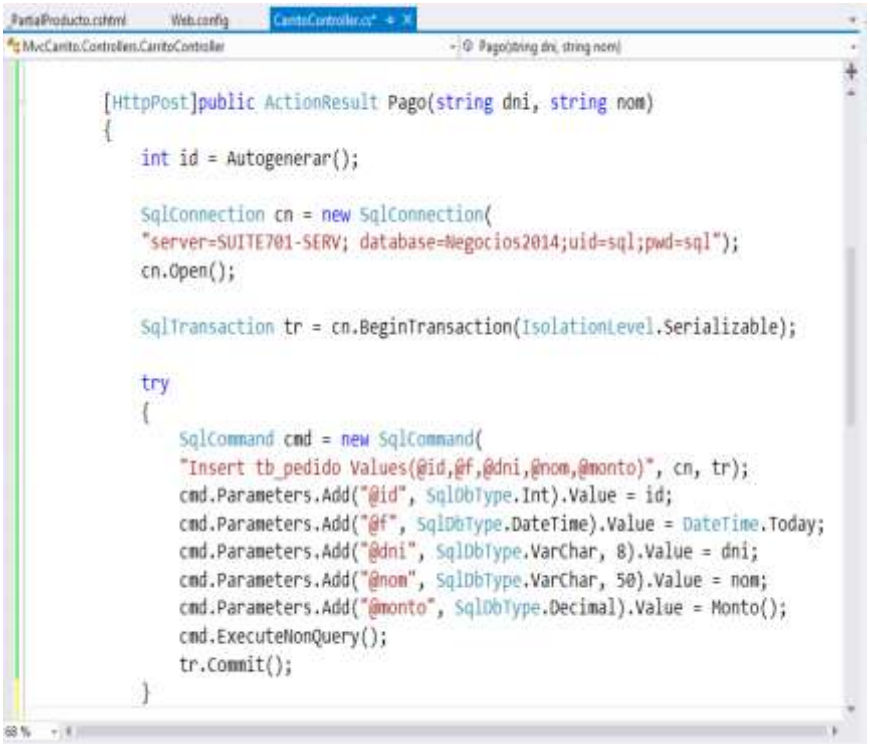


Función Monto(), retorna el total acumulados del campo monto de detalle.



```
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.



```
[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();
    }
```

```
            /*detalle del pedido*/
            List<registro> detalle = (List<registro>)Session["carrito"];
            foreach (registro it in detalle)
            {
                cmd = new SqlCommand(
                    "Insert tb_detapedido Values(@id,@prod,@pre,@q)", cn, tr);
                cmd.Parameters.Add("@id", SqlDbType.Int).Value = id;
                cmd.Parameters.Add("@prod", SqlDbType.Int).Value = it.idproducto;
                cmd.Parameters.Add("@pre", SqlDbType.Decimal).Value = it.precio;
                cmd.Parameters.Add("@q", SqlDbType.Int).Value = it.cantidad;
                cmd.ExecuteNonQuery();
            }
            tr.Commit();
        }
        catch (Exception ex){
            tr.Rollback();
        }
        finally{
            cn.Close();
        }
        return View("Vista");
    }
```

Archivos del proceso

```
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
```

Consulta ejecutada correctamente. 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://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

UNIDAD

4

MANEJO DE REPORTES (REPORTVIEWER)

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno exporta los datos de un origen de datos utilizando ReportViewer, desde una aplicación de ASP.NET MVC.

Temario

Tema 8: Manejo de reportes (4 horas)

- Diseño el reporte asignando origen de datos.
- Trabajando con ReportViewer.

ACTIVIDADES PROPUESTAS

- Los alumnos implementan un servicio para realizar consulta de datos desde un origen de datos remotos.
- Los alumnos crear aplicaciones para generar reportes por ReportViewer.
- Los alumnos desarrollan los laboratorios de la semana.

8. Manejo de Reportes

8.1 Introducción

En ciertas ocasiones en nuestras aplicaciones es necesario generar reportes para mostrar resultados de nuestra lógica de negocio. Pero bien que herramienta usar. La verdad hay bastantes herramientas case para generar reportes, como Cristal Report, Stimulsoft Reports, entre otras. Unas con licencia otras Open Source, pero no tenemos en cuenta lo que nos provee Visual Studio en su suite.

Microsoft Visual Studio incluye funcionalidad de diseño de informes y controles ReportViewer para que pueda agregar informes completos a las aplicaciones personalizadas. Los informes pueden contener datos tabulares, agregados y multidimensionales. Los controles ReportViewer se proporcionan para que pueda procesar y mostrar el informe en su aplicación.

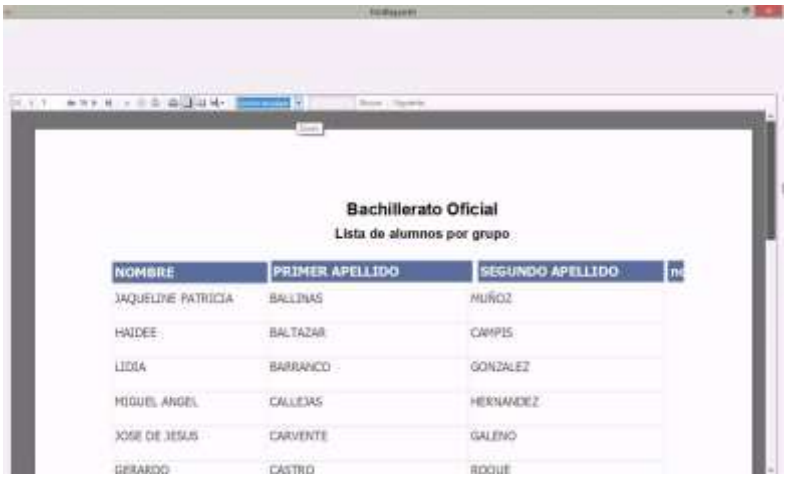


Figura 1: reportes con ReportViewer

8.2 Diseño de Reportes asignando origen de datos

Un archivo de definición de informe de cliente (.rdlc) incluye elementos de conjuntos de datos que definen la estructura de los orígenes de datos que utiliza el informe. Debe agregar uno o varios conjuntos de datos al informe antes de poder usar los datos en la definición de informe. Después de crear un conjunto de datos, puede arrastrar un campo específico hasta una región de datos o un cuadro de texto en el informe.

Los archivos de definición de informe del cliente (.rdlc) los procesa el control ReportViewer como informes locales. A diferencia de los informes de servidor, los informes locales requieren que se procesen los datos antes de que el control ReportViewer pueda procesar los informes. Los informes locales pueden utilizar datos procedentes de cualquier origen, siempre que puedan ser suministrados como un objeto DataTable o como una colección IEnumerable de objetos comerciales. La tabla de datos o el objeto comercial devuelve una lista de campos que pueden utilizarse para el informe. Cada campo contiene un puntero a un campo de base de datos y una propiedad de nombre, un campo de un origen de datos del objeto o una columna de una DataTable. Es posible arrastrar campos desde la ventana Datos de informe hasta la superficie de diseño del informe.

Se recomiendan los enfoques siguientes para configurar un DataSet o una enumeración IEnumerable de objetos comerciales como un origen de datos de informe. Después

configurar un origen de datos, puede enlazar el DataSet o los objetos comerciales al informe.

Usar tablas de datos

Para crear una DataTable, utilice el comando Agregar nuevo elemento del menú Proyecto y seleccione el objeto DataSet. Arrastre un TableAdapter desde el cuadro de herramientas al Editor de DataSet para configurar la DataTable con el Asistente para la configuración de TableAdapter. El Asistente para la configuración de TableAdapter proporciona un generador de consultas y una característica de vista previa de datos para que pueda confirmar los resultados de la consulta inmediatamente.

8.3 Trabajando con el ReportViewer

El componente ReportViewer se va a encargar de contener nuestro informe, el cual podremos cargar tanto dinámica, como estáticamente.
Para insertar un ReportViewer, basta con ir al cuadro de herramientas y arrastrarlo en nuestro formulario.

PROPIEDADES DE UN REPORTVIEWER

Entre las propiedades del ReportViewer podemos destacar las siguientes:

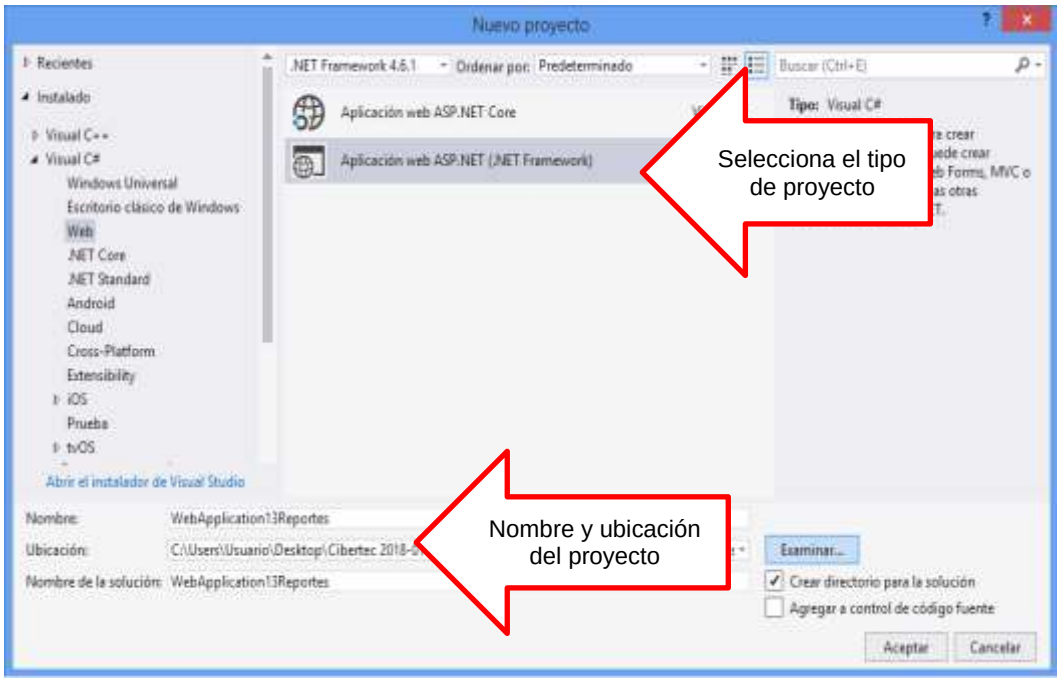
- SizeToReportContent: Determina si el área del informe tiene un tamaño fijo o si equivale al tamaño del contenido del informe.
- Propiedades de la categoría “Barra de herramientas”: Hace que se muestre u oculte cualquier botón del ReportViewer (flechas de navegación, botón imprimir,…).
- AsyncRendering: Determina si el informe se representa asincrónicamente a partir del resto de la página.

Laboratorio 8.1
Reportes con ReportViewer

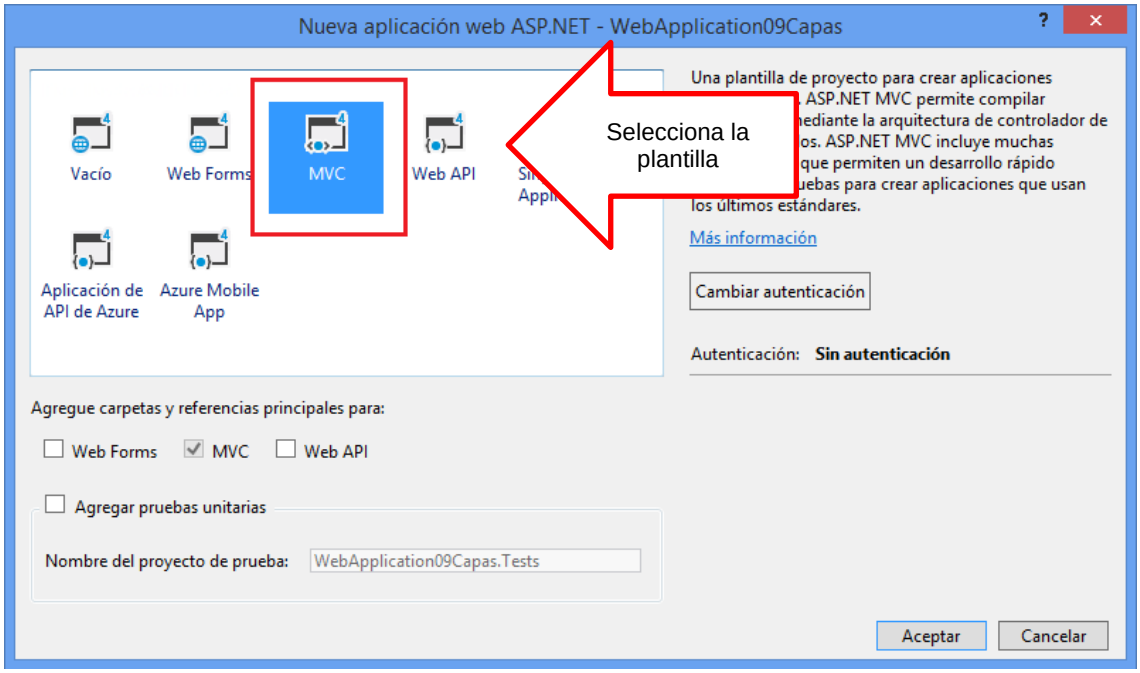
Implemente un proyecto ASP.NET MVC, donde permita generar reportes para listar los datos desde un origen de datos

1. Creando el proyecto

Iniciamos Visual Studio; seleccionar el proyecto Web; selecciona la plantilla Aplicación web ASP.NET (.NET Framework), asignar el nombre y ubicación del proyecto; y presionar el botón ACEPTAR

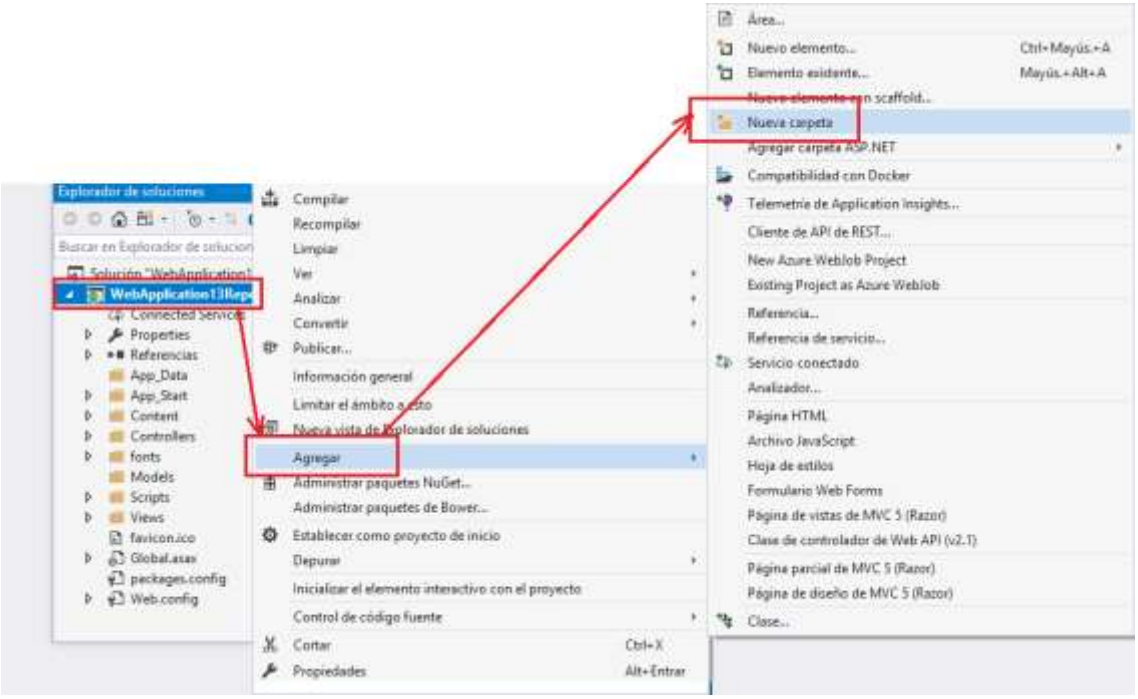


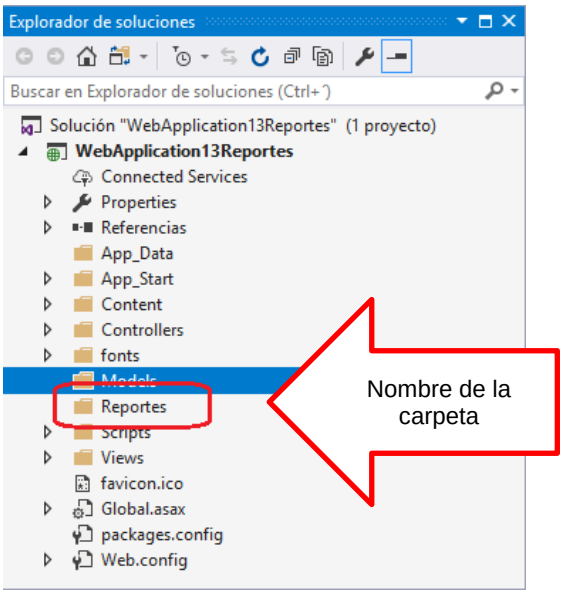
A continuación, seleccionar la plantilla del proyecto MVC.



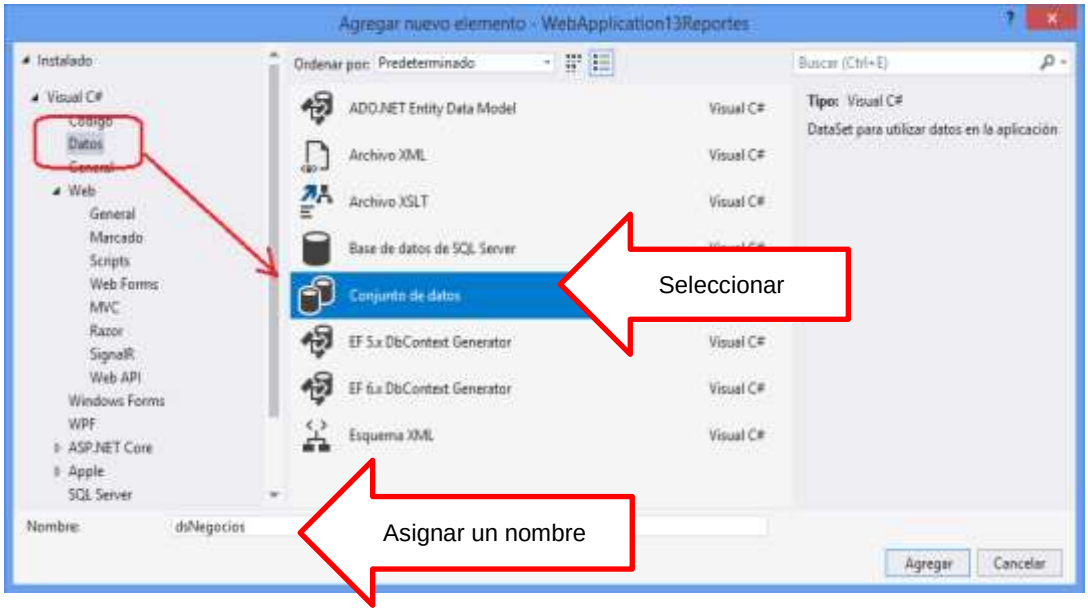
2. Trabajando con un DataSet

En este proceso vamos a definir un DataSet para ello, creamos una carpeta llamada Reportes en el proyecto Web, tal como se muestra.

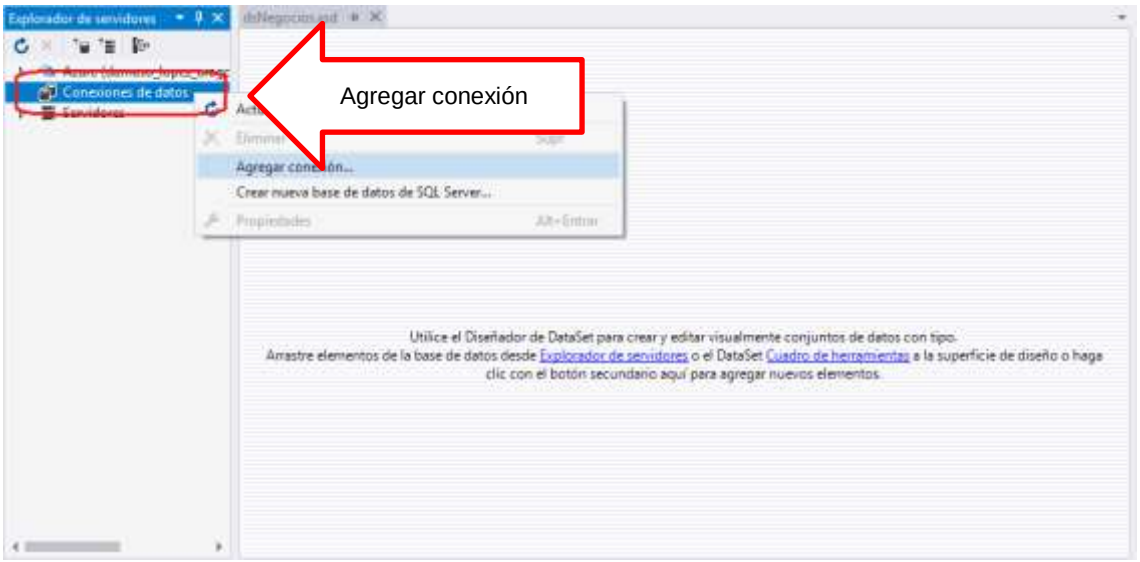




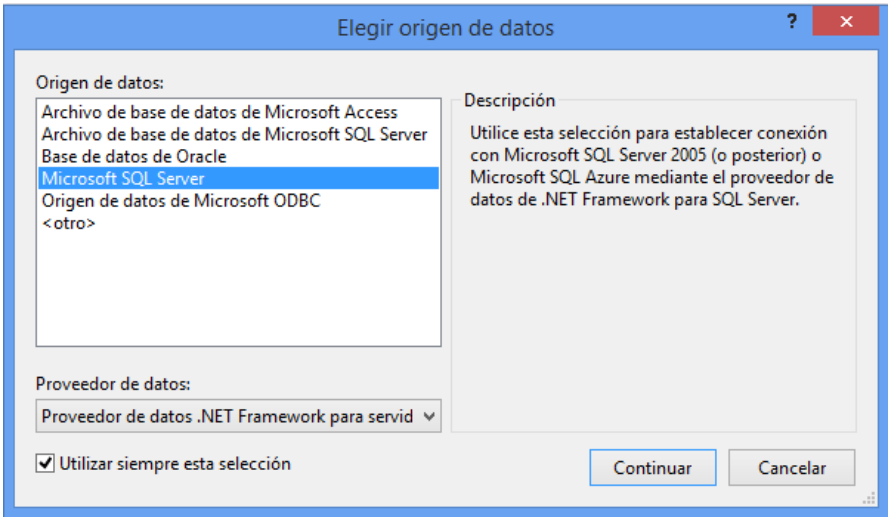
En la carpeta **Reportes**, agrega un DataSet llamado **dsNegocios**, tal como se muestra: selecciona desde la opción Datos → Conjunto de Datos



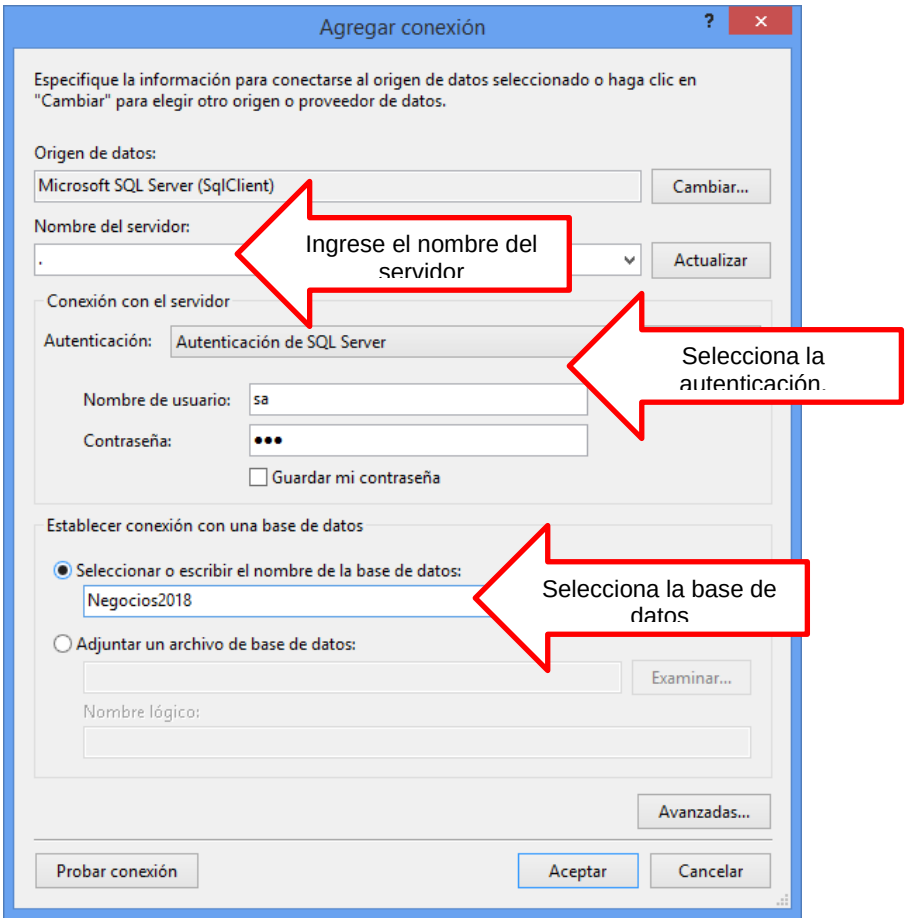
En la ventana **Explorador de Servidores**, agregar una conexión a la base de datos Negocios2018, tal como se muestra.



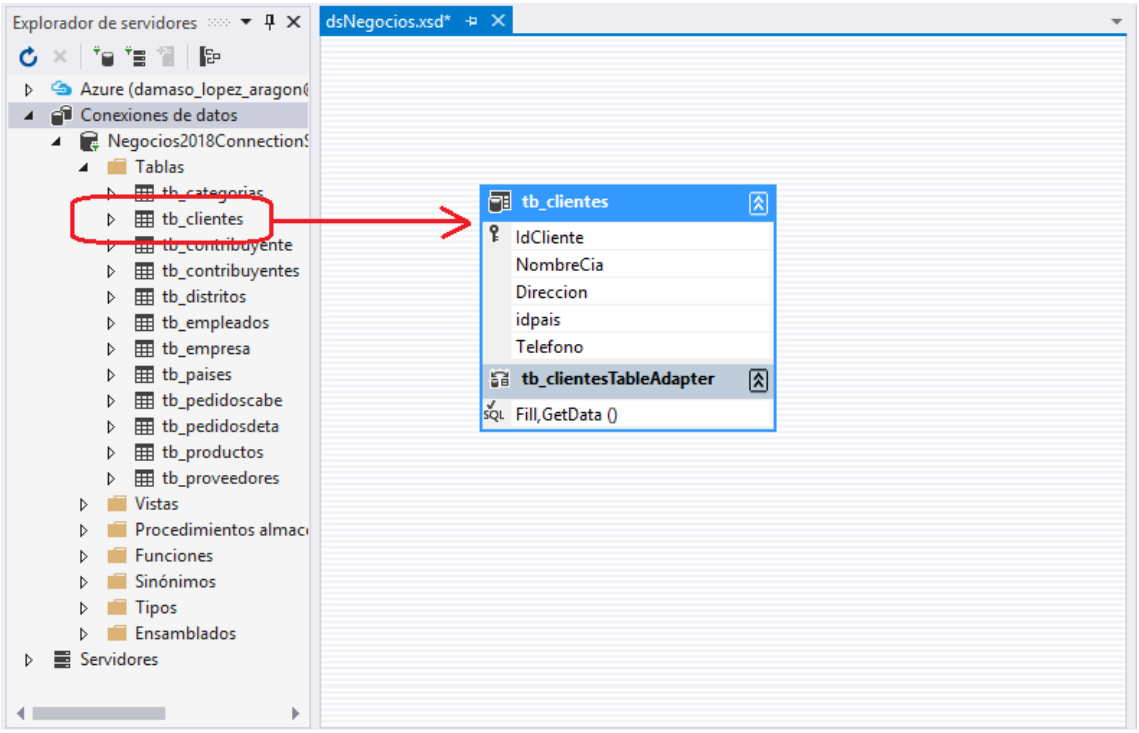
Selecciona el origen de datos: Microsoft SQL Server



Defina la conexión a la base de datos: Ingrese el nombre del servidor, selecciona la autenticación: Windows o SQL Server y selecciona la base de datos, tal como se muestra

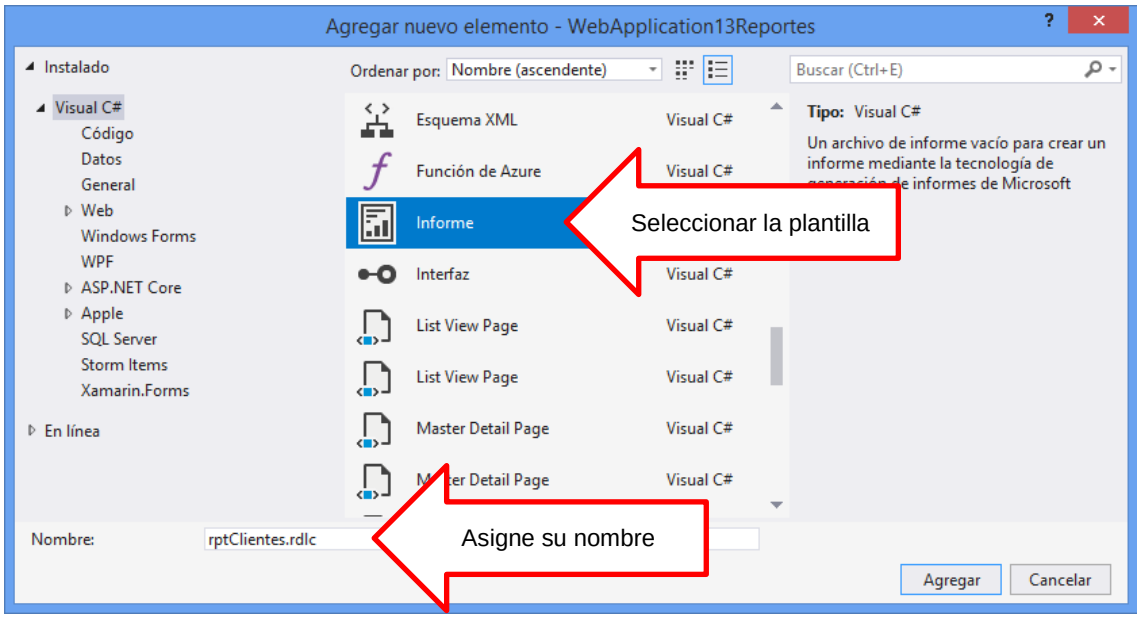


Expanda la conexión, arrastrar la tabla tb_clientes al dataSet, tal como se muestra

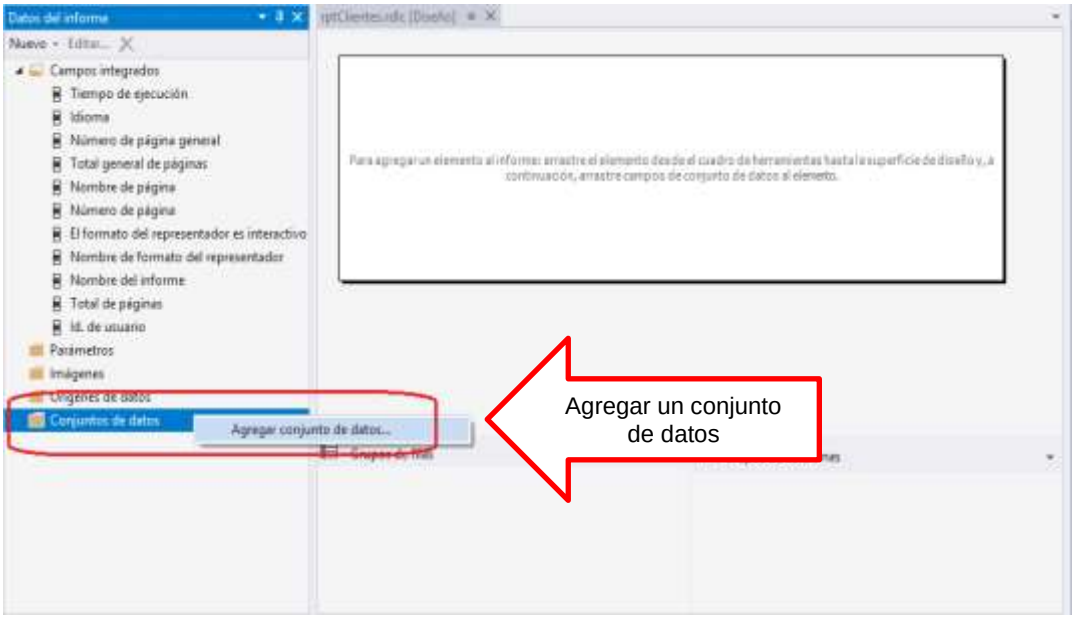


3. Agregando el archivo de Reporte rdlc

En la carpeta Reportes, agregar un archivo rdlc: Selecciona el item Informe, asigne el nombre del archivo, tal como se muestra

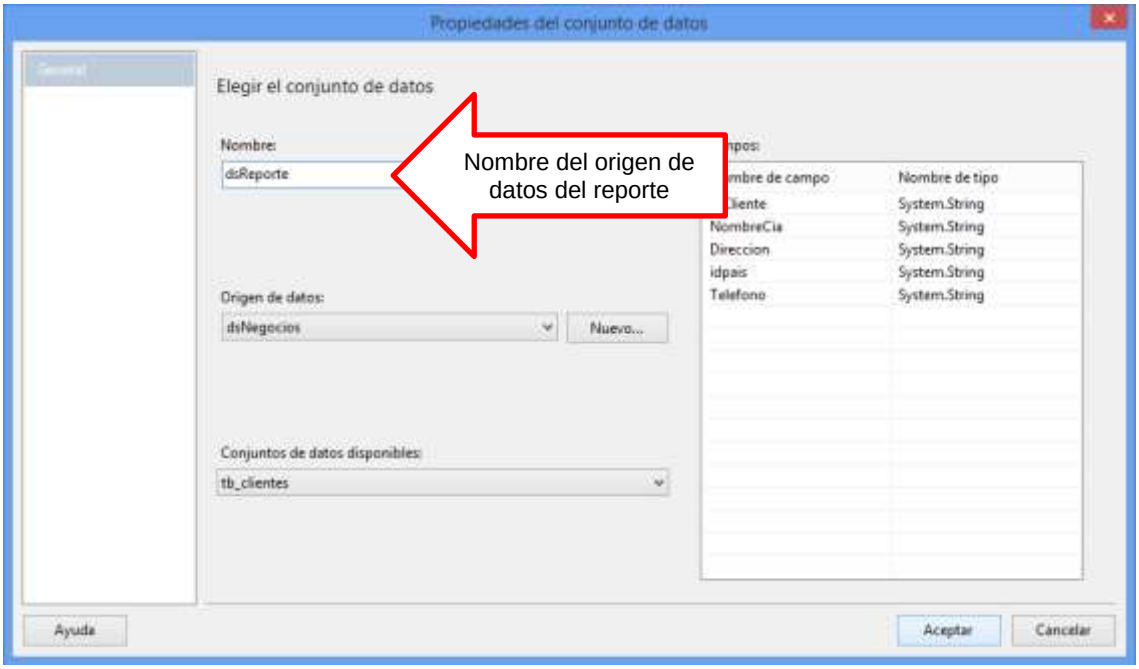


Desde el cuadro de herramientas, agregamos un conjunto de Datos: DataSet. En la carpeta Conjunto de datos, click derecho y selecciona la opción Agregar conjunto de datos

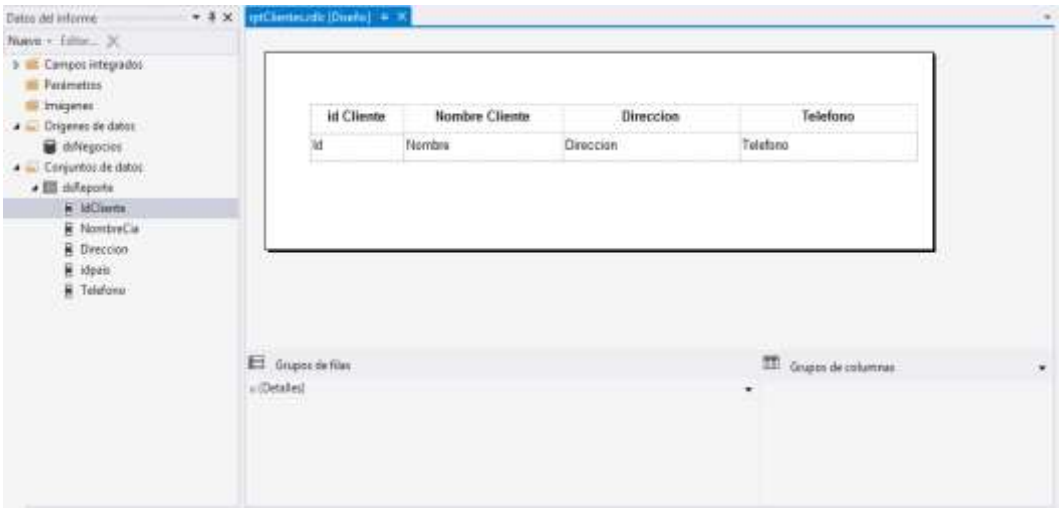


En la ventana de propiedad:

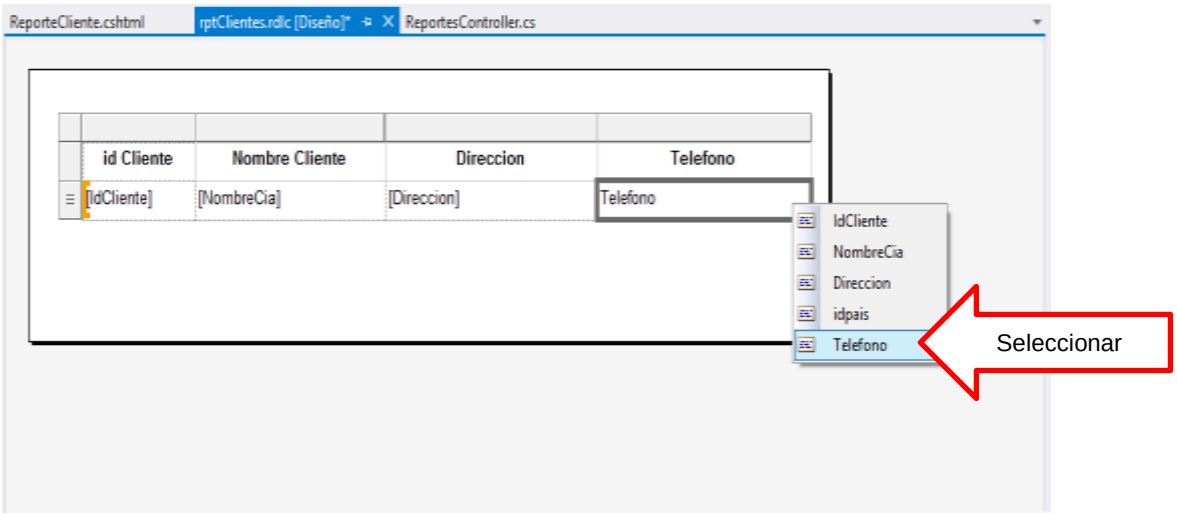
- Asigne el nombre del conjunto de datos: dsReporte
- Selecciona el origen de datos: dsNegocios
- Selecciona el conjunto de datos disponibles: tb_clientes



Diseña el archivo de reportes utilizando los campos del dataset dsReporte, tal como se muestra

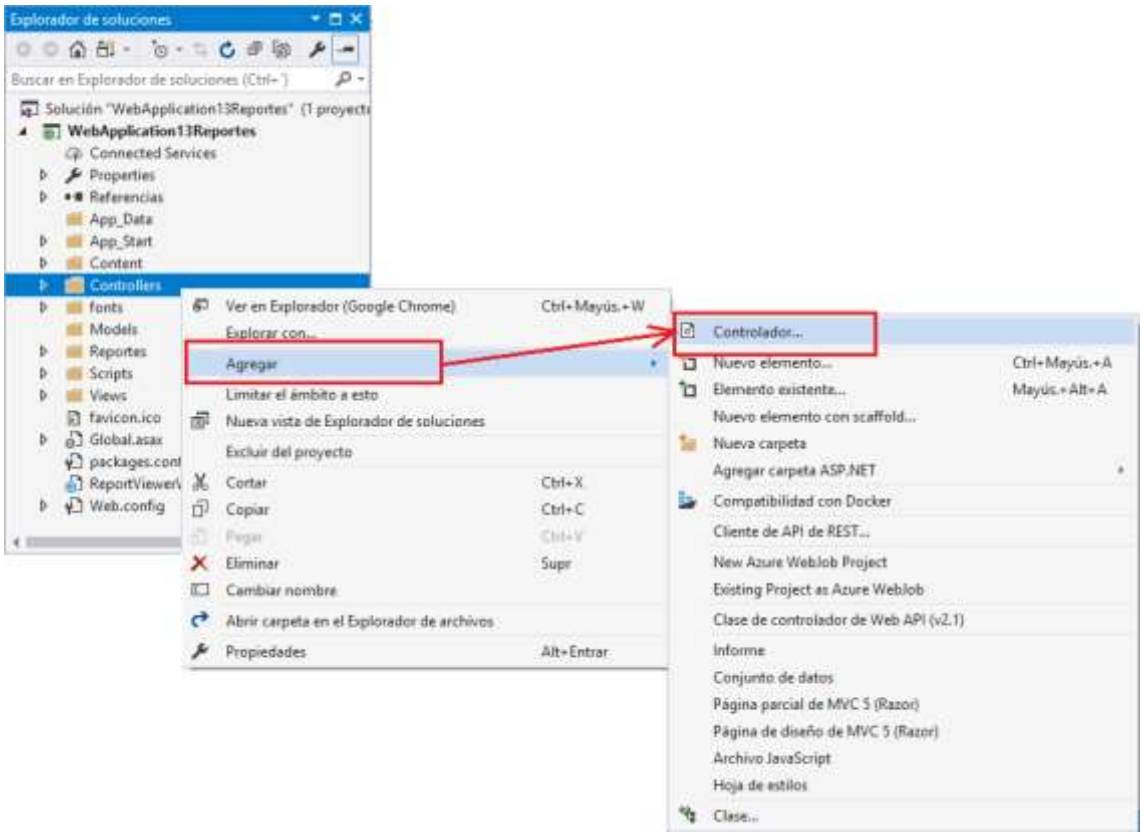


A cada uno de los campos, selecciona el origen de cada uno de ellos, tal como se muestra

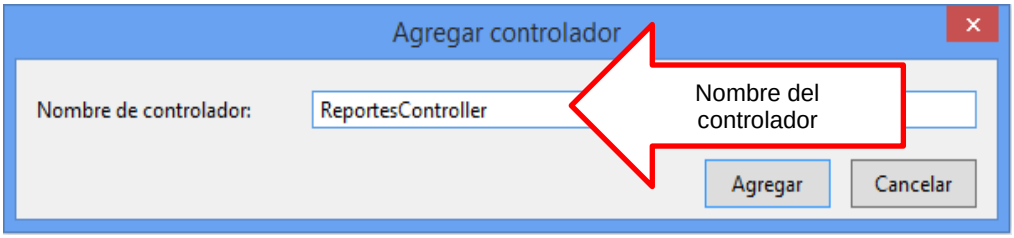


4. Trabajando con el Controlador

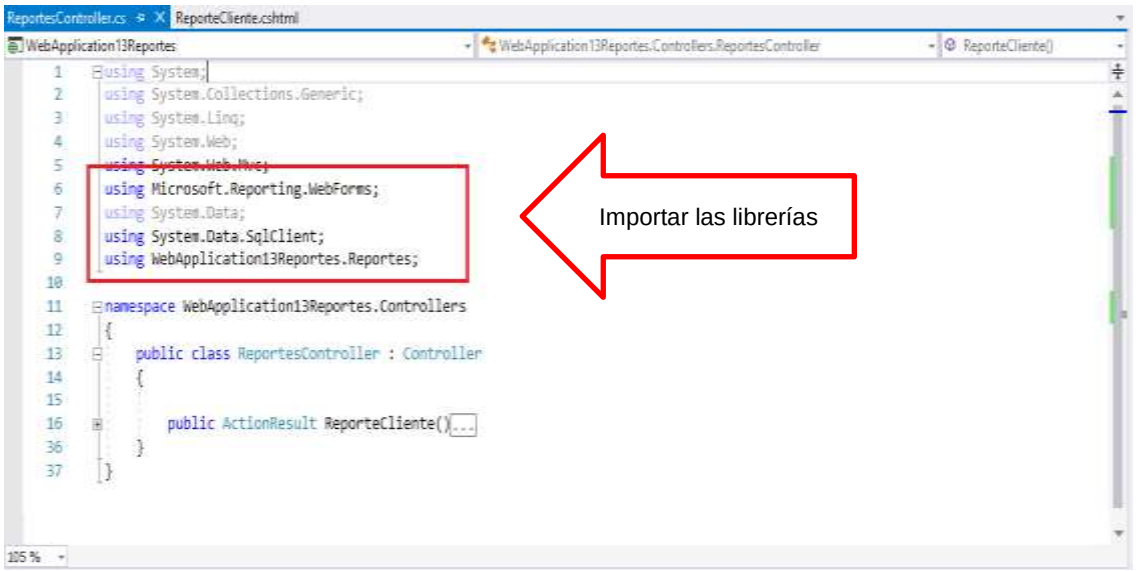
Agregar en la carpeta Controllers un nuevo controlador, tal como se muestra



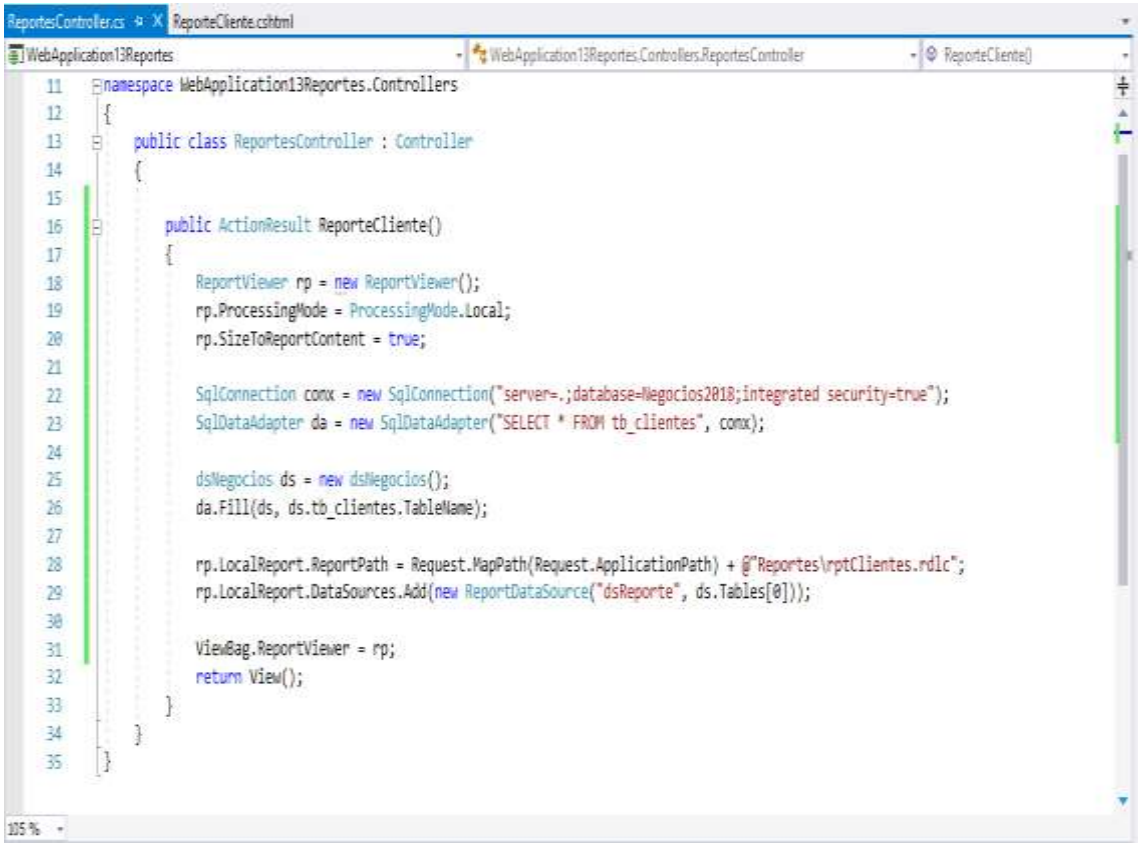
Seleccionar el Scaffold en blanco.



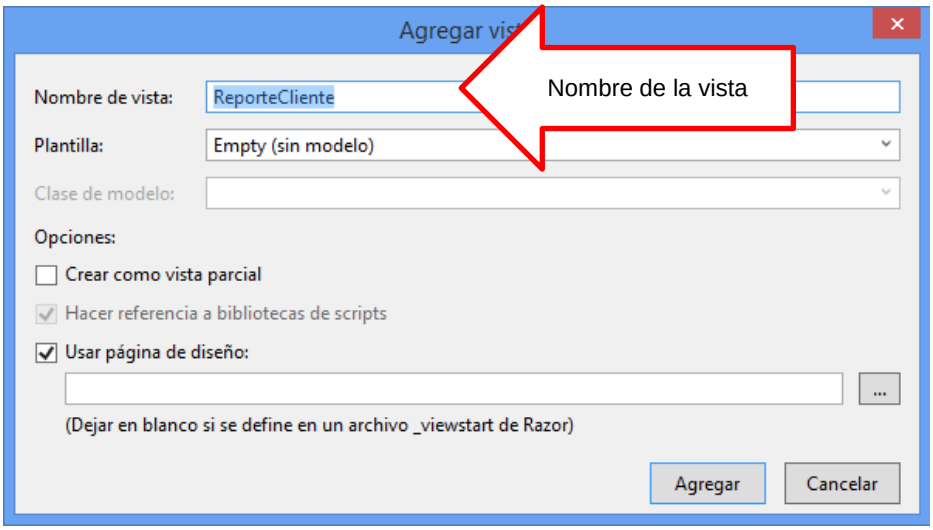
Importar las librerías para el proceso



Implementa el ActionResult del proceso, almacenando el valor del ReportViewer llamado rp en el ViewBag, tal como se muestra

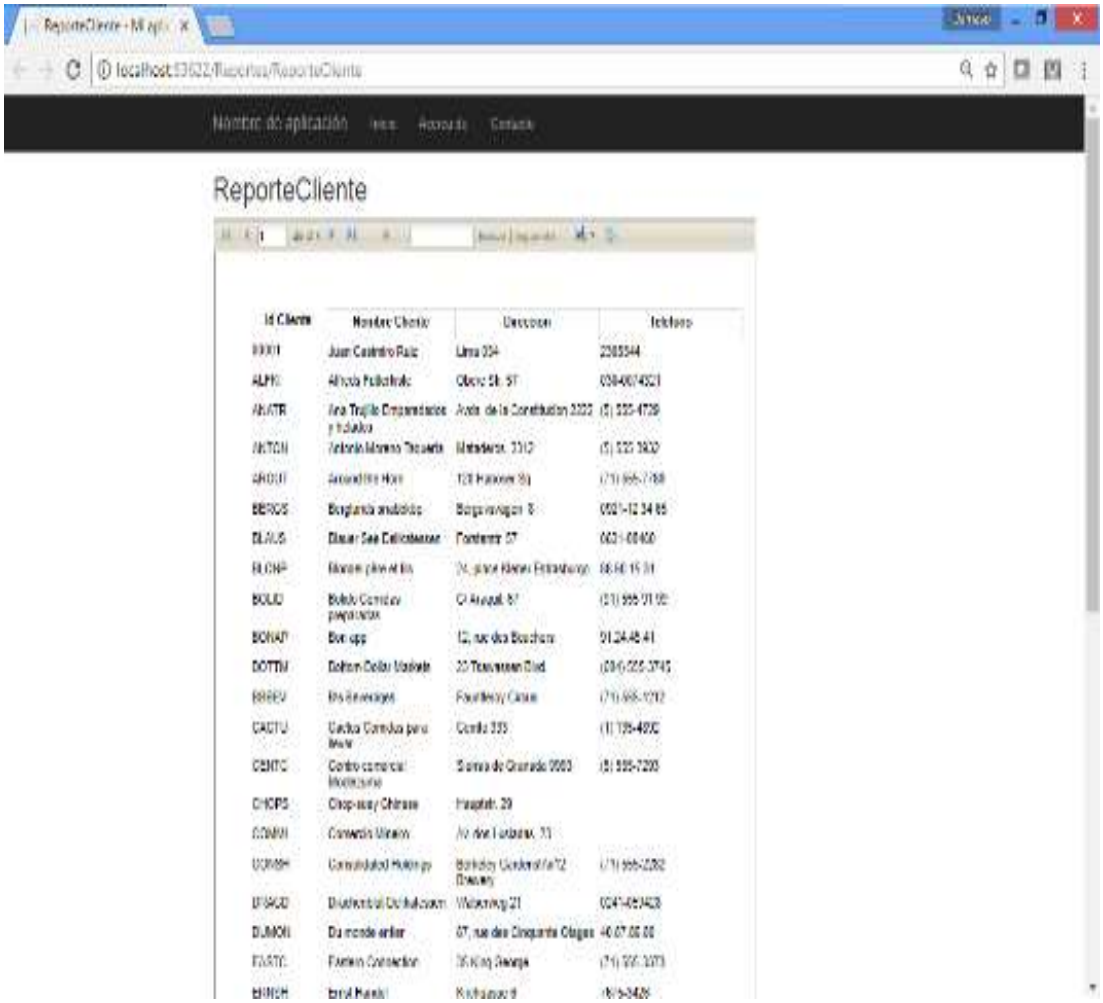


Agregar la vista del reporte, la plantilla estará vacia.



Agregar la librería ReportViewerForMVC y Html.ReportViewer para visualizar el contenido del archivo

Ejecuta la página, la cual visualiza el reporte de los clientes



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.

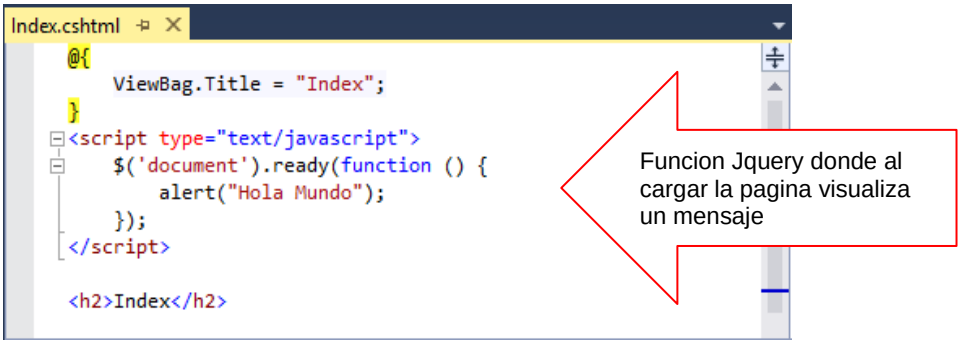
jQuery comparte con Prototype muchas ideas e incluso dispone de funciones con el mismo nombre. Sin embargo, su diseño interno tiene algunas diferencias drásticas respecto a Prototype, sobre todo el "encadenamiento" de llamadas a métodos.

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 ejecutaruna 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".



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 catagorias 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 elemento 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
:heder	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 moved 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.

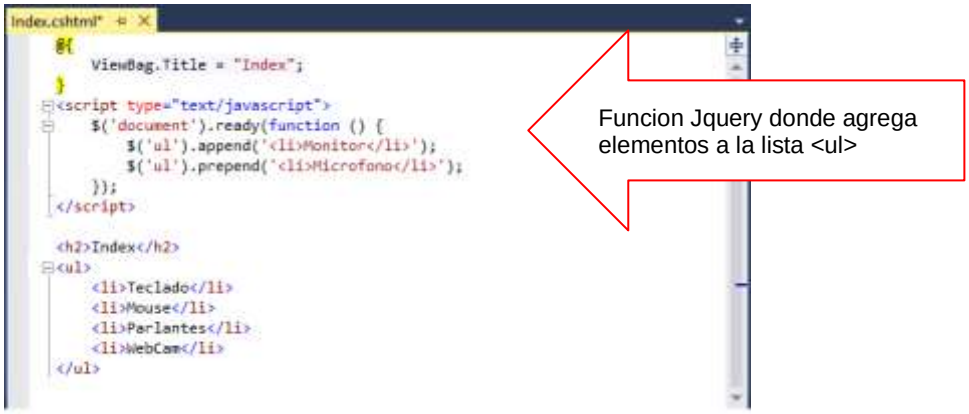
Un ejemplo es asignarle a la etiqueta <img> los atributos del origen y alt

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



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	Devuelte 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 padre 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

```
Index.cshtml*
@{
    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()**.

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 agrega '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 especifica.

```
$.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.

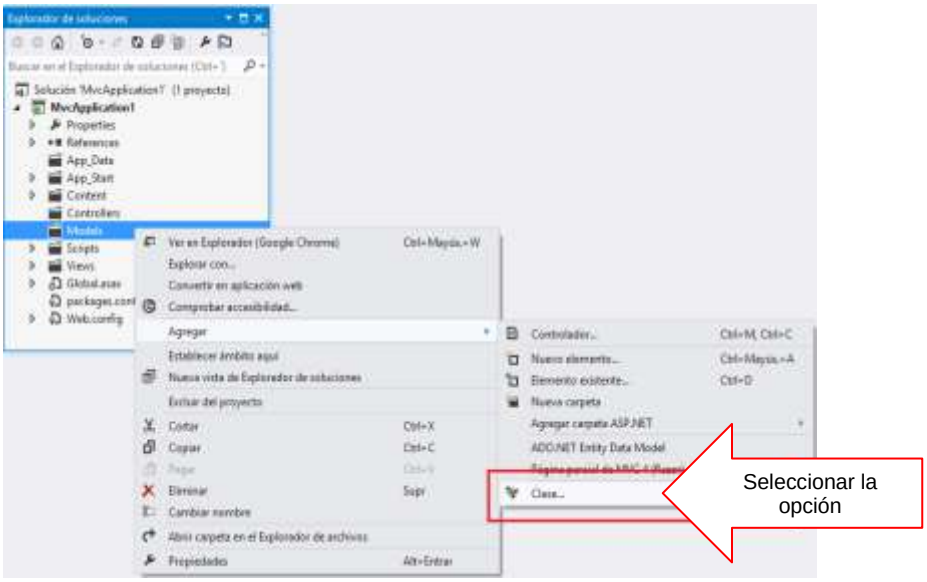
```
$('document').ajaxSuccess(function () {
    alert('Detenido');
});
```

Laboratorio
Creando una aplicación ASP.NET MVC y Jquery

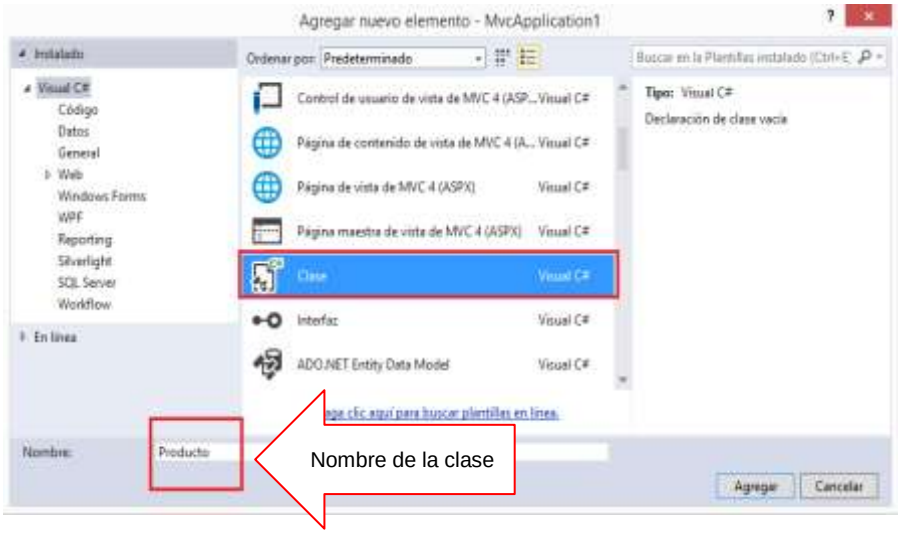
Implemente un proyecto ASP.NET MVC donde permita realizar operaciones de listado y actualización de productos utilizando anotaciones y ventanas de dialogo en jquery.

Trabajando con el Modelo

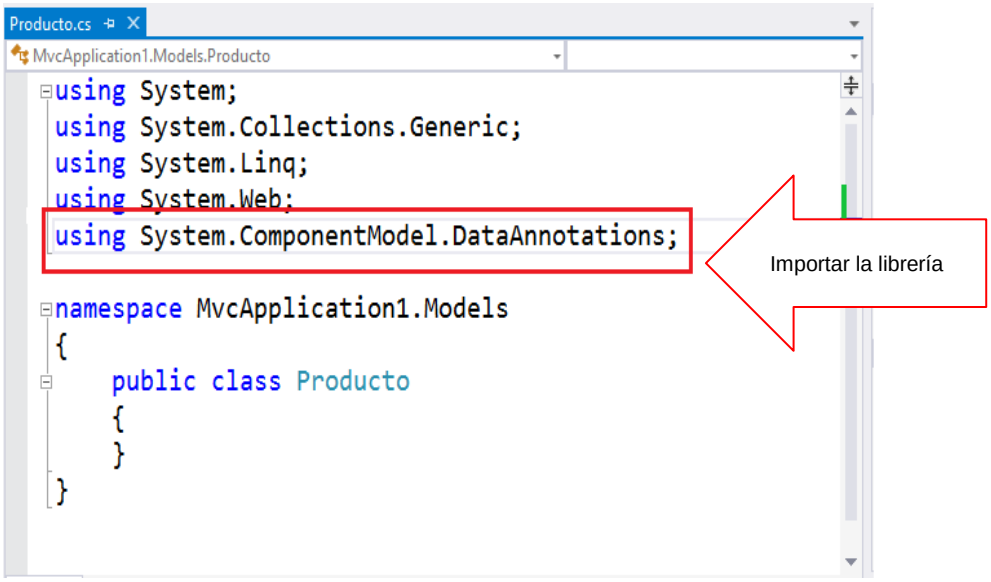
Primeramente definimos la clase en la carpeta Models: Agregar una clase llamada Producto



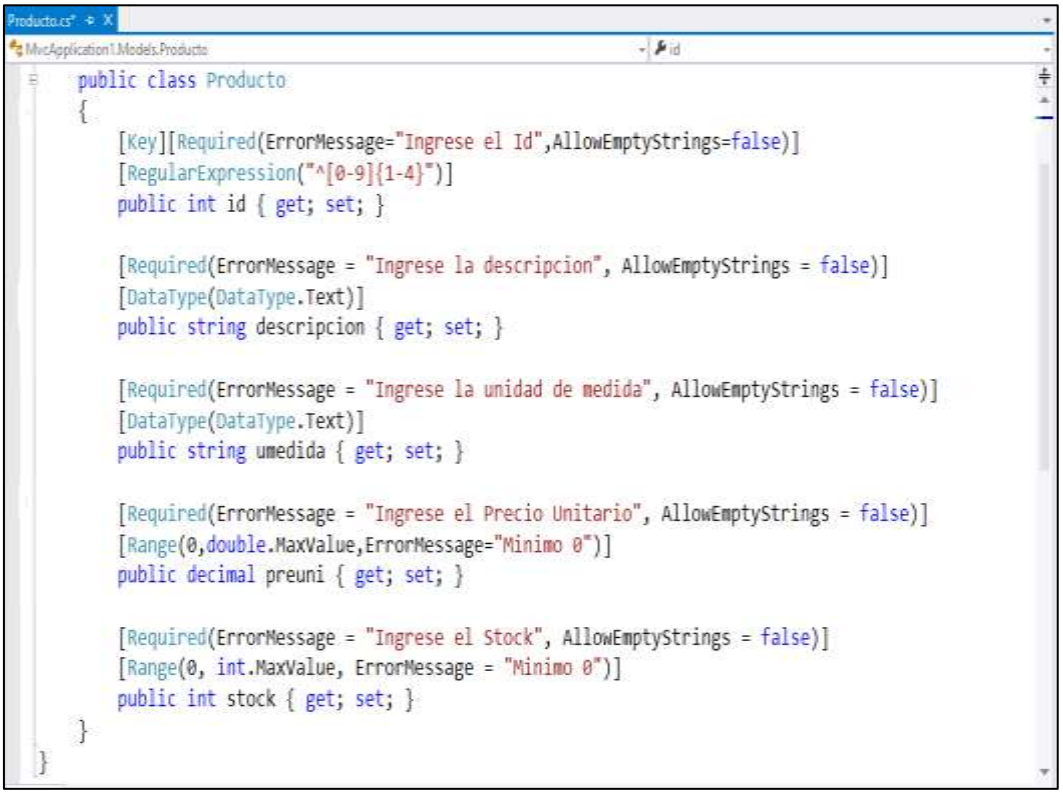
En la plantilla selecciona Clase y asigne su nombre: Producto, presiona el botón Agregar



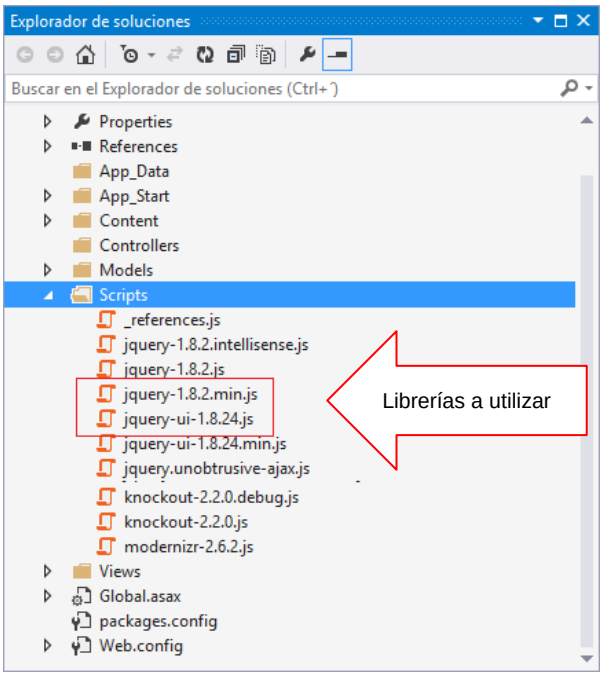
En la clase defina la librería de Anotaciones y Validaciones



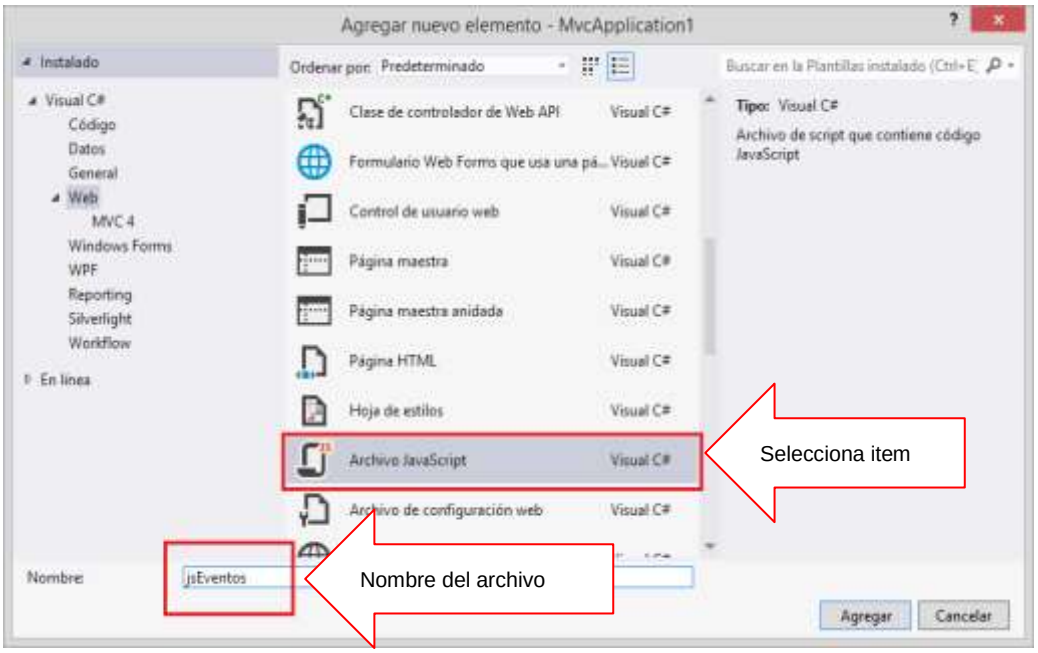
A continuación defina la estructura de la clase Producto, validando el ingreso de sus datos.



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();
        });
});
```

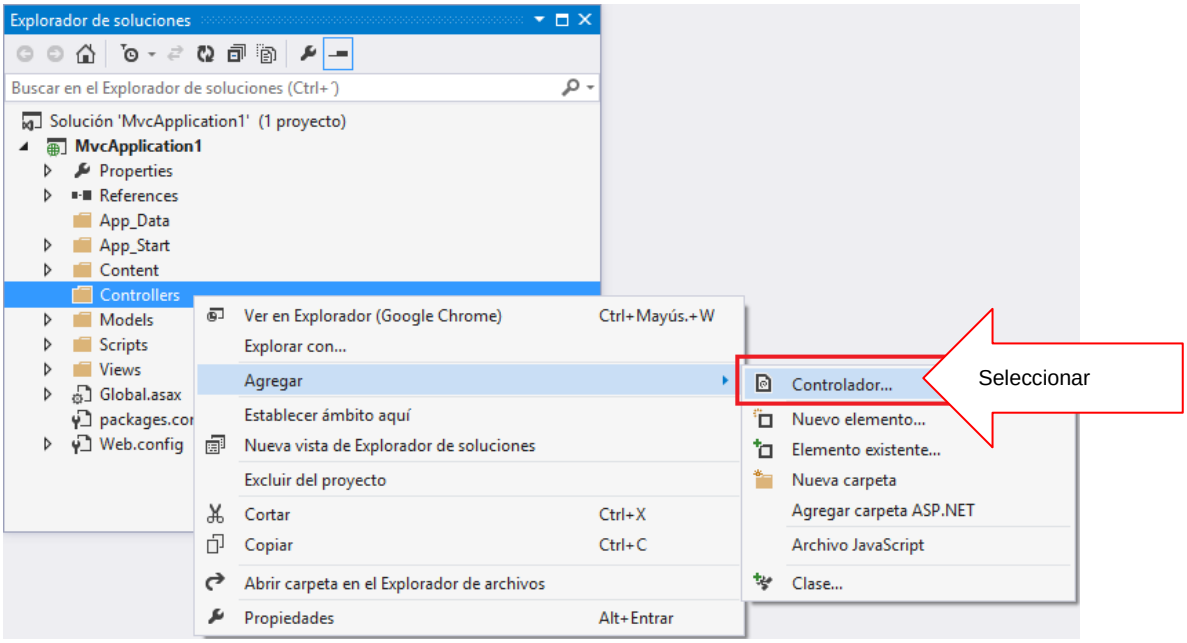
En el _Layout.cshtml, agregue las librerías de trabajo, tal como se muestra y comentar la línea @Script.Render.

```
<!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>
```

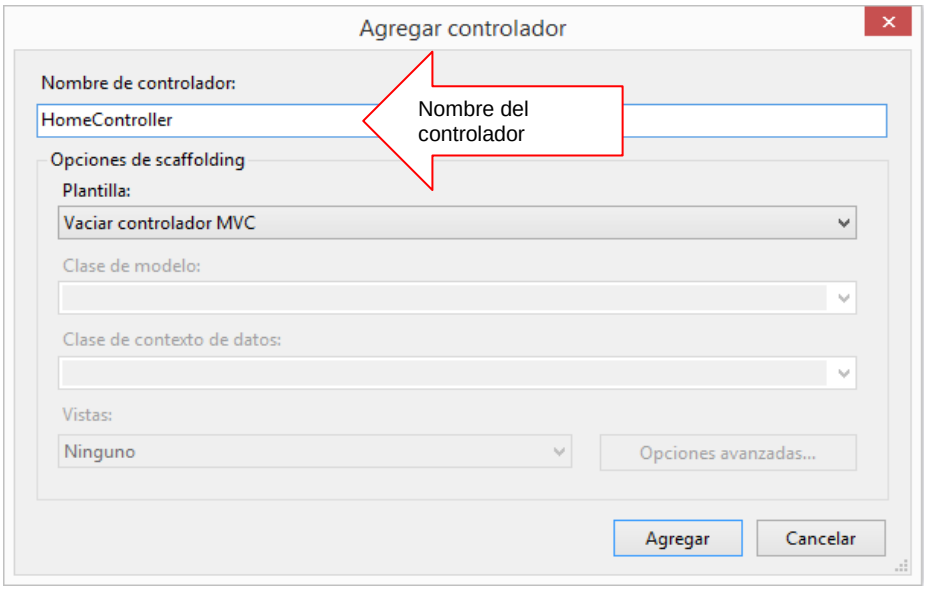
Agregar las librerías al _Layout

Comentar la línea

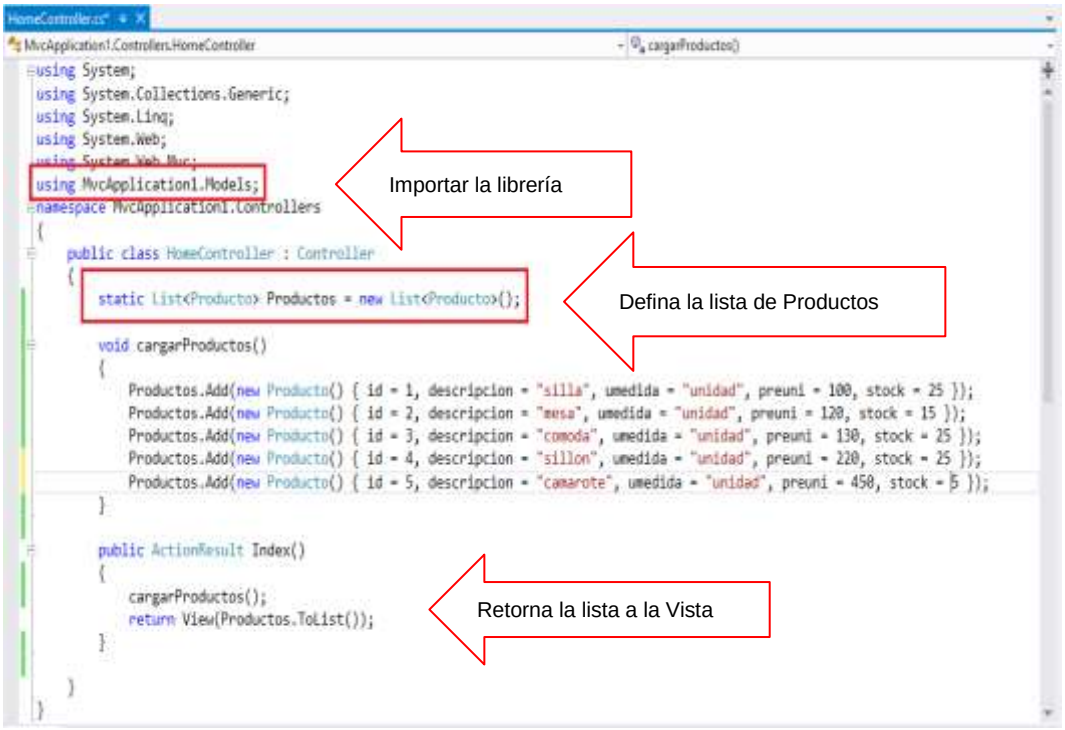
A continuación agregamos, en la carpeta Controllers, un controlador llamado Home, tal como se muestra.



Agregar el controlador llamado HomeController, tal como se muestra

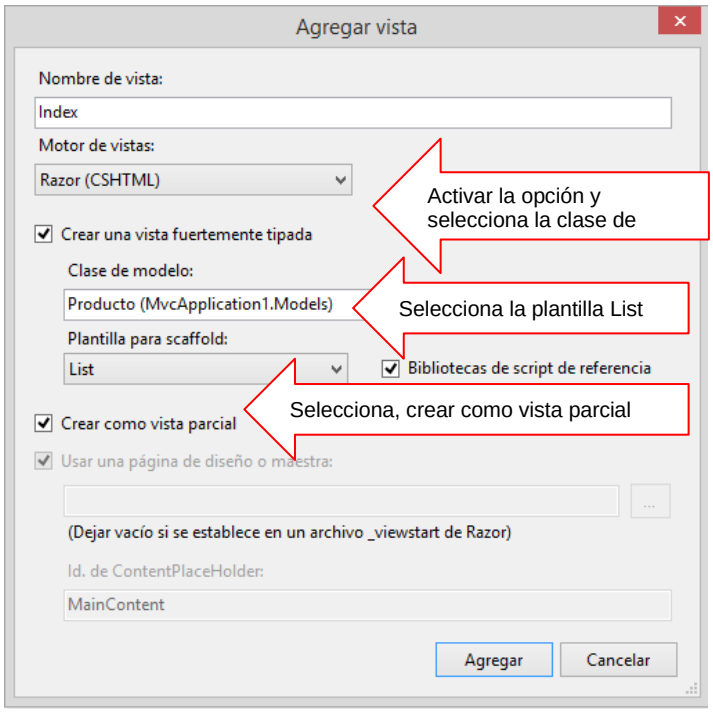


En el controlador Home, primero importamos la librería Models.
A continuación definimos la lista de Producto, llamada Productos, tal como se muestra.
Definimos un metodo para cargar los datos a los productos llamado cargarProductos()
En el Action Index, enviamos la lista de Productos.



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 boton Agregar

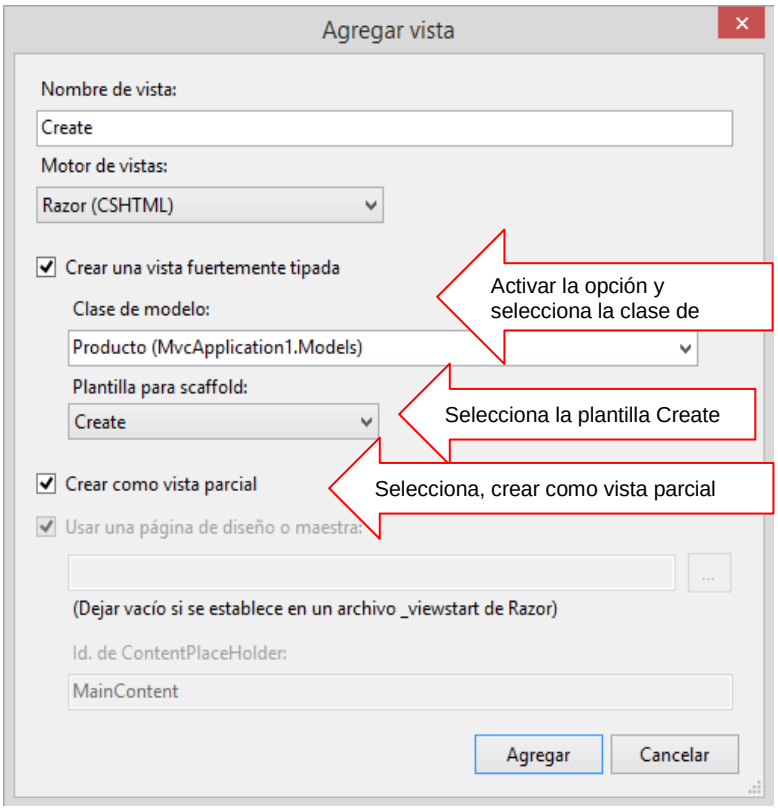


A continuación se diseña la Vista Index, donde aparece en el encabezado el modelo de la clase, la lista de productos y las opciones para Agregar, Editar, Actualizar y Eliminar

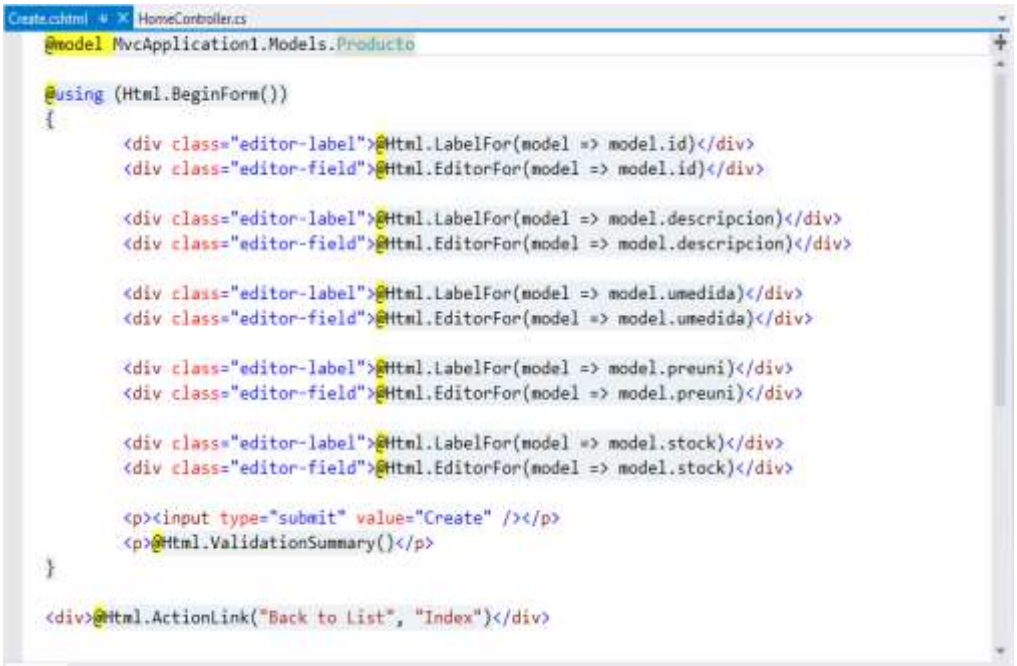
Action Create

En el controlador defina la opción Create, el cual agrega un nuevo producto a la Lista

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



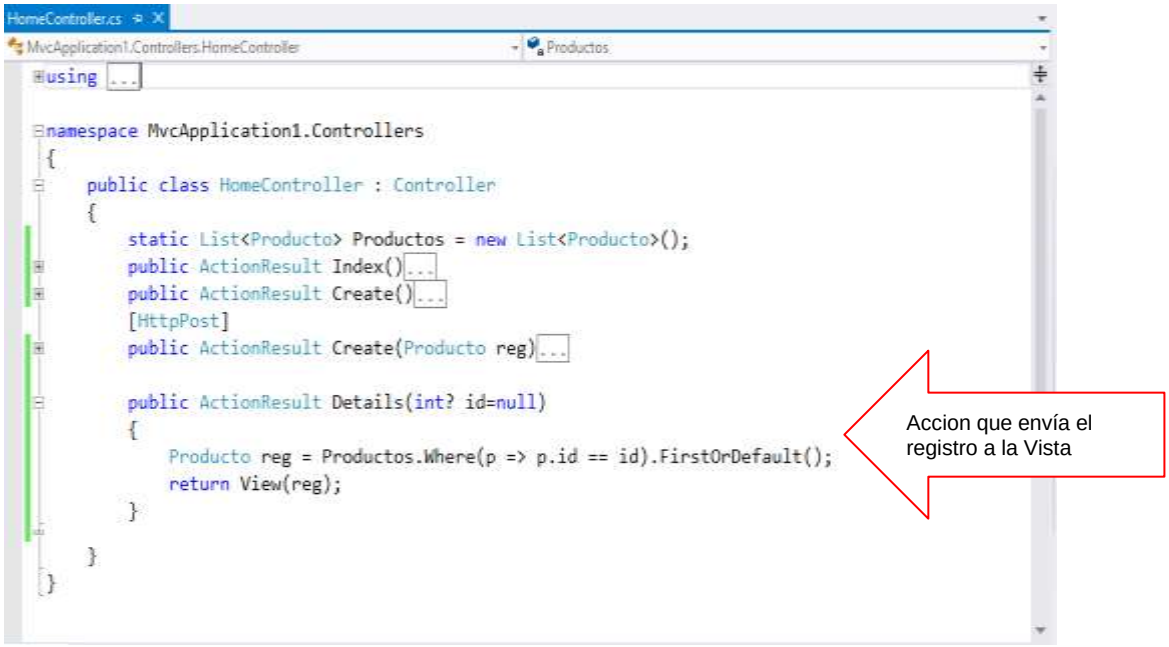
En la vista Create, diseñarla tal como se muestra en la figura.



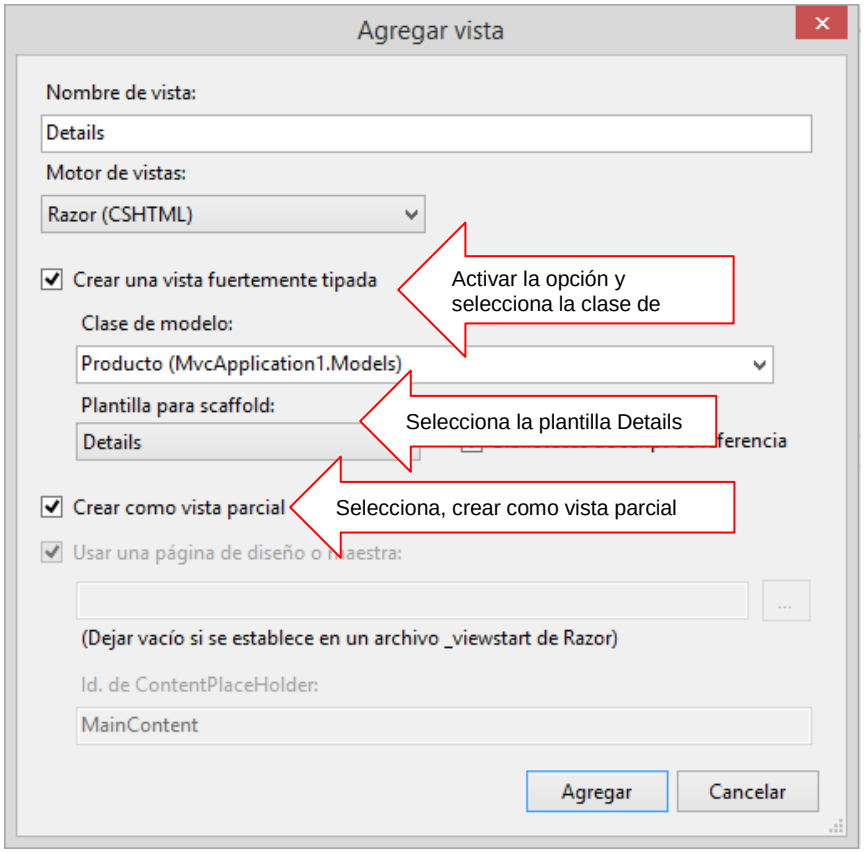
```
@model MvcApplication1.Models.Producto

@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.



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

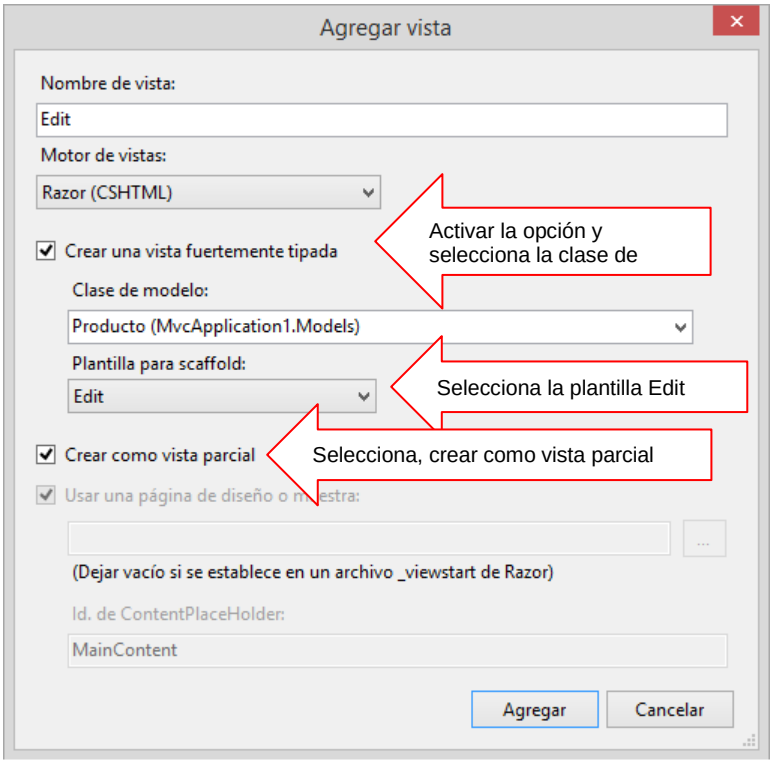
En el controlador defina la acción Edit, el cual retorna el registro de Productos por su campo id, y recibe el registro para actualizar sus datos. Defina el proceso tal como se muestra

```
        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");
        }

    }
}
```

Accion que envía el registro a la Vista

Accion que recibe el registro de la Vista y actualiza los datos

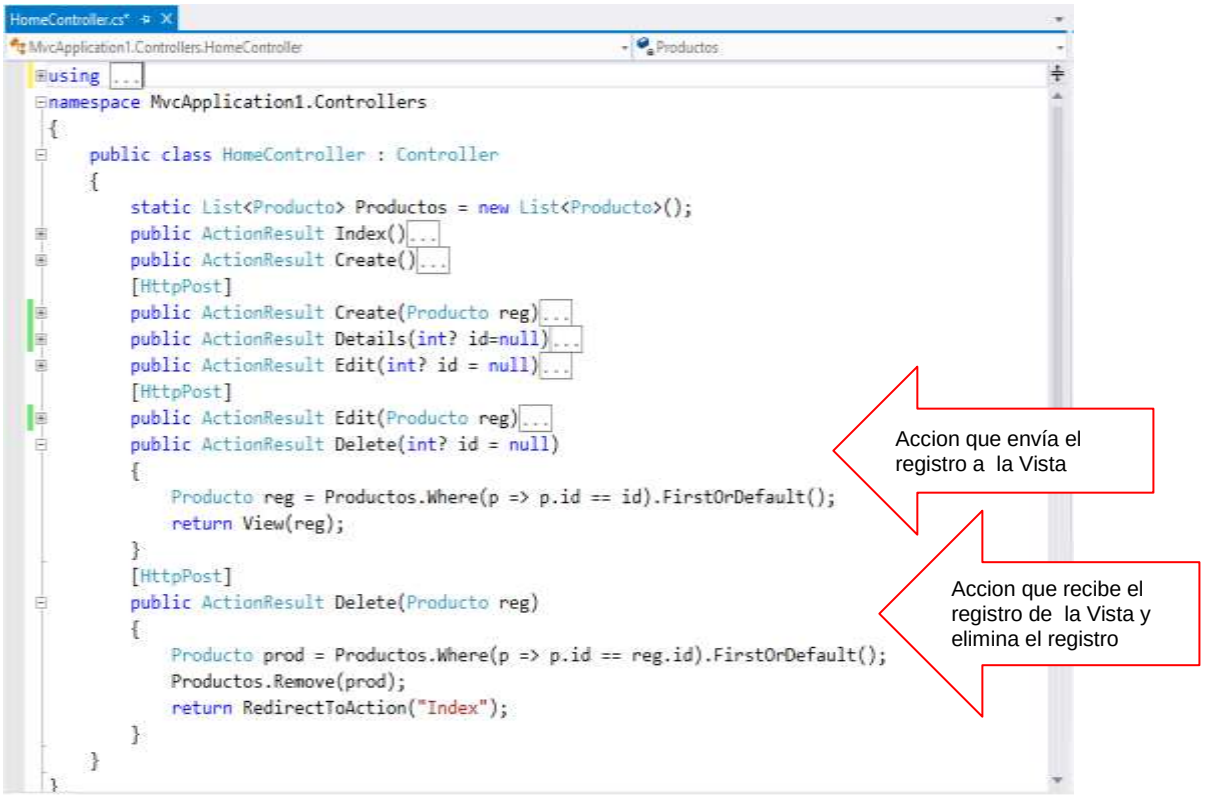
A continuación agrega la Vista a la acción Edit, tal como se muestra.



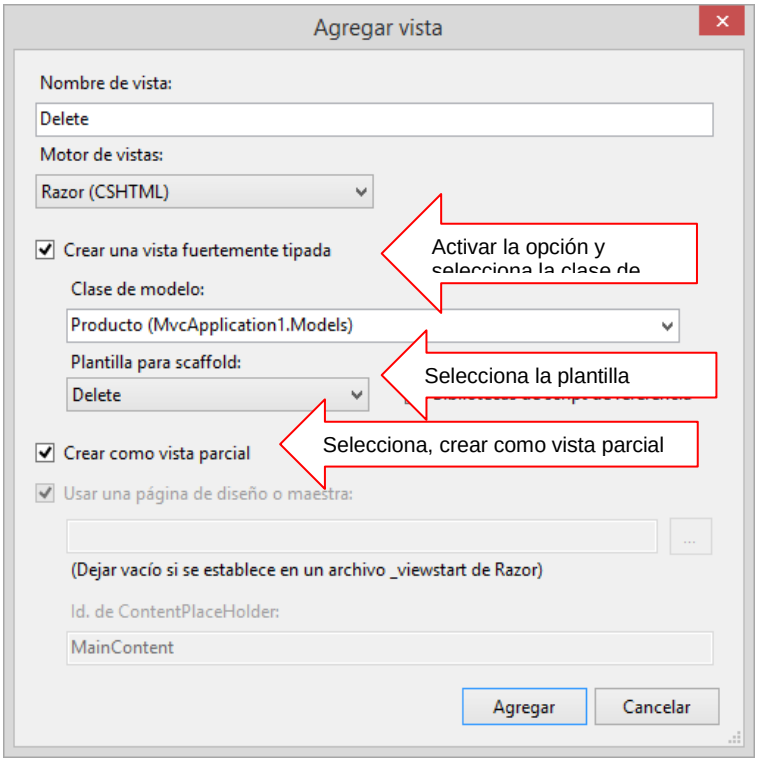
A continuación diseña la Vista, tal como se muestra.

Action Delete

En el controlador defina la acción Delete, el cual retorna el registro de Productos por su campo id, y confirmas la eliminación del registro por su id. Defina el proceso tal como se muestra



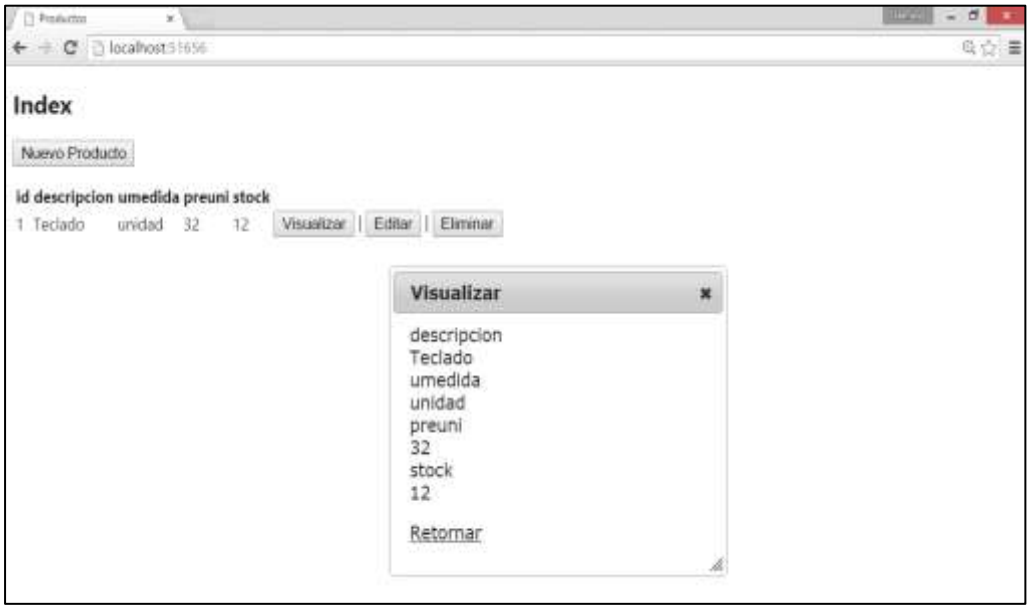
A continuación agrega la Vista Delete, tal como se muestra



Defina la vista tal como se muestra.

```
Delete.cshtml  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

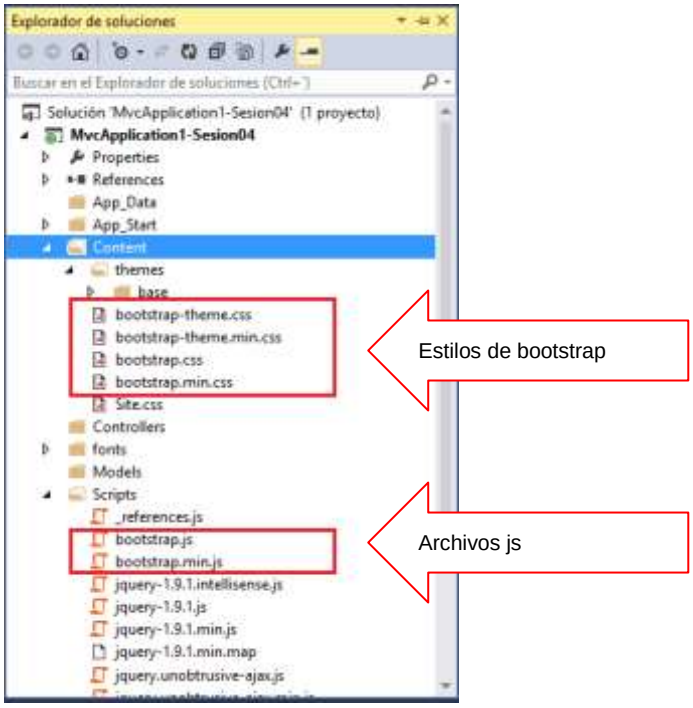
Bootstrap proporciona un conjunto de hojas de estilo que proveen definiciones básicas de estilo para todos los componentes de HTML. Esto otorga una uniformidad al navegador y al sistema de anchura, da una apariencia moderna para el formateo de los elementos de texto, tablas y formularios.

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).



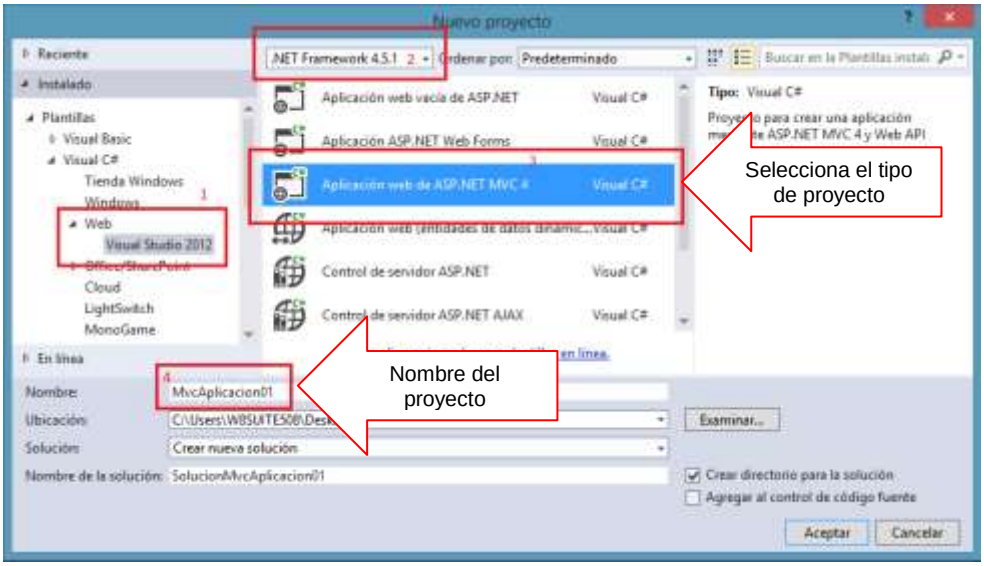
Aplicación ASP.NET MVC con Bootstrap

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:

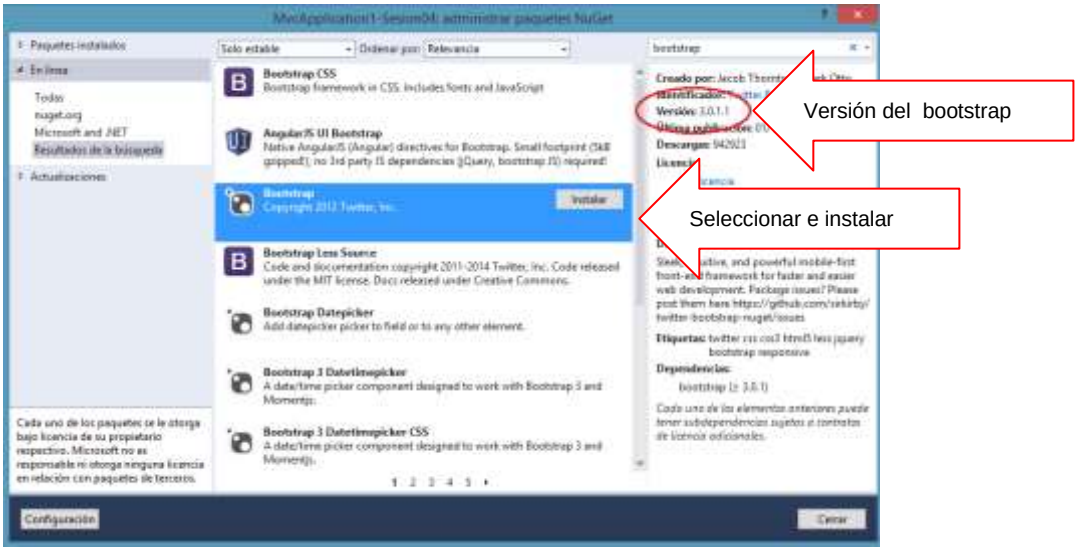
1. Seleccionar el proyecto Web Visual Studio 2012
2. Seleccionar el FrameWork: 4.5.1
3. Seleccionar la plantilla Aplicación web de ASP.NET MVC 4
4. Asignar el nombre del proyecto
5. Presionar el botón ACEPTAR



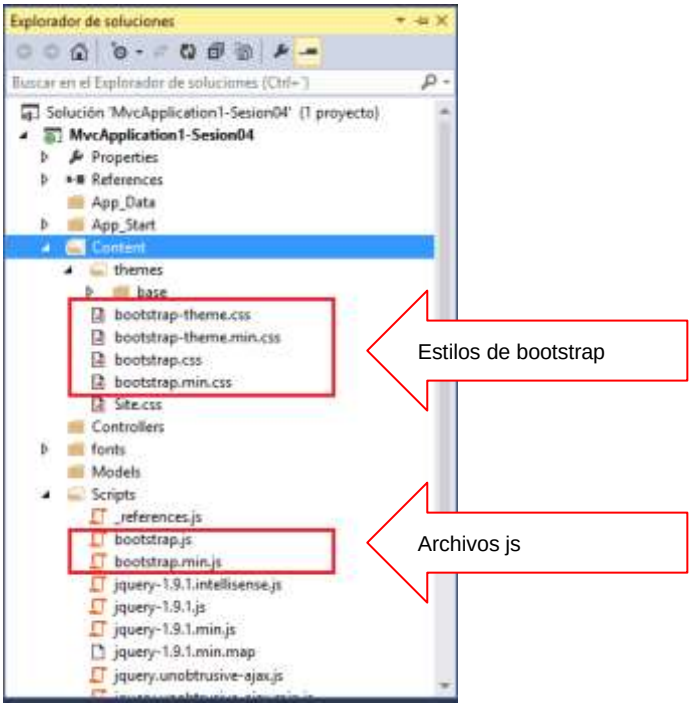
A continuación, seleccionar la plantilla del proyecto:

- Seleccionar la plantilla Básico
- Seleccionar el motor de vistas: Razor

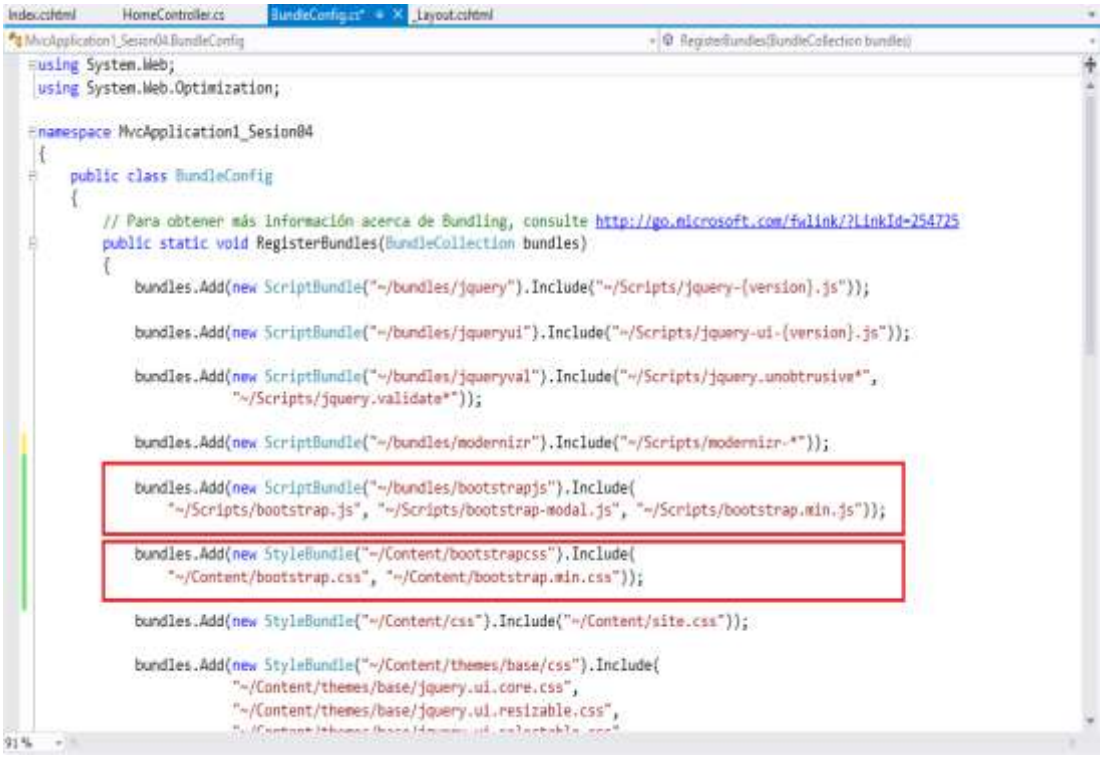
En el proyecto agregar el FrameWork Bootstrap para ser implementado en la aplicación ASP.NET MVC, tal como se muestra



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.Rendery Scripts.Render



```
using System.Web;
using System.Web.Optimization;

namespace MvcApplication1_Sesion04
{
    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",
```

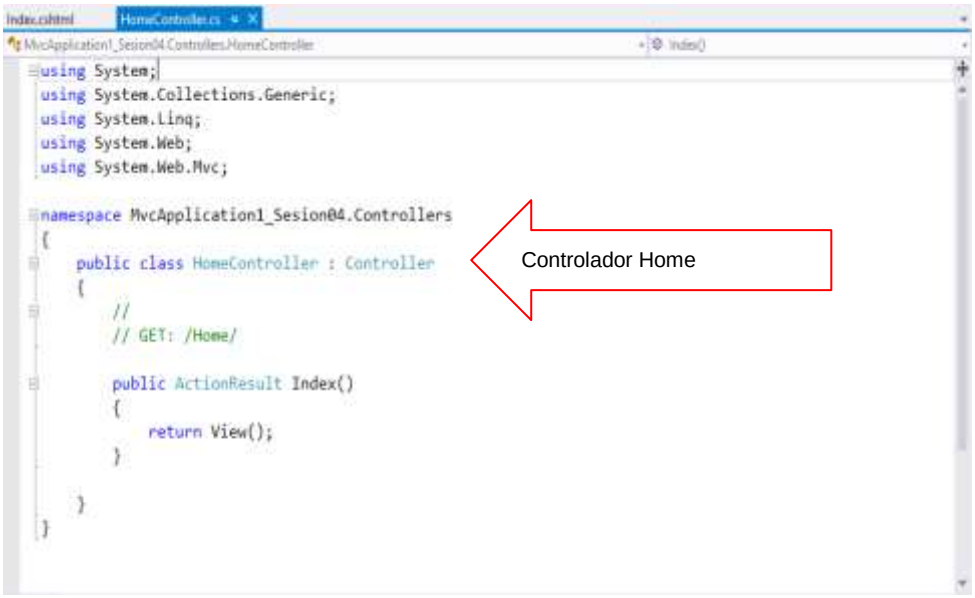
En la pagina _Layout.cshtml, referenciamos las librerías utilizando @Scripts. Render y @Styles.Render, tal como se muestra

```
<!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")

    @* @Styles.Render("~/Content/css")*@
    @Scripts.Render("~/bundles/modernizr")
    @Styles.Render("~/Content/bootstrapcss")
</head>
<body>
    @RenderBody()

    @RenderSection("scripts", required: false)
</body>
</html>
```

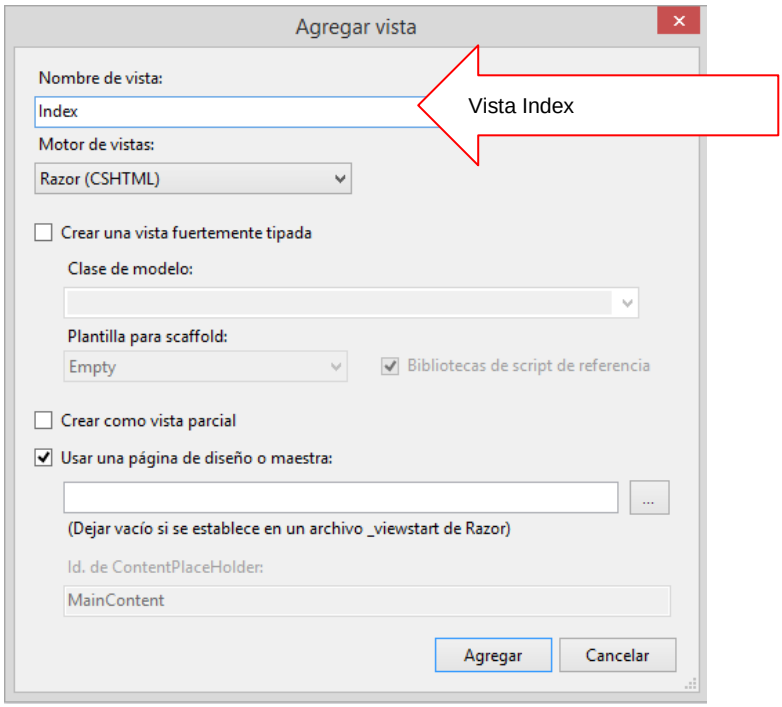
Referenciar a las librerías .js

Referenciar a las librerías .css

En la carpeta Controllers, agrega un controlador llamado Home.

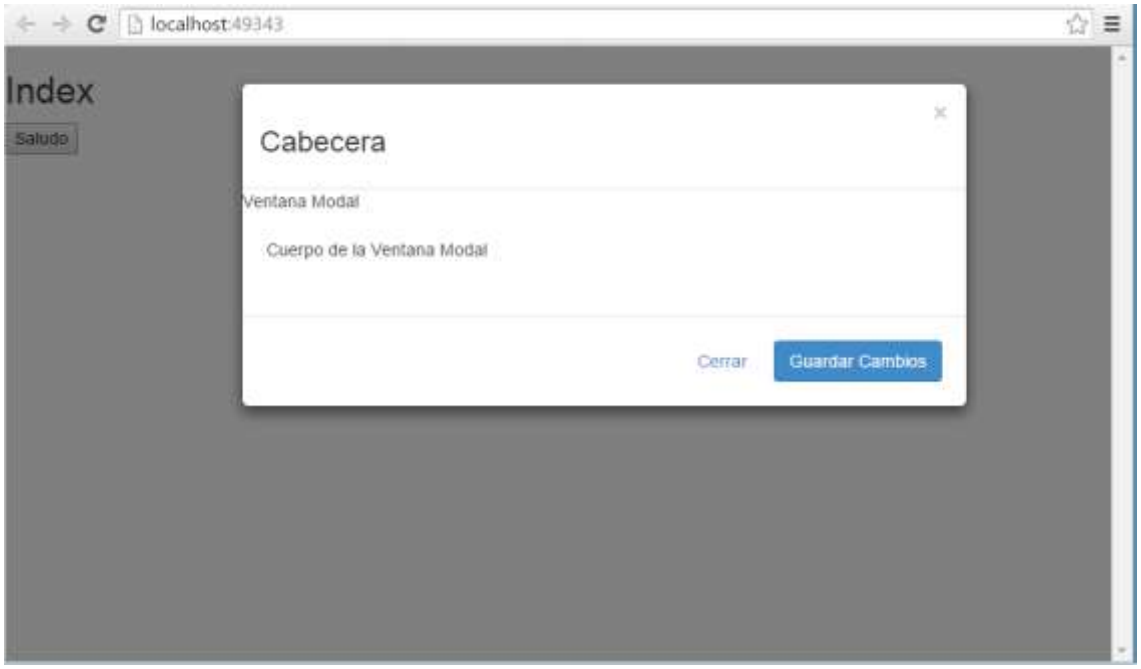


En la acción Index, agrega una vista, tal como se muestra.



En la ventana Index, agrega el código script, tal como se muestra

Ejecuta el proyecto, presionando F5. Al presionar el boton Saludo, se visualiza una ventana Modal, tal como se muestra.



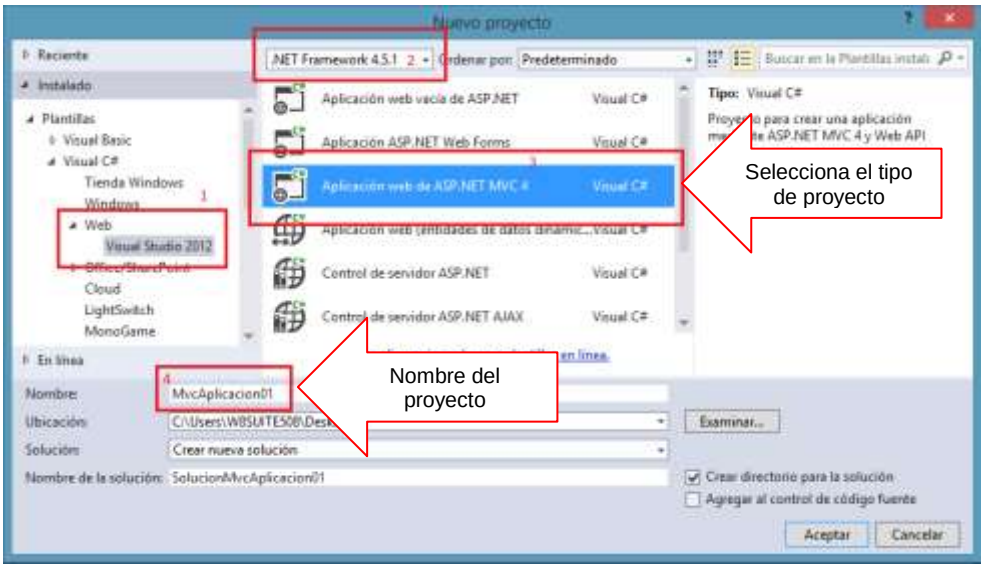
Aplicación ASP.NET MVC con Bootstrap

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:

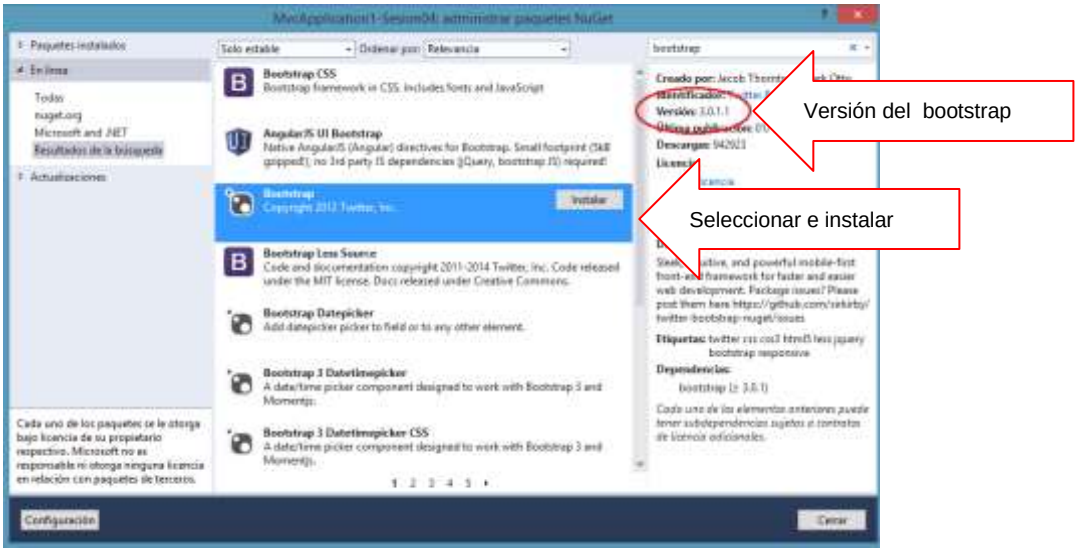
1. Seleccionar el proyecto Web Visual Studio 2012
2. Seleccionar el FrameWork: 4.5.1
3. Seleccionar la plantilla Aplicación web de ASP.NET MVC 4
4. Asignar el nombre del proyecto
5. Presionar el botón ACEPTAR



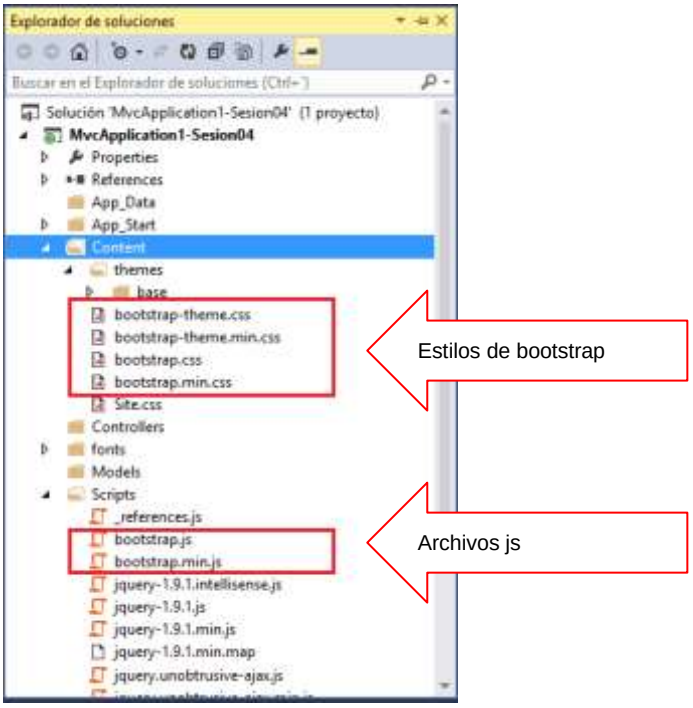
A continuación, seleccionar la plantilla del proyecto:

- 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

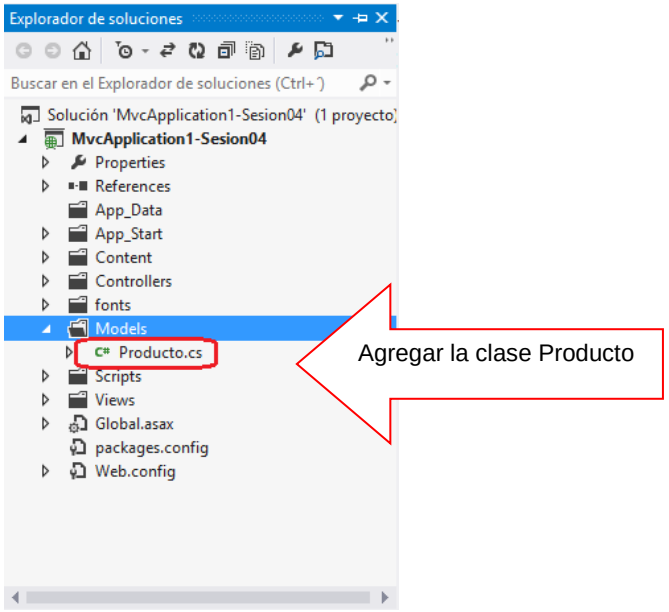


Instalado el framework, se podrá visualizar los archivos dentro del explorador de soluciones

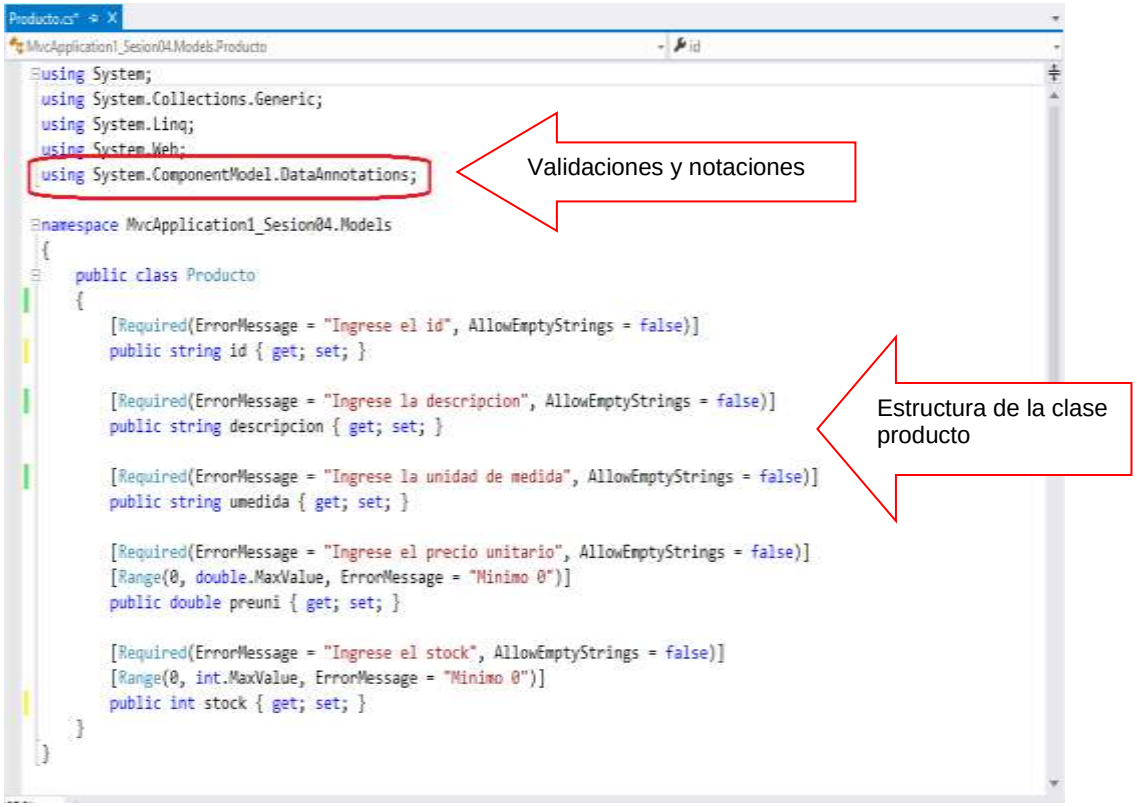


Trabajando con el modelo de datos

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



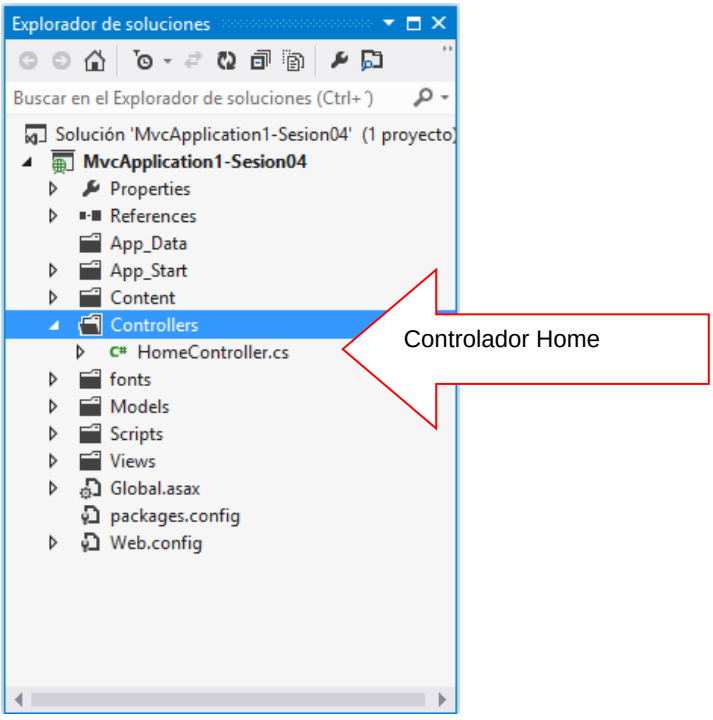
En el archivo compartido _Layout.cshtml, agregar los Scripts y Styles del bootstrap, tal como se muestra

_Layout.cshtml HomeController.cs

```
<!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>
```

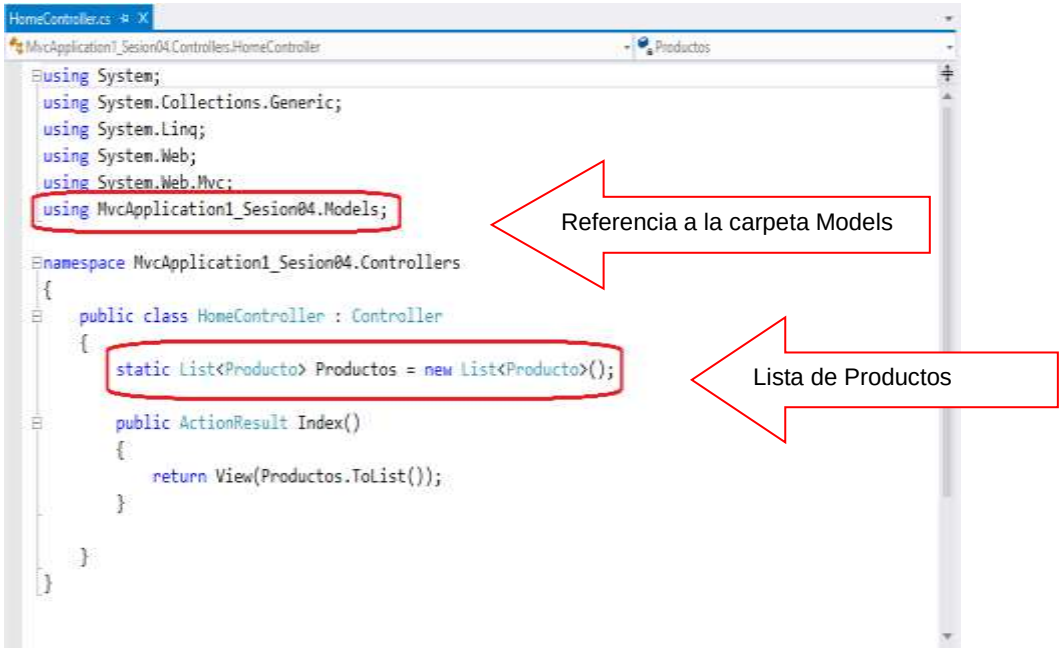
Trabajando con el Controlador y las Vistas

En la carpeta Controllers, agregar el controlador llamado HomeController, tal como se muestra en la figura.

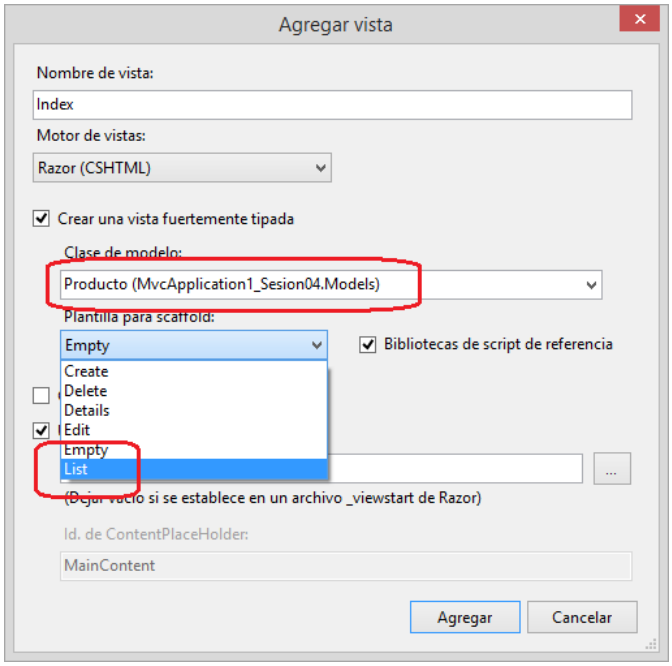


ActionResult Index

En la ventana del controlador, defina el ActionResult Index, el cual retorna la lista de los registros de Producto.



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.

```
@model IEnumerable<MvcApplication1_Sesion04.Models.Producto>

@{ViewBag.Title = "Index";}

<h2>Index</h2>

<p>@Html.ActionLink("Create New", "Create")</p>
<table>
    <tr>
        <th>Codigo</th>
        <th>Descripcion</th>
        <th>Unidad de Medida</th>
        <th>Precio Unitario</th>
        <th>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>
            @Html.ActionLink("Edit", "Edit", new { id=item.id }) |
            @Html.ActionLink("Details", "Details", new { id=item.id })
        </td>
    </tr>
}
</table>
```

Codifica en la vista Index, para trabajar con ventanas modales en los procesos

```
@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>
    </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>

<table>...</table>
```

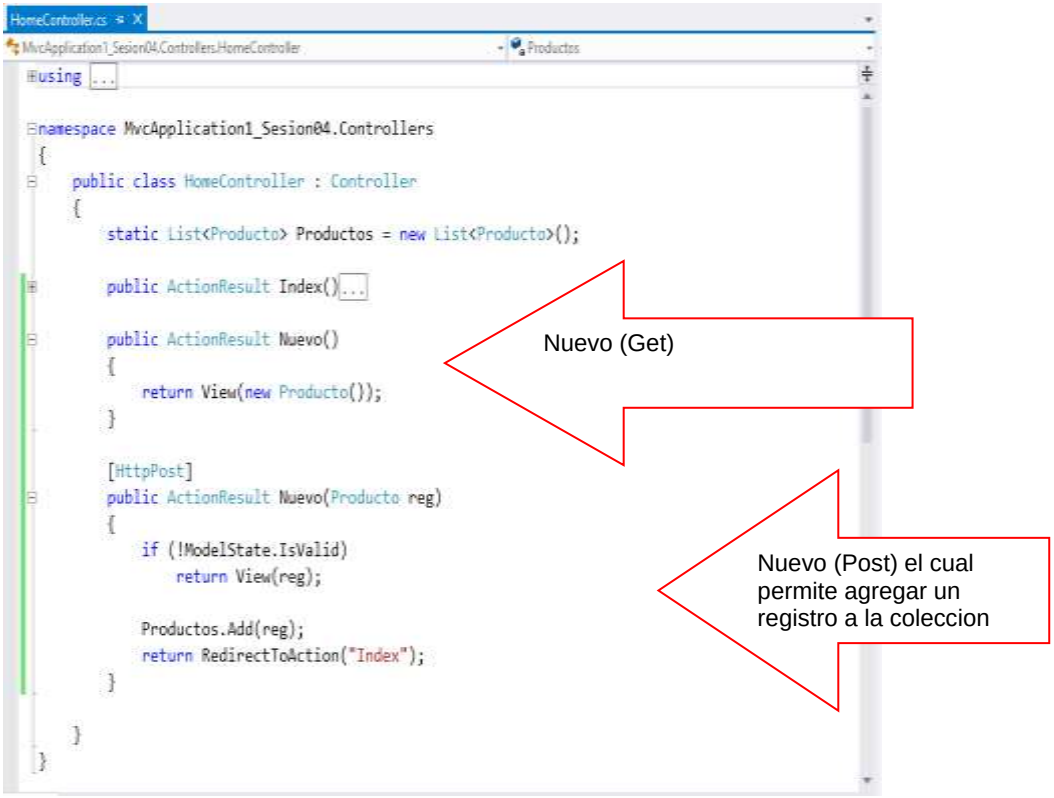
Script para programar el evento Click de btnNuevo

Bloque donde se muestra la ventana modal al ejecutar un proceso

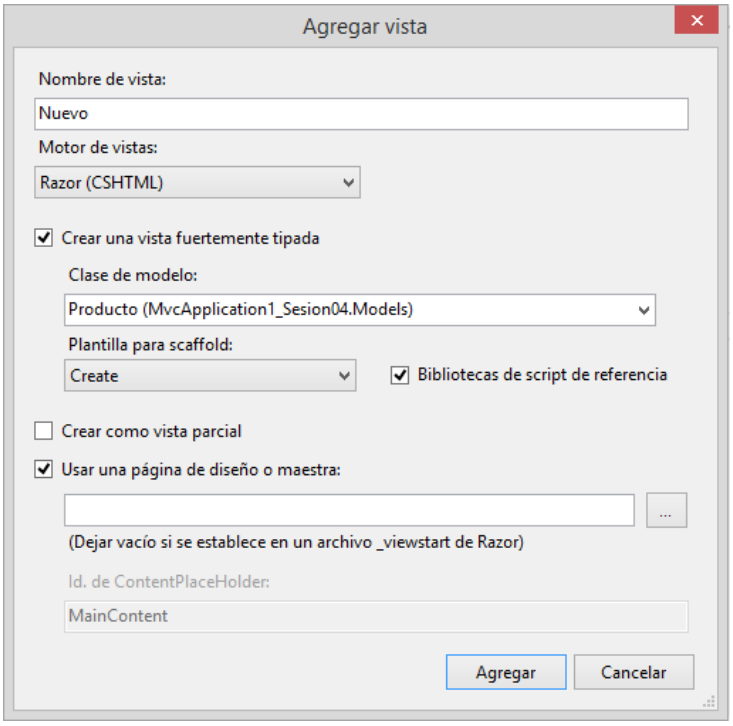
Etiqueta btnNuevo

Trabajando con el ActionResult Nuevo

A continuación defina el ActionResult Nuevo, tal como se muestra



A continuación agregamos la Vista a la acción Nuevo, donde seleccionamos la clase Producto y la plantilla será Create, tal como se muestra

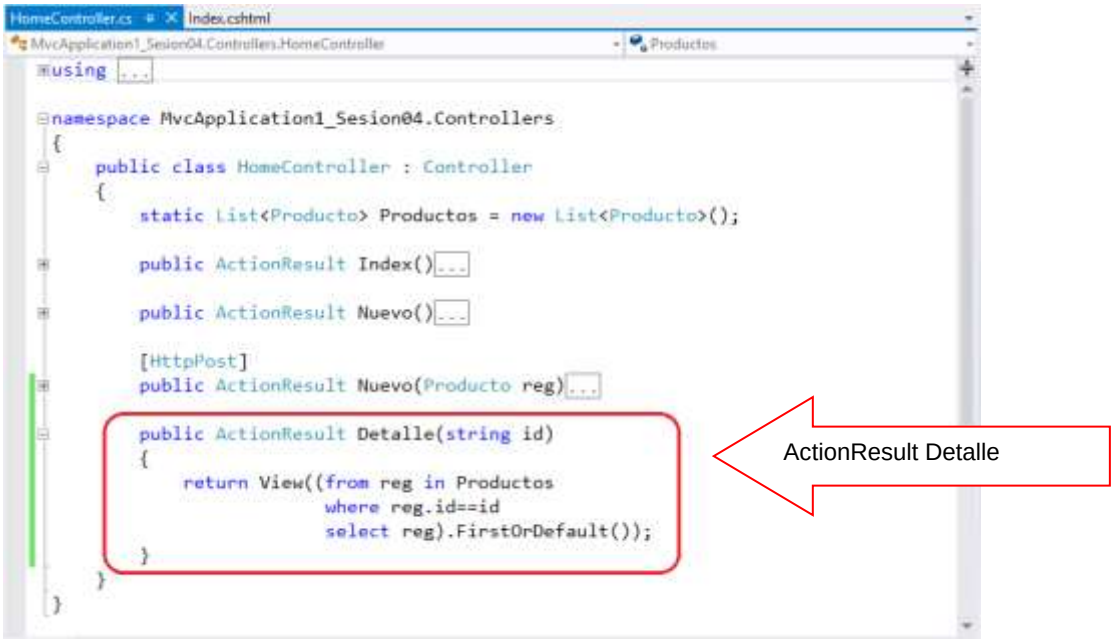


En la vista Nuevo, realizar los cambios a la vista, tal como se muestra

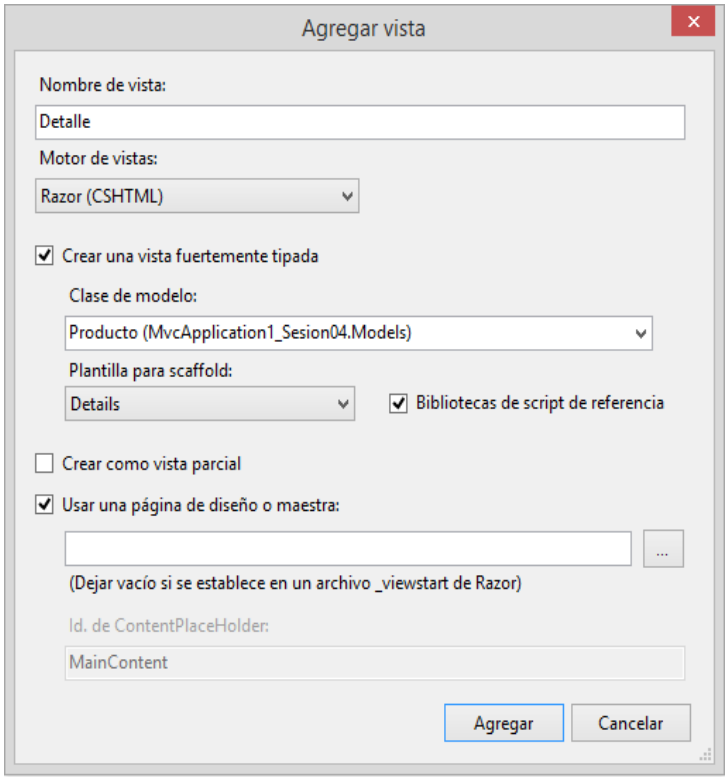
Para verificar el proceso, presiona la tecla F5, al presionar el boton Nuevo, se muestra una ventana modal para el ingreso de productos. Ingrese los datos, al presionar el boton Create, se cierra la ventana modal y se visualiza la ventana principal con el registro agregado

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.



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)...);

        $("#btnDetalle").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>...</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 { id=item.id })
        </td>
    </tr>
}
</table>
```

Evento del boton btnDetalle

Objeto btnDetalle

Para comprobar su funcionalidad, presionar F5 para ejecutar el proyecto. Al presionar la opción Detalle, se visualiza los datos del producto seleccionado.