



Programación Orientada a Objetos

II

Curso	Programación Orientada a Objetos II (4692)
Formato	Manual de curso
Autor Institucional	Cibertec
Páginas	275 p.
Elaborador	López Aragón, Dámaso

Índice

Presentación	5
Red de contenidos	6
UNIDAD DE APRENDIZAJE 1: INTRODUCCIÓN AL ASP.NET MVC	7
1.1 Tema 1 : ASP.NET MVC	8
1.1.1 : Introducción	8
1.1.2 : Plataforma de ASP.NET MVC	9
1.1.2.1 : Métodos y selectores del Action del Controlador	10
1.1.2.2 : Verbos: HttpGet y HttpPost	13
1.1.2.3 : Transferencia de datos: ViewBag, ViewData y TempData	16
1.1.3 : Estructura del MVC: Modelo, Vista y Controlador	17
1.1.4 : URL de enrutamiento	19
Laboratorio 1.1.: Creando una aplicación ASP.NET MVC	21
1.2 Tema 2 : Manejo de vistas	36
1.2.1 : Introducción	36
1.2.2 : Sintaxis Razor y scaffolding	37
1.2.3 : Vista layout	38
1.2.4 : Vistas parciales	39
Laboratorio 2.1.: Trabajando con Vistas Parciales	41
UNIDAD DE APRENDIZAJE 2: TRABAJANDO CON DATOS EN ASP.NET MVC	58
2.1 Tema 3 : Introducción a ADO.NET	60
2.1.1 : Arquitectura, proveedor de datos en ADO.NET	61
2.1.2 : Trabajando una conexión a un origen de datos	65
2.1.3 : Publicación de una cadena de conexión	66
Laboratorio 3.1.: Creando Procedimiento Almacenado en el SQL Server	67
Laboratorio 3.2.: Creando el procedimiento almacenado	80
2.2 Tema 4 : Recuperación de datos y paginación	87
2.2.1 : Clase DataReader: métodos y propiedades	87
2.2.2 : Consulta de datos con parámetros utilizando DataReader	89
2.2.3 : Trabajando con vistas parciales en una consulta de datos	89
2.2.4 : Paginación de datos recuperados	91
Laboratorio 4.1.: Ejecutando consulta de datos con parámetro	91
Laboratorio 4.2.: Ejecutando consulta de datos con dos parámetros	102
Laboratorio 4.3.: Paginación de los registros recuperados	106
Laboratorio 4.4.: Búsqueda y paginación de los registros recuperados	111
2.3 Tema 5 : Manipulación de datos	117
2.3.1 : Validaciones de datos: uso de DataAnnotations	117

2.3.2 : Operaciones de actualización sobre un origen de datos, manejo de la clase Command	119
2.3.3 : Trabajando con imágenes: uso de la clase File y HttpPostFile	123
2.3.4 : Manejo de transacciones, uso de la clase Transaction	125
Laboratorio 5.1.: Actualizando datos con procedimientos almacenados	127
Laboratorio 5.2.: Manejo de Transacciones	148
2.4 Tema 6 : Arquitectura de capas con acceso a datos	167
2.4.1 : Introducción	167
2.4.2 : Implementando una arquitectura de capas en un proceso de actualización de datos	170
Laboratorio 6.1.: Actualizando datos en una Arquitectura de Capas DDD	172
UNIDAD DE APRENDIZAJE 3: IMPLEMENTANDO E-COMMERCE EN ASP.NET MVC	206
3.1 Tema 7 : Implementando una aplicación e-commerce	207
3.1.1 : Proceso del e-commerce	209
3.1.2 : Persistencia de datos: uso del objeto Session, TempData	211
3.1.3 : Implementando el proceso del e-commerce: diseño, acceso a datos, lógica de negocios, validación de usuario y transacciones	213
Laboratorio 7.1.: Implementando el proceso del e-commerce en ASP.NET MVC	214
UNIDAD DE APRENDIZAJE 4: TRABAJANDO CON ANGULAR EN ASP.NET	242
4.1 Tema 8 : AngularJS	243
4.1.1 : Introducción al Angular	243
4.1.2 : Integrando Angular en ASP.NET MVC	246
Laboratorio 8.1.: Implementando “Bienvenido” en una aplicación ASP.NET MVC	250
Laboratorio 8.2.: Ingreso de datos en una aplicación ASP.NET MVC	257
Laboratorio 8.3.: Listar una colección de datos en una aplicación ASP.NET MVC	261
Laboratorio 8.4.: Listar los registros de la tabla tb_clientes en una aplicación ASP.NET MVC	264
Laboratorio 8.5.: Insertar los registros a la tabla tb_clientes en una aplicación ASP.NET MVC	270
Bibliografía	275

Presentación

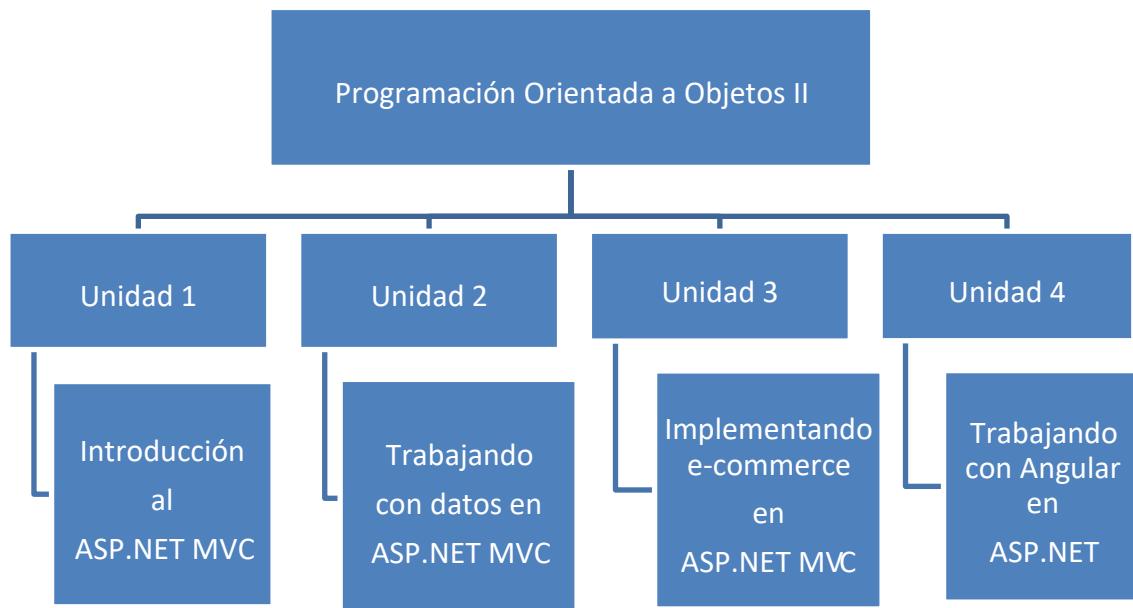
Visual Studio y su plataforma .NET FrameWork es una plataforma de desarrollo donde permite implementar desarrollos de software de manera rápida y robusta. ASP.NET, tanto en Web Form como en MVC, admiten crear aplicaciones en tiempo mínimo bajo una plataforma de librerías del .NET Framework. De esta manera, 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 permitirán 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 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 de datos, implementando dicho proceso en una aplicación de actualización de datos. En la tercera unidad implementaremos una aplicación E-Commerce para un proceso de ventas donde manejaremos datos de persistencia, validación de usuario y transacciones. Por último, desarrollamos un SPA en AngularJS utilizando ASP.NET MVC.

Red de contenidos





INTRODUCCIÓN AL ASP.NET MVC

LOGRO DE LA UNIDAD DE APRENDIZAJE

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

TEMARIO

1.1 Tema 1 : ASP.NET MVC

- 1.1.1 : Introducción
- 1.1.2 : Plataforma de ASP.NET MVC
- 1.1.3 : Estructura del MVC: Modelo, Vista y Controlador
- 1.1.4 : URL de enrutamiento

1.2 Tema 2 : Manejo de vistas

- 1.2.1 : Introducción
- 1.2.2 : Sintaxis Razor y scaffolding
- 1.2.3 : Vista layout
- 1.2.4 : Vistas parciales

ACTIVIDADES PROPUESTAS

- Los alumnos desarrollan los laboratorios de esta semana.
- Los alumnos desarrollan las interfaces para una aplicación Web
- Los alumnos crean una aplicación Web en ASP.NET aplicando el patrón MVC.

1.1. ASP.NET MVC

1.1.1. Introducción

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

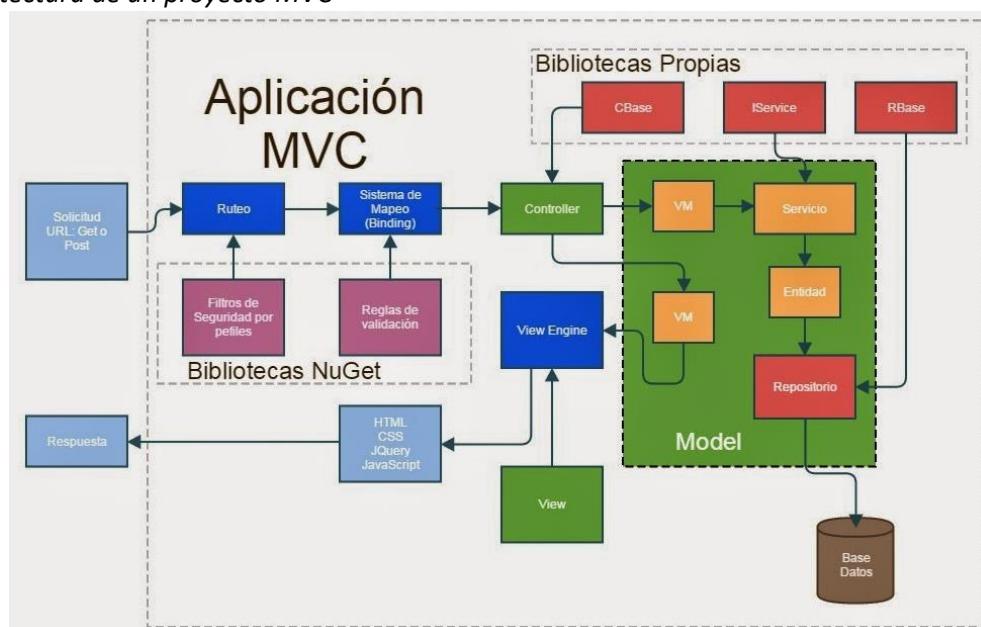
ASP.NET MVC nace como una opción para hacer frente al ya consagrado y alabado Ruby on Rails, un framework que procura hacer uso de buenas prácticas de programación como la integración de Unit tests o la separación clara de ocupaciones, lo que permite brindar casi todos los beneficios otorgados por Ruby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

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

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

La siguiente figura muestra los principales componentes de su arquitectura:

Figura 1
Arquitectura de un proyecto MVC



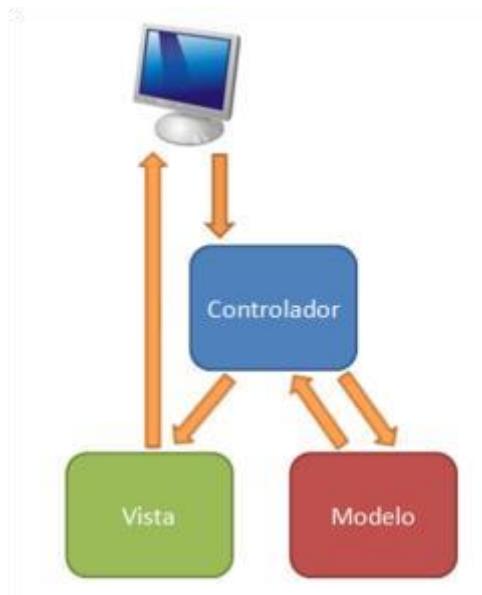
Nota. Tomado de [La arquitectura de mis proyectos MVC](http://msaspnetmvc.blogspot.com/2015/03/la-arquitectura-de-mis-proyectos-mvc.html), por Jose Ojeda, 12 de marzo de 2015, Blogspot.com (<http://msaspnetmvc.blogspot.com/2015/03/la-arquitectura-de-mis-proyectos-mvc.html>)

1.1.2. Plataforma de 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. (Aguilar, s.f.)

Figura 2

Esquema de una petición en ASP.NET MVC



Nota. Tomado de *ASP.NET MVC*, por .Net y Punto, 2011, [punonetypunto.wordpress.com](https://punonetypunto.wordpress.com/2011/06/16/asp-net-mvc-introduccion/) (<https://punonetypunto.wordpress.com/2011/06/16/asp-net-mvc-introduccion/>)

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

- 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 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 estos de manera muy simple.

- **Templates de proyectos móviles:** En el caso de que esté empezando un nuevo proyecto y quiera que el mismo corra exclusivamente en navegadores de dispositivos móviles y tablets, se puede utilizar el Mobile Application Project template, el cual está basado en jQuery Mobile, una librería open source para construir interfaces optimizadas para el uso táctil
- **Modos de visualización:** el nuevo modo de visualización permite a MVC seleccionar el tipo de vista más conveniente dependiendo del navegador que se encuentre realizando el requerimiento. La disposición de las vistas y las vistas parciales pueden ser sobreescritas dependiendo de un tipo de navegador en particular.
- **Soporte para llamados asíncronos basados en tareas:** se puede escribir métodos asíncronos para cualquier controlador como si fueran métodos ordinarios, los cuales retornan un objeto tipo Task o Task<ActionResult>.
- **Unión y minimización:** 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 las 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 y utiliza la sintaxis basada en tilde (~/), así como también, provee soporte para atributos HTML condicionales.

1.1.2.1. Métodos y selectores del Action del Controlador

Los controladores de MVC son responsables de responder a las solicitudes realizadas en un sitio web de ASP.NET MVC. Cada solicitud del explorador se asigna a un controlador determinado. Por ejemplo, escriba la siguiente dirección URL en la barra de direcciones del explorador:
<http://localhost:4576/Inventario/Index/3>

En esta operación, se invoca un controlador llamado **InventarioController**. Este controlador es responsable de generar la respuesta a la solicitud del explorador.

Por ejemplo, el controlador podría devolver una vista determinada al explorador o el controlador podría redirigir al usuario a otro controlador.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Mvc.Ajax;

namespace MvcApplication1.Controllers
```

```
{  
    public class InventarioController : Controller  
    {  
        public ActionResult Index()  
        {  
            return View();  
        }  
    }  
}
```

En este caso, se llama al método Index () en la clase InventarioController. El método Index () es un ejemplo de una acción de controlador.

Una acción de controlador debe ser un método público de una clase de controlador. Tenga en consideración que cualquier método público que agregue a una clase de controlador se expone automáticamente como una acción de controlador. Hay algunos requisitos adicionales que debe cumplir una acción de controlador. Un método utilizado como una acción de controlador no se puede sobrecargar. Además, una acción de controlador no puede ser un método estático. Aparte de eso, puede usar prácticamente cualquier método como una acción de controlador.

Descripción de los resultados de la Acción (Action)

Una acción de controlador devuelve algo llamado resultado de acción. Un resultado de acción es lo que devuelve una acción del controlador en respuesta a una solicitud del explorador.

El marco de MVC de ASP.NET admite varios tipos de resultados de acciones, entre los que se incluyen:

Tabla 1

Tipos de resultados de acciones

Acción	Resultado
ViewResult	Representa HTML y marcado
EmptyResult	No representa ningún resultado
RedirectResult	Representa una redirección a una nueva dirección URL
JsonResult	Representa un resultado de notación de objetos JavaScript que se puede usar en una aplicación AJAX
JavaScriptResult	Representa un script de JavaScript
ContentResult	Representa un resultado de texto
FileContentResult	Representa un archivo descargable (con el contenido binario)
FilePathResult	Representa un archivo descargable (con una ruta de acceso)

Nota. Adaptado de *Información general sobre el controlador de ASP.NET MVC (C#)*, por Microsoft, 23 de marzo de 2024, (<https://learn.microsoft.com/es-es/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs>)

Todos estos resultados de acción heredan de la clase base ActionResult.

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 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:

Figura 3

Métodos de Acción del Controlador

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

Nota. Elaboración propia.

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.

Figura 4

Métodos de Acción del Controlador

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

Nota. Elaboración propia.

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.

1.1.2.2 Verbos: `HttpGet` y `HttpPost`

HTTP es un protocolo de transferencia de hipertexto que está diseñado para enviar y recibir datos entre el cliente y el servidor mediante páginas web. HTTP tiene dos métodos que se utilizan para publicar datos desde páginas web al servidor. Estos dos métodos son `HttpGet` y `HttpPost`.

`HttpGet`

El método `HttpGet` envía datos mediante una cadena de consulta. Los datos se adjuntan a la URL y son visibles para todos los usuarios. Sin embargo, no es seguro, pero es rápido. Se usa principalmente cuando no está publicando datos confidenciales en el servidor como nombre de usuario, contraseña, información de tarjeta de crédito, etc.

Características:

1. El método es rápido, pero no es seguro.
2. Es el método por defecto.
3. Se adjunta los datos del formulario en la cadena de consulta, por lo que los datos son visibles para otros usuarios.
4. Se utiliza la pila para el paso de forma variable.
5. Los datos están limitados a la longitud máxima de la cadena de consulta.
6. Es muy útil cuando no es sensible a los datos.
7. Crea URL que es fácil de leer.
8. Solo puede transportar datos de texto.

Figura 5

`HttpGet`

```

namespace WebApplication01.Controllers
{
    public class HomeController : Controller
    {
        [HttpGet]
        public ActionResult Submit(int id=0, string nombre="")
        {
            ViewBag.id=id;
            ViewBag.nombre=nombre;
            return View();
        }
    }
}

```

Nota. Elaboración propia.

Figura 6
Diseño de vista de *HttpGet*



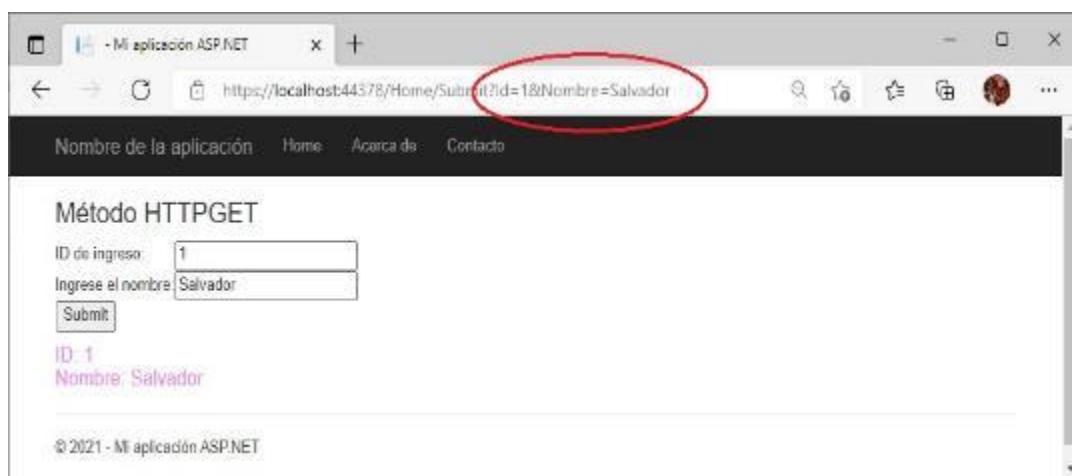
```

1 <h3>Método HTTPGET</h3>
2 <using (Html.BeginForm("Submit", "Home", FormMethod.Get))>
3 {
4     <table>
5         <tr>
6             <td> ID de ingreso: </td>
7             <td> @Html.TextBox("Id") </td>
8         </tr>
9         <tr>
10            <td> Ingrese el nombre: </td>
11            <td> @Html.TextBox("Nombre") </td>
12        </tr>
13        <tr>
14            <td colspan="2"> <input type="submit" value="Submit"> </td>
15        </tr>
16    </table>
17 }
18 <h4 style="color:violet">
19     ID: @ViewBag.id <br />
20     Nombre: @ViewBag.nombre <br />
21 </h4>
22

```

Nota. Elaboración propia.

Figura 7
Vista de *HttpGet*



Nota. Elaboración propia.

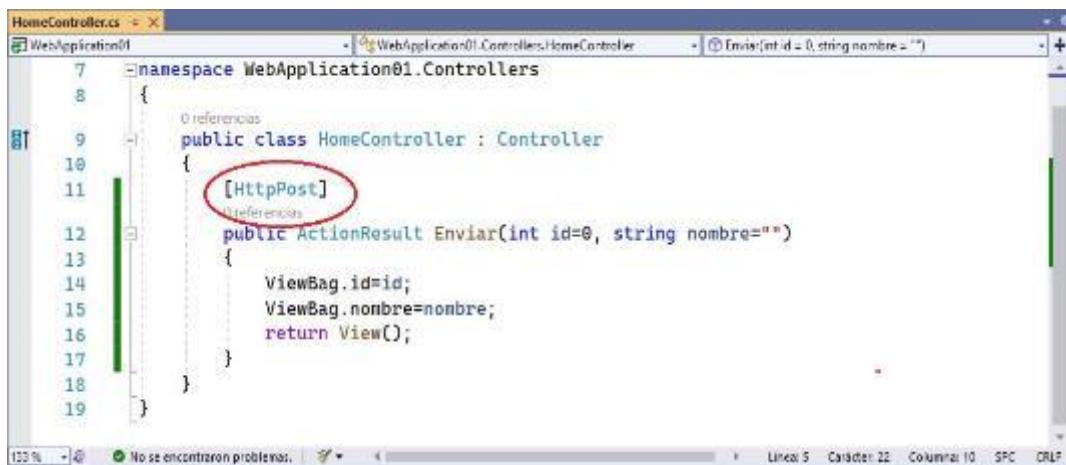
HttpPost

El método *HttpPost* oculta información de la URL y no vincula datos a la URL. Es más seguro que el método *HttpGet*, pero es más lento. Solo es útil cuando está pasando información confidencial al servidor.

Características:

1. Los datos se envían a través del método *HttpPost*, no son visibles para el usuario.
2. Es más segura pero más lento que *HttpGet*.
3. Se utiliza la pila de método para el paso de forma variable.

4. No tiene ninguna restricción para pasar datos y puede publicar variables de formulario ilimitadas.
5. Se utiliza cuando se envían datos críticos.
6. Se puede llevar ambos datos binarios y de texto.

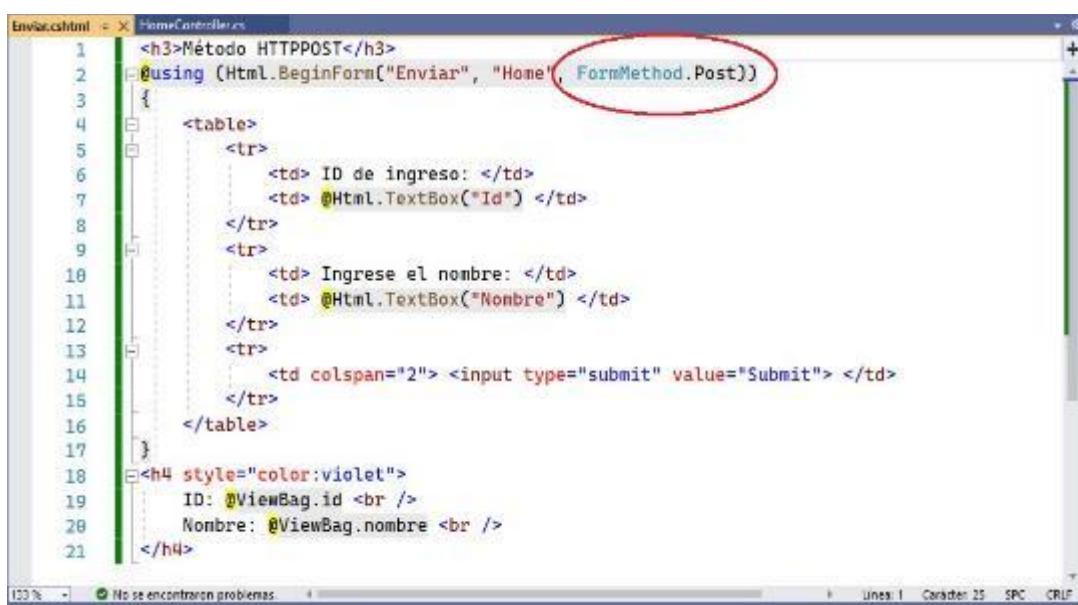
Figura 8*HttpPost*


```

namespace WebApplication01.Controllers
{
    public class HomeController : Controller
    {
        [HttpPost]
        public ActionResult Enviar(int id=0, string nombre="")
        {
            ViewBag.id=id;
            ViewBag.nombre=nombre;
            return View();
        }
    }
}

```

Nota. Elaboración propia.

Figura 9*Diseño de vista de HttpPost*


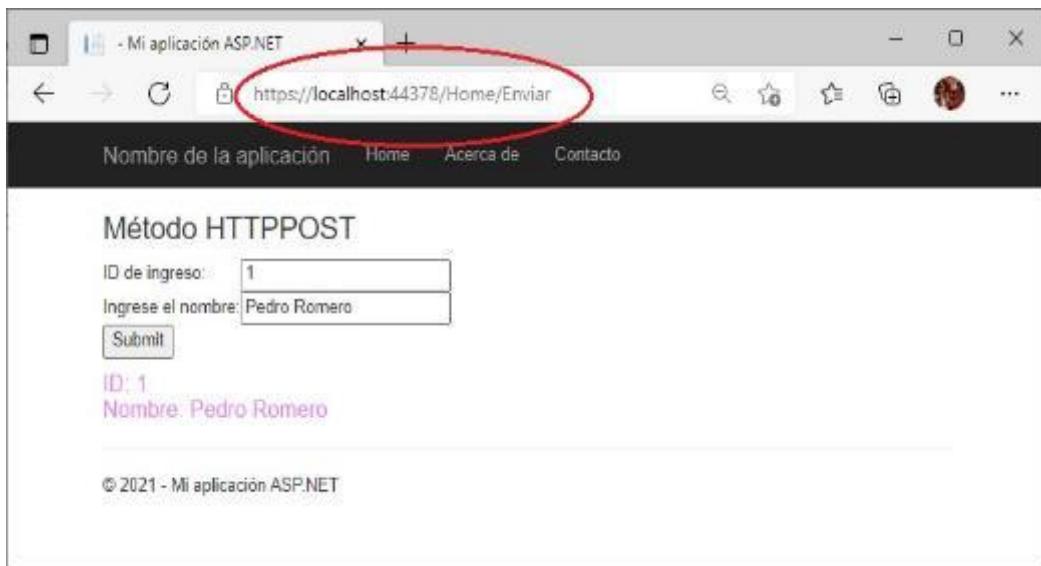
```

<h3>Método HTTPPOST</h3>
@using (Html.BeginForm("Enviar", "Home", FormMethod.Post))
{
    <table>
        <tr>
            <td> ID de ingreso: </td>
            <td> @Html.TextBox("Id") </td>
        </tr>
        <tr>
            <td> Ingrese el nombre: </td>
            <td> @Html.TextBox("Nombre") </td>
        </tr>
        <tr>
            <td colspan="2"> <input type="submit" value="Submit"> </td>
        </tr>
    </table>
}
<h4 style="color:violet">
    ID: @ViewBag.id <br />
    Nombre: @ViewBag.nombre <br />
</h4>

```

Nota. Elaboración propia.

Figura 10
Vista de HttpPost



Nota. Elaboración propia.

1.1.2.3 Transferencia de datos: ViewBag, ViewData y TempData

ViewData, ViewBag y TempData se utilizan para transferir datos y objetos entre Controller a View o de un Controller a otro en ASP.Net MVC.

ViewBag

1. ViewBag es un contenedor construido alrededor de ViewData.
2. ViewBag es una propiedad dinámica y utiliza las características dinámicas de C # 4.0.
3. Durante la recuperación, no es necesario utilizar datos de conversión de tipos.
4. ViewBag se utiliza para pasar valor de Controller a View.
5. ViewBag está disponible solo para Solicitud actual. Se destruirá en la redirección.

ViewData

1. ViewData se deriva de la clase ViewDataDictionary y es básicamente un objeto de diccionario, es decir, claves y valores donde las claves son cadenas, mientras que los valores serán objetos.
2. Los datos se almacenan como Objeto en ViewData.
3. Durante la recuperación, los datos deben ser convertidos en tipo a su tipo original, ya que los datos se almacenan como objetos y también requiere verificaciones NULL durante la recuperación.
4. ViewData se utiliza para pasar valor de Controller a View.
5. ViewData está disponible solo para Solicitud actual. Se destruirá en la redirección.

TempData

1. TempData se deriva de la clase TempDataDictionary y es básicamente un objeto de diccionario, es decir, claves y valores donde las claves son cadenas, mientras que los valores serán objetos.
2. Los datos se almacenan como Objeto en TempData.

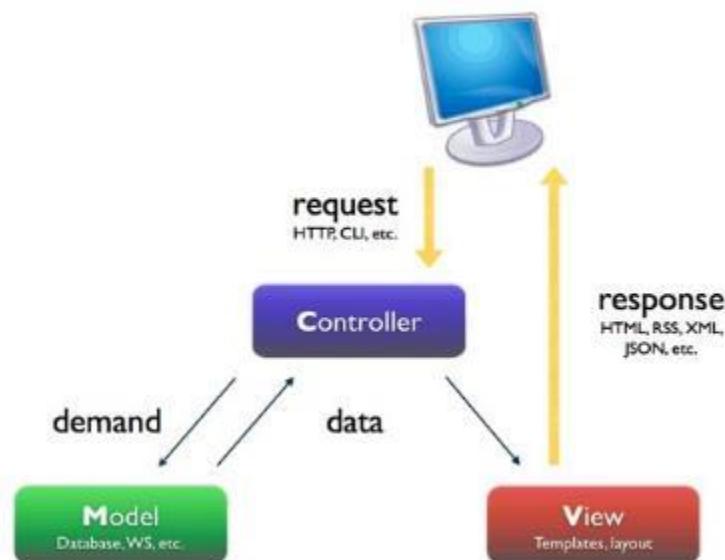
3. Durante la recuperación, los datos deben ser convertidos en tipo a su tipo original, ya que los datos se almacenan como objetos y también requiere verificaciones NULL durante la recuperación.
4. TempData se puede utilizar para pasar valor de Controlador a Vista y también de Controlador a Controlador.
5. TempData está disponible para solicitudes actuales y posteriores. No se destruirá en la redirección.

1.1.3. Estructura del MVC: Modelo, Vista y Controlador

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. (Mozilla Foundation, 2023)

Figura 11
Arquitectura de MVC



Nota. Tomado de [La arquitectura MVC](#), por Uniwebsidad, 2023, uniwebsidad.com

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. (Microsoft, 2022)

El diagrama del patrón MVC funciona de la siguiente manera:

1. El navegador realiza una petición a una determinada URL.
2. ASP.NET MVC recibe la petición y determinar el controlador que debe ejecutarse (ver más adelante como se realiza este proceso).
3. El controlador recibe la petición HTTP.
4. 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.

1.1.4. URL de enrutamiento

El enrutamiento es responsable de asignar las solicitudes entrantes del explorador a determinadas acciones de controlador.

ASP.NET MVC ofrece dos enfoques para el enrutamiento:

1. La tabla de rutas, que es una colección de rutas que se pueden usar para hacer coincidir las solicitudes entrantes con las acciones del controlador.
2. Enrutamiento de atributos, que realiza la misma función, pero se logra mediante la decoración de las propias acciones, en lugar de editar una tabla de rutas global.

Tabla de rutas

La tabla de rutas se configura cuando se inicia la aplicación. Normalmente, se usa una llamada a método estático para configurar la colección de rutas global, como se indica a continuación:

Figura 12

Tabla de rutas

```
public class RouteConfig
{
    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 }
        );
    }
}
```

Nota. Elaboración propia.

La tabla de rutas se administra mediante el tipo, **RouteCollection** usa para agregar nuevas rutas con MapRoute. Las rutas se definen e incluyen una cadena de ruta, que puede incluir parámetros para controladores, acciones, áreas y otros marcadores de posición. Si una aplicación sigue una convención estándar, la mayoría de sus acciones se pueden controlar mediante esta única ruta predeterminada, con cualquier excepción a esta convención configurada mediante rutas adicionales.

Enrutamiento de atributos en ASP.NET MVC

Las rutas que se definen con sus acciones pueden ser más fáciles de detectar y razonar que las rutas definidas en una ubicación externa. Mediante el enrutamiento de atributos, un método de acción individual puede tener su ruta definida con el atributo **[Route]**:

Figura 13
Enrutamiento de atributos

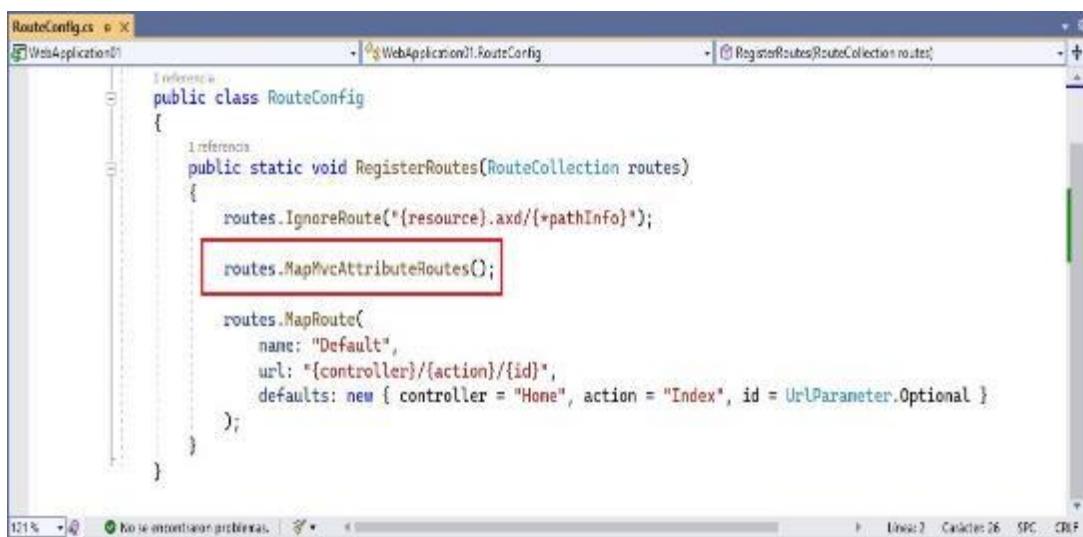
```
public class HomeController : Controller
{
    [Route("productos")]
    public ActionResult Index()
    {
        return View();
    }

    [Route("productos/{id}")]
    public ActionResult Buscar(string id)
    {
        return View();
    }
}
```

Nota. Elaboración propia.

La configuración del enrutamiento de atributos requiere agregar una línea a la configuración predeterminada de la tabla de rutas: **routes.MapMvcAttributeRoutes()**

Figura 14
Enrutamiento de atributos



Nota. Elaboración propia.

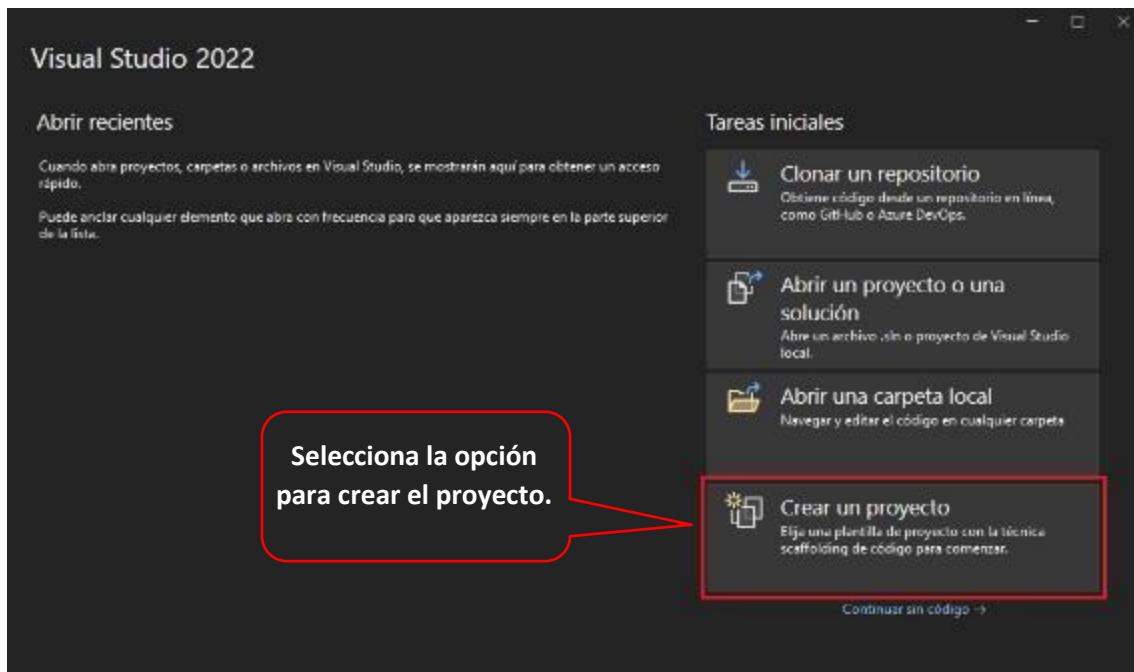
LABORATORIO 1.1.: Creando una aplicación ASP.NET MVC

Implemente un proyecto ASP.NET MVC, el cual permita crear una página de inicio del sitio Web.

Inicio del Proyecto

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

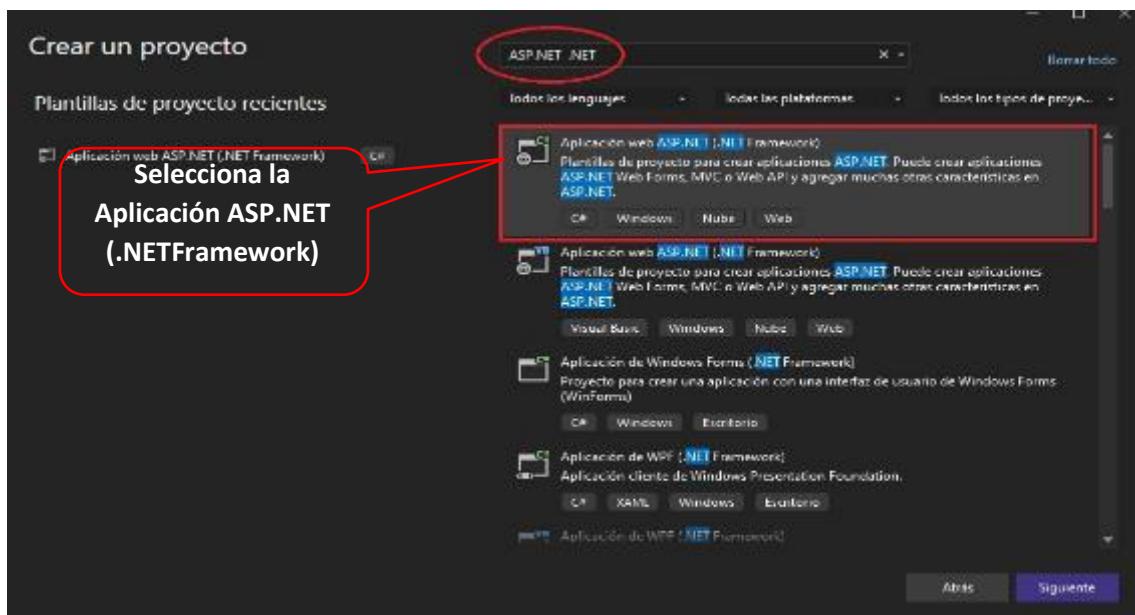
Figura 15
Desarrollo Práctico



Nota. Elaboración propia.

Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción Siguiente.

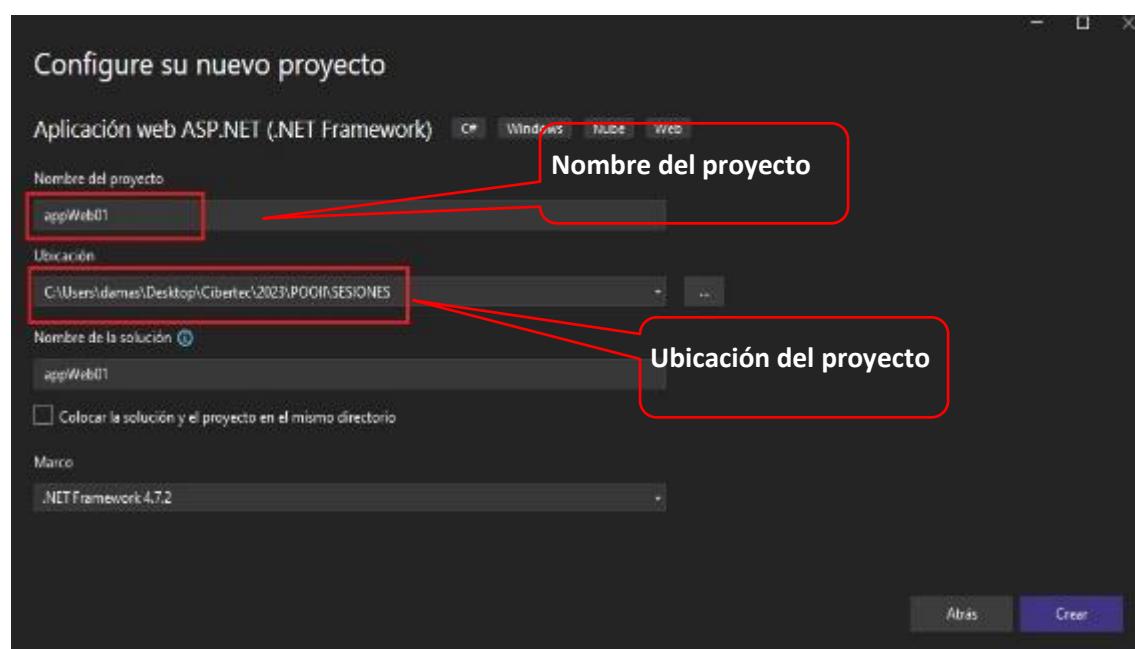
Figura 16
Desarrollo Práctico



Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciona la ubicación del mismo, al terminar presiona la opción **Crear**.

Figura 17
Desarrollo Práctico



Nota. Elaboración propia.

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

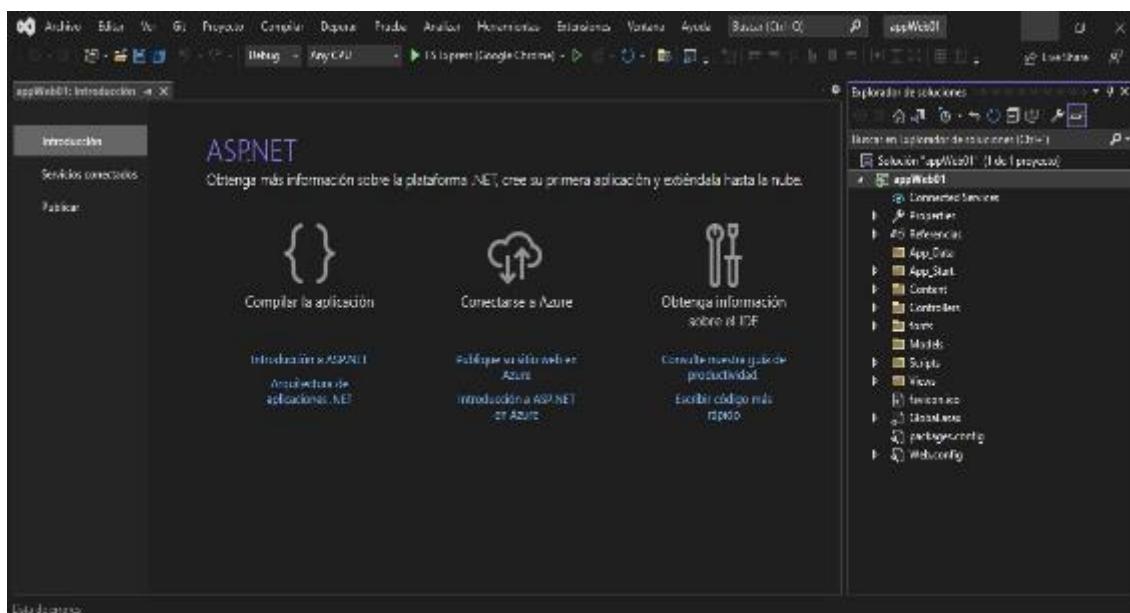
Figura 18
Desarrollo Práctico



Nota. Elaboración propia.

IDE del proyecto Web

Figura 19
Desarrollo Práctico



Nota. Elaboración propia.

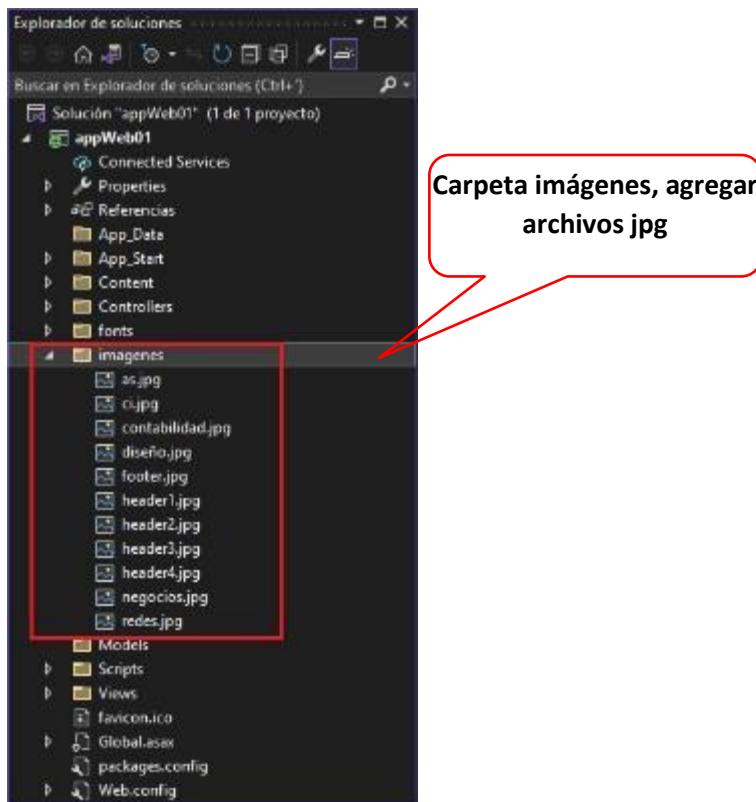
Desarrollando la página de Inicio (Home) de la aplicación

Agregar la carpeta imágenes

En el explorador de soluciones, agregue una **carpeta Nueva**, llamada imágenes. En dicha carpeta agregue los archivos de imágenes de extensión .jpg, descargar de Internet.

Figura 20

Desarrollo Práctico

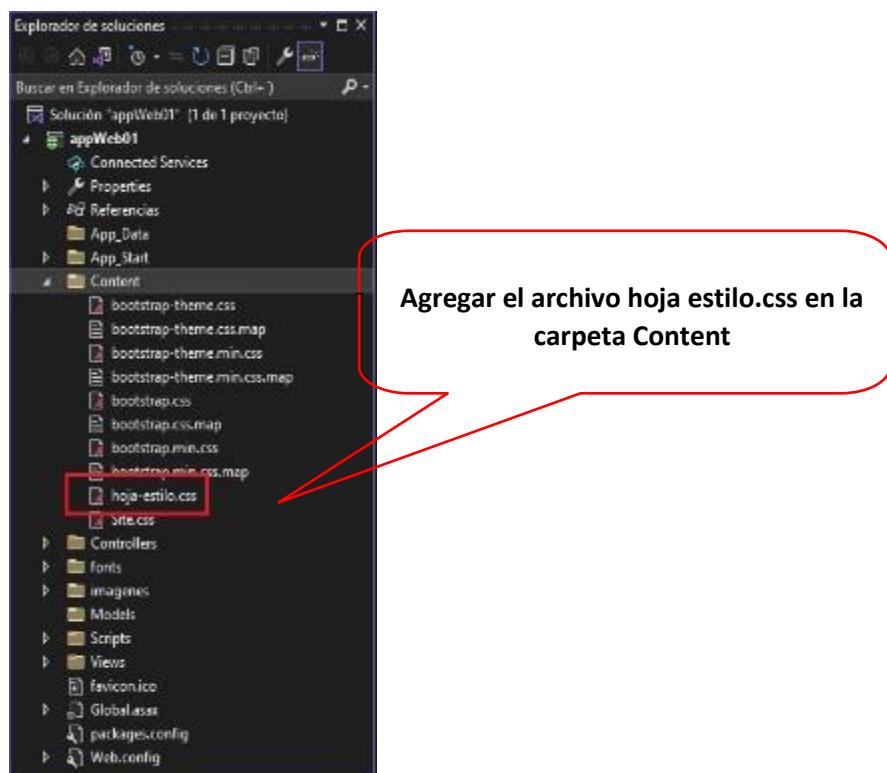


Nota. Elaboración propia.

Agregue un Hoja de Estilo CSS en la aplicación

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

Figura 21
Desarrollo Práctico



Nota. Elaboración propia.

A continuación, defina estilos a las etiquetas de clase que utilizarás en el diseño de la vista Home.

Figura 22
Desarrollo Práctico

```

header,nav,footer{ width:100%; height:auto; }

.logo{ width:100%; height:350px; border-radius:20px; }

.logo-footer{width:100%; height:120px; }

.menu-date { display: grid; grid-template-rows: repeat(2,1fr); column-gap: 5px; }

.menu{display:grid; grid-template-columns:repeat(6,1fr); column-gap:5px; }

.menu-item{text-align:center; padding-top:5px; padding-bottom:5px; display:block; font-size:18px; }

.menu-item:hover{background-color:yellow; color:blue; }

.wraper { display: grid; grid-template-rows: repeat(3,1fr); column-gap: 10px; row-gap:10px }

.wraper-item{text-align:center; border:1px solid; border-radius:10px}

.wraper-item img{ width:90%; height:220px; padding:5%}

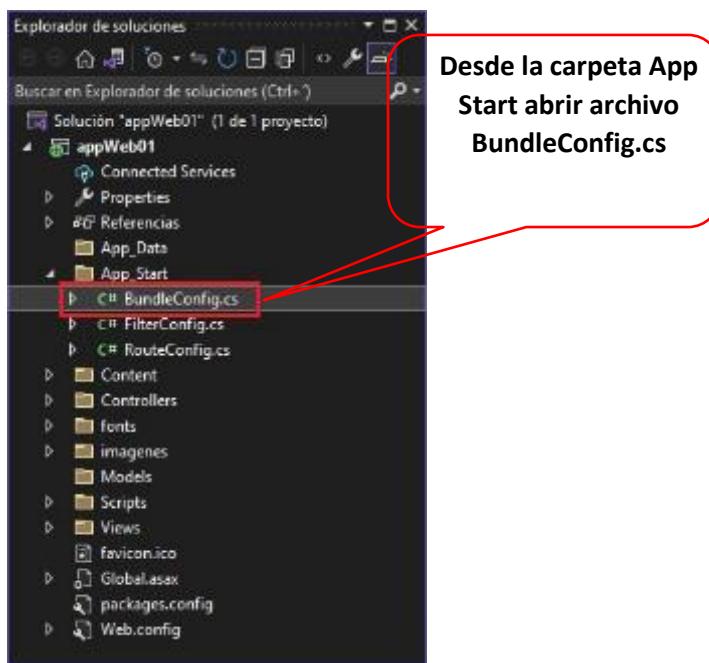
```

Nota. Elaboración propia.

Registrando el estilo en el BundleConfig.cs

Desde la carpeta App_Start abrir el archivo BundleConfig.cs, tal como se muestra.

Figura 23
Desarrollo Práctico



Nota. Elaboración propia.

En el archivo BundleConfig.cs, agregar en bundles.Add la dirección del archivo hoja-estilo.css para registrarlo.

Figura 24
Desarrollo Práctico

```

1 referencia
public class BundleConfig
{
    1 referencia
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                    "~/Scripts/jquery-{version}.js"));

        bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
                    "~/Scripts/jquery.validate*"));

        bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
                    "~/Scripts/modernizr-*"));

        bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
                    "~/Scripts/bootstrap.js"));

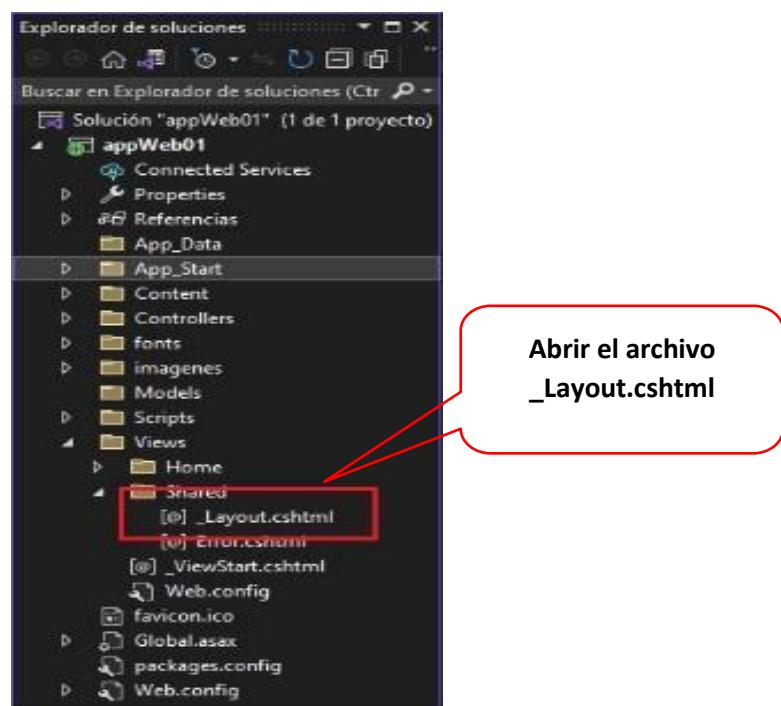
        bundles.Add(new StyleBundle("~/Content/css").Include(
                    "~/Content/bootstrap.css",
                    "~/Content/site.css",
                    "~/Content/hoja-estilo.css"));
    }
}

```

Nota. Elaboración propia.

En la carpeta Shared almacenada en la carpeta View, abrir el archivo _Layout.cshtml

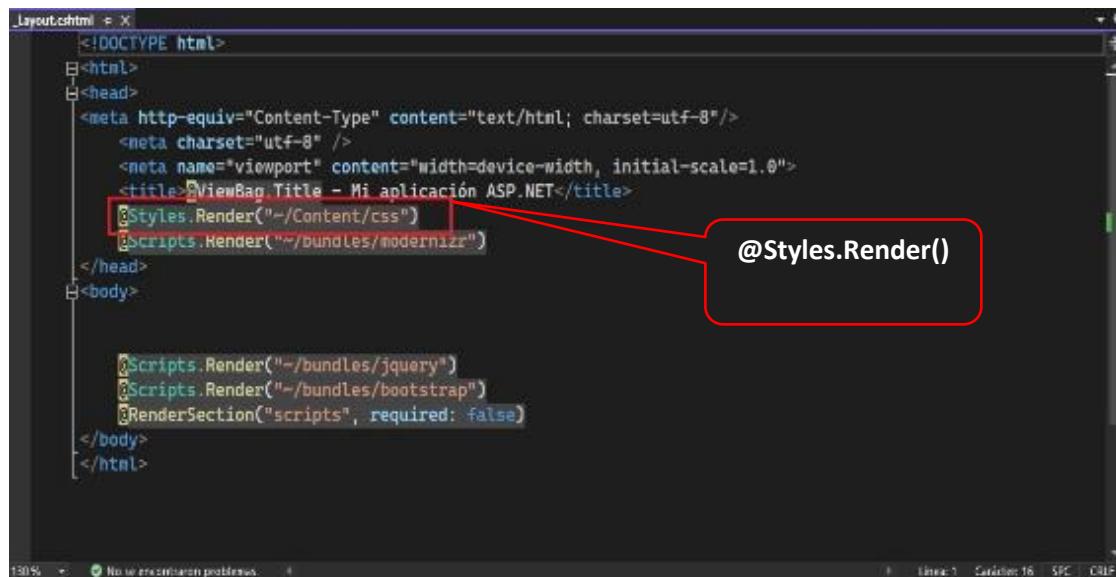
Figura 25
Desarrollo Práctico



Nota. Elaboración propia.

Al abrir el archivo encontramos `@Style.Render("~/Content/css")` el cual recupera los archivos css registrados en el bundles.

Figura 26
Desarrollo Práctico



```
<!DOCTYPE html>

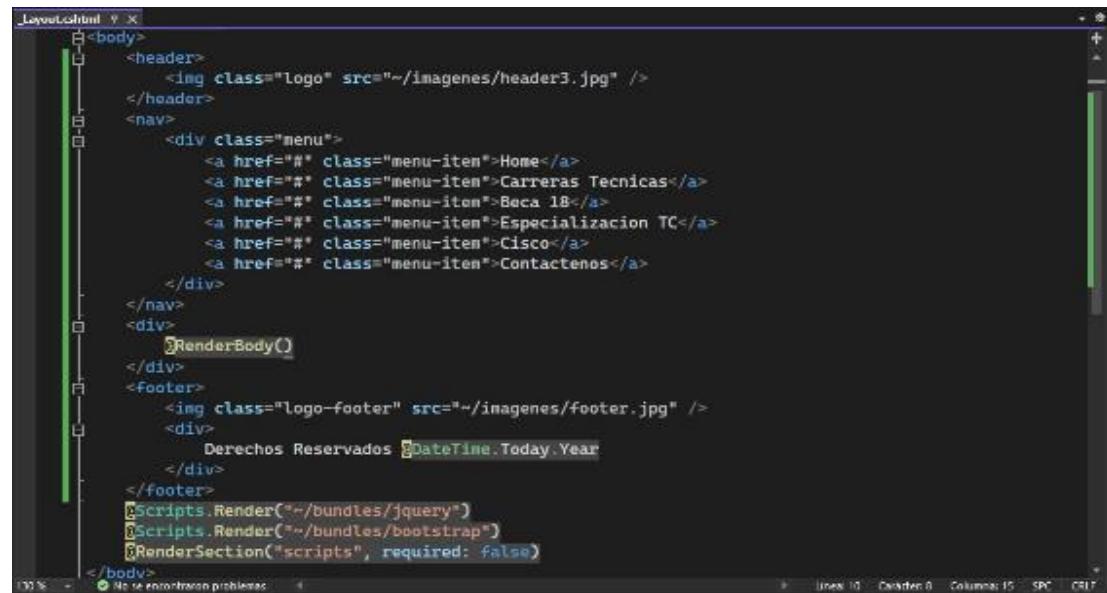
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>@ViewBag.Title - Mi aplicación ASP.NET</title>
        @Styles.Render("~/Content/css")
        @Scripts.Render("~/bundles/modernizr")
    </head>
    <body>

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

Nota. Elaboración propia.

En el body del _Layout, agrega los bloques y elementos la cual se visualizará en todas las páginas que referencien a la página maestra (_ViewStart). Guarde los cambios efectuados en el archivo.

Figura 27
Desarrollo Práctico



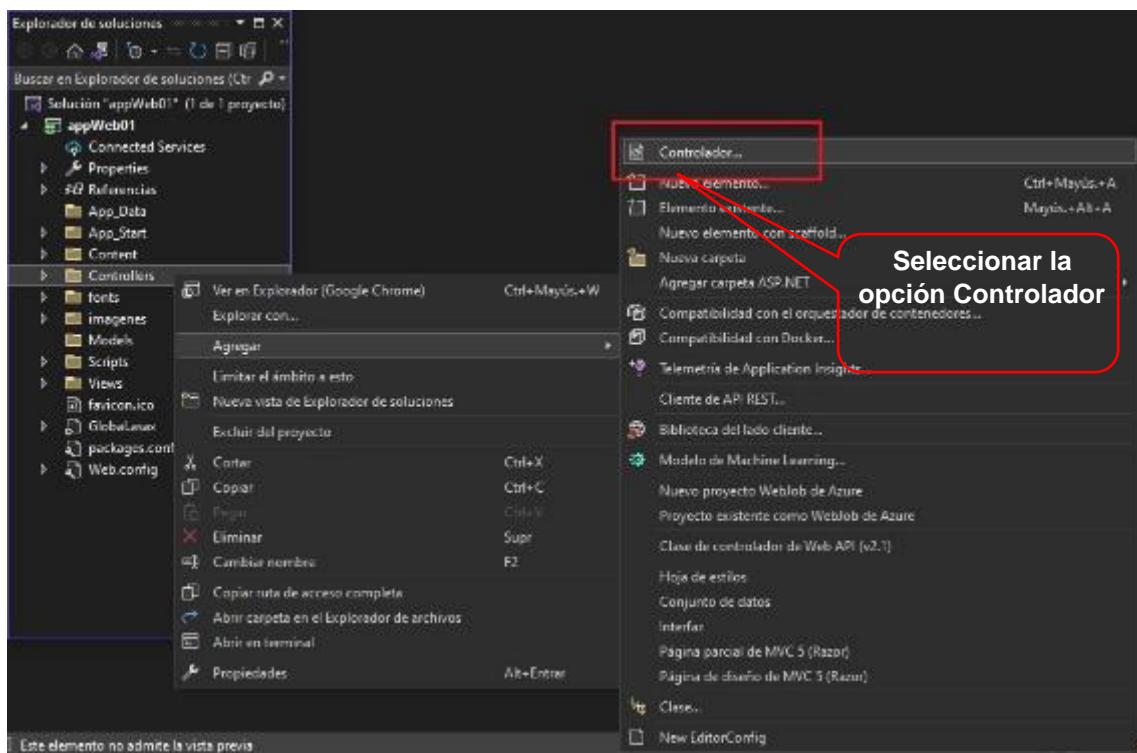
```
<body>
    <header>
        
    </header>
    <nav>
        <div class="menu">
            <a href="#" class="menu-item">Home</a>
            <a href="#" class="menu-item">Carreras Técnicas</a>
            <a href="#" class="menu-item">Beca 18</a>
            <a href="#" class="menu-item">Especialización TC</a>
            <a href="#" class="menu-item">Cisco</a>
            <a href="#" class="menu-item">Contactenos</a>
        </div>
    </nav>
    <div>
        @RenderBody()
    </div>
    <footer>
        
        <div>
            Derechos Reservados @DateTime.Today.Year
        </div>
    </footer>
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
```

Nota. Elaboración propia.

4. Trabajando con el Controlador

En la carpeta Controllers, seleccione la opción Agregar → Controlador...

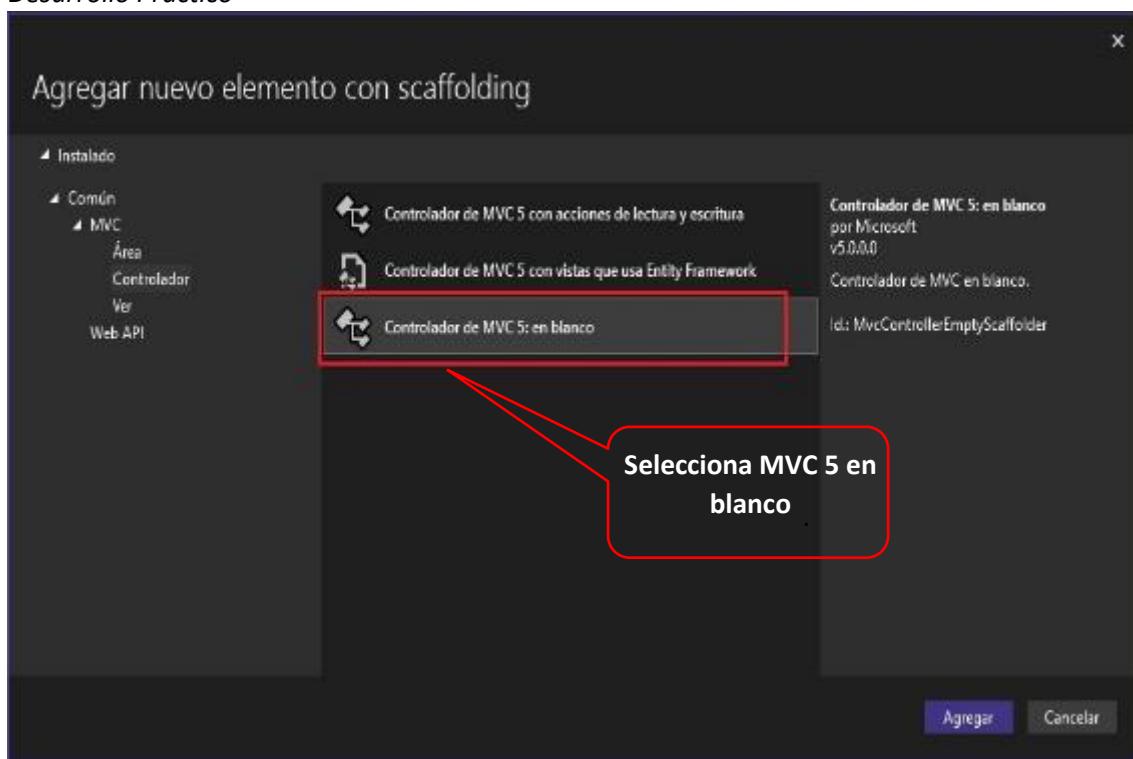
Figura 28
Desarrollo Práctico



Nota. Elaboración propia.

En la ventana Scaffolding, seleccione el tipo de controlador. Para la aplicación seleccione el controlador en blanco, tal como se muestra. A continuación, presione el botón Agregar.

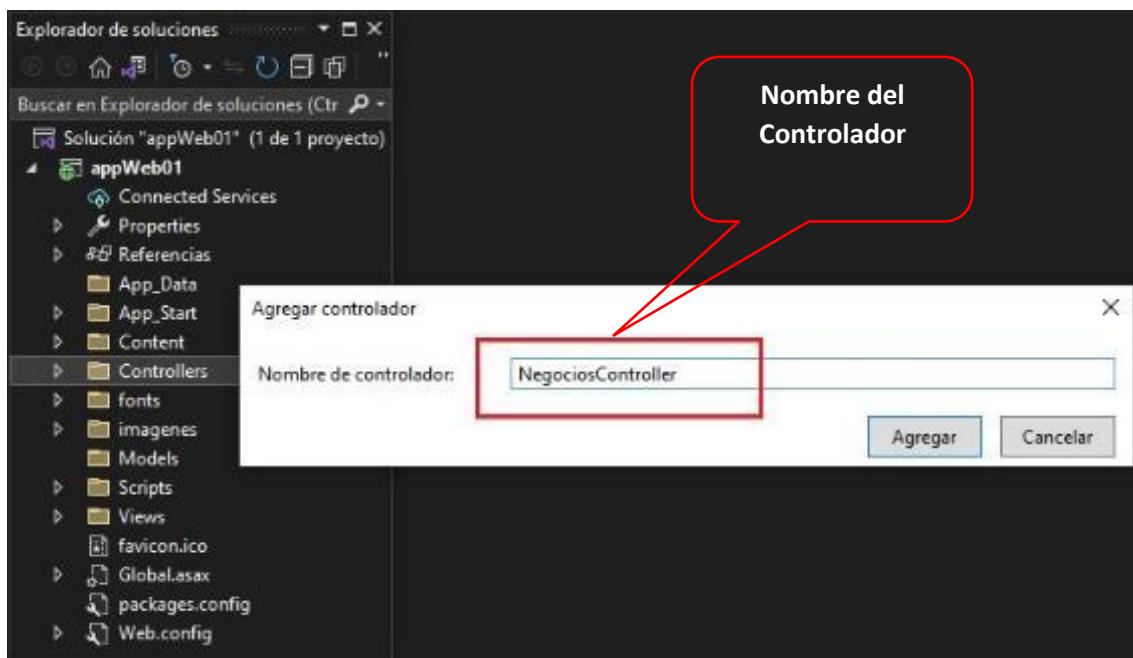
Figura 29
Desarrollo Práctico



Nota. Elaboración propia.

Asigne al nombre al Controlador, tal como se muestra. Presiona el botón AGREGAR.

Figura 30
Desarrollo Práctico



Nota. Elaboración propia.

En el controlador se encuentra definido el ActionResult Index(), tal como se muestra. A continuación, agregamos su Vista.

Figura 31
Desarrollo Práctico

```

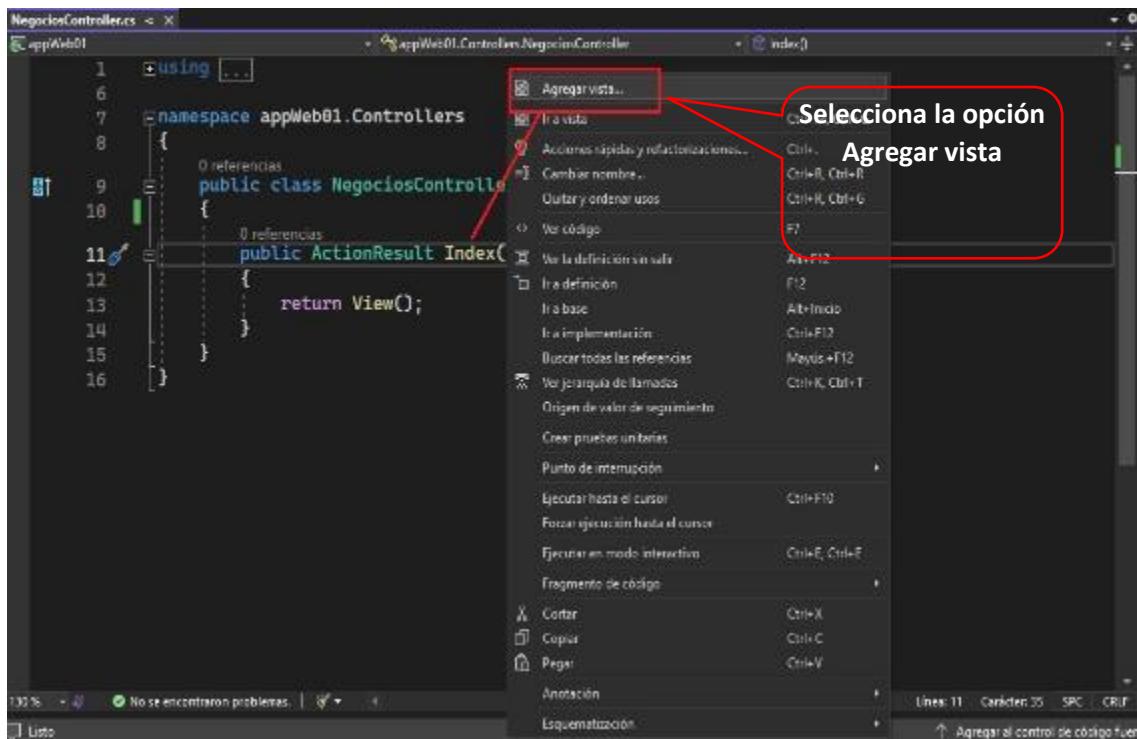
NegociosController.cs < x
appWeb01.Controllers.NegociosController > Index()
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6
7  namespace appWeb01.Controllers
8  {
9      public class NegociosController : Controller
10     {
11         public ActionResult Index()
12         {
13             return View();
14         }
15     }
16 }

```

Nota. Elaboración propia.

Para agregar la Vista del ActionResult, hacer clic derecho en el nombre del ActionResult, selecciona la opción Agregar vista tal como se muestra.

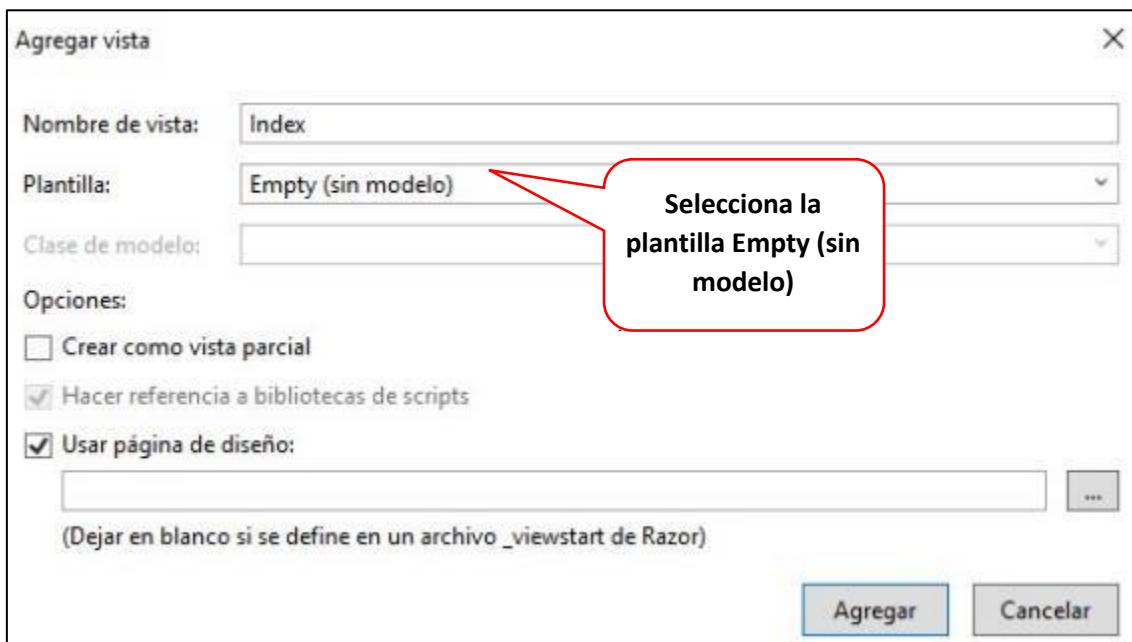
Figura 32
Desarrollo Práctico



Nota. Elaboración propia.

En el diseño de la Vista, el nombre es Index, la plantilla es vacío: Empty (sin modelo), presiona la opción AGREGAR tal como se muestra.

Figura 33
Desarrollo Práctico



Nota. Elaboración propia.

En la vista, diseña el contenido de la página para visualizar las diferentes carreras de Cibertec, utilizamos la clase wrapper y wrapper-item.

Figura 34
Desarrollo Práctico

```

Index.cshtml
ViewBag.Title = "Index"; }

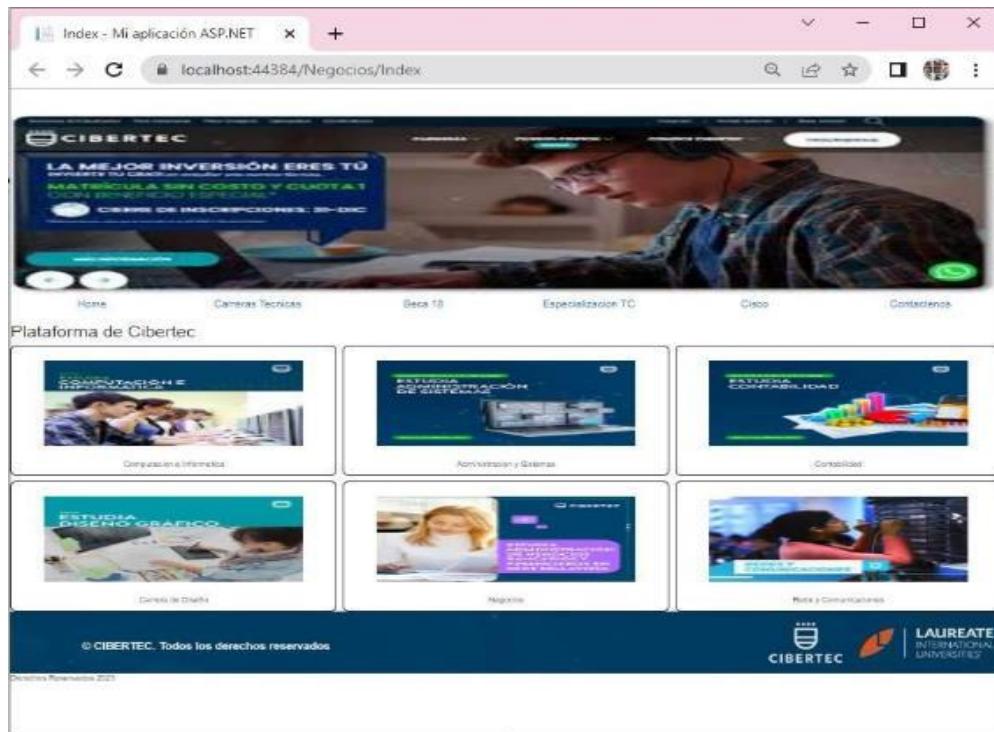
<h2>Plataforma de Cibertec</h2>
<div class="wraper">
    <article class="wraper-item">
        
        <p>Computación e Informática</p>
    </article>
    <article class="wraper-item">
        
        <p>Administración y Sistemas</p>
    </article>
    <article class="wraper-item">
        
        <p>Contabilidad</p>
    </article>
    <article class="wraper-item">
        
        <p>Carrera de Diseño</p>
    </article>
    <article class="wraper-item">
        
        <p>Negocios</p>
    </article>
    <article class="wraper-item">
        
        <p>Rede y Comunicaciones</p>
    </article>
</div>

```

Nota. Elaboración propia.

Para ejecutar la vista Index, presiona F5, donde se compila el proyecto y ejecuta la vista.

Figura 35
Desarrollo Práctico



Nota. Elaboración propia.

Resumen

1. 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.
2. Entre las características más destacables de ASP.NET MVC, se tienen 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
3. 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. (Microsoft, 2022)
4. 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.
5. 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.
6. Entre las características de este patrón, se tienen:
 - 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 Inversión 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.

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/tutorials/how-to-display-command-line-arguments>
- <http://msaspnetmvc.blogspot.com/2015/03/la-arquitectura-de-mis-proyectos-mvc.html>
- https://www-aspsnippets-com.translate.goog/Articles/1621/ASPNet-MVC-ViewData-ViewBag-and-TempData-Similarities-and-Differences-with-examples/?x_tr_sl=en&x_tr_tl=es&x_tr_hl=es&x_tr_pto=sc

1.2. MANEJO DE VISTAS

1.2.1 Introducción

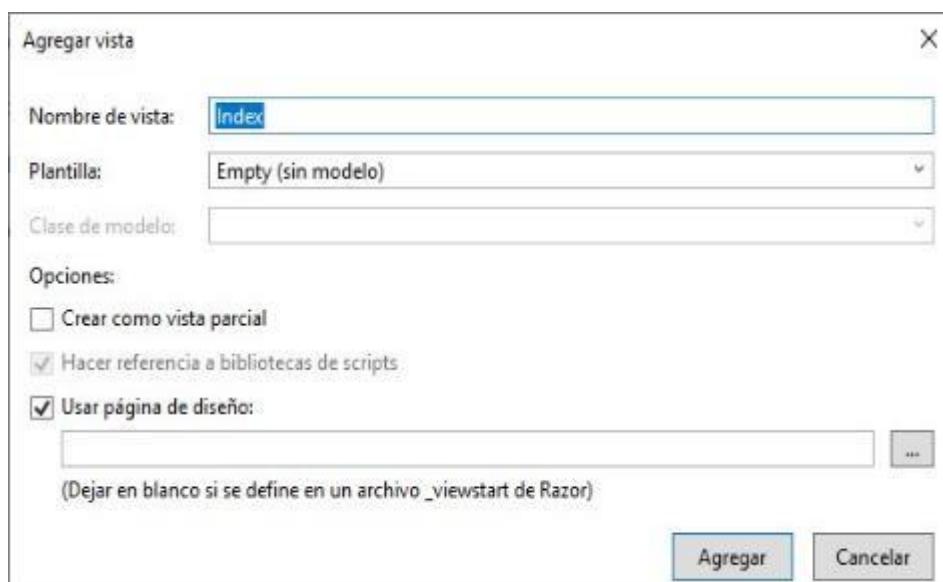
“En el modelo Model-View-Controller (MVC), las vistas están pensadas exclusivamente para encapsular la lógica de presentación. No deben contener lógica de aplicación ni código de recuperación de base de datos. El controlador debe administrar toda la lógica de aplicación” (Ingeniería Systems, s.f.). Una 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 seleccione cuando la vista va a estar relacionada a un Modelo y este objeto debe ser un parámetro de la acción en el controlador.
- Creación de Vistas Parciales: Cuando es necesario reutilizar código HTML en diferentes partes del proyecto, se crea una vista de este tipo. Por ejemplo, el menú debe estar presente en gran parte de la aplicación, esta vista sería parcial y solo se crearía una sola vez. Para crear una vista parcial, se debe nombrar de la siguiente forma: _NombreVistaParcial, al nombre se le debe anteponer el símbolo “_”. Ejemplo _LoginPartial.cshtml ubicado en la carpeta /Views/Shared.
- Usar como Plantilla o Pagina Maestra: Es una vista genérica de toda la aplicación, es la que contendrá el llamado a los archivos JS y CSS. Las vistas de este tipo deben cumplir con la misma regla para llamar el archivo, al nombre se le debe anteponer el símbolo “_”. El llamado dinámico de las vistas por el Controlador se realiza por medio de la función RenderBody()

Figura 36

Diseño de Vista – Scaffolding



Nota. Elaboración propia.

1.2.2. Sintaxis Razor y Scaffolding

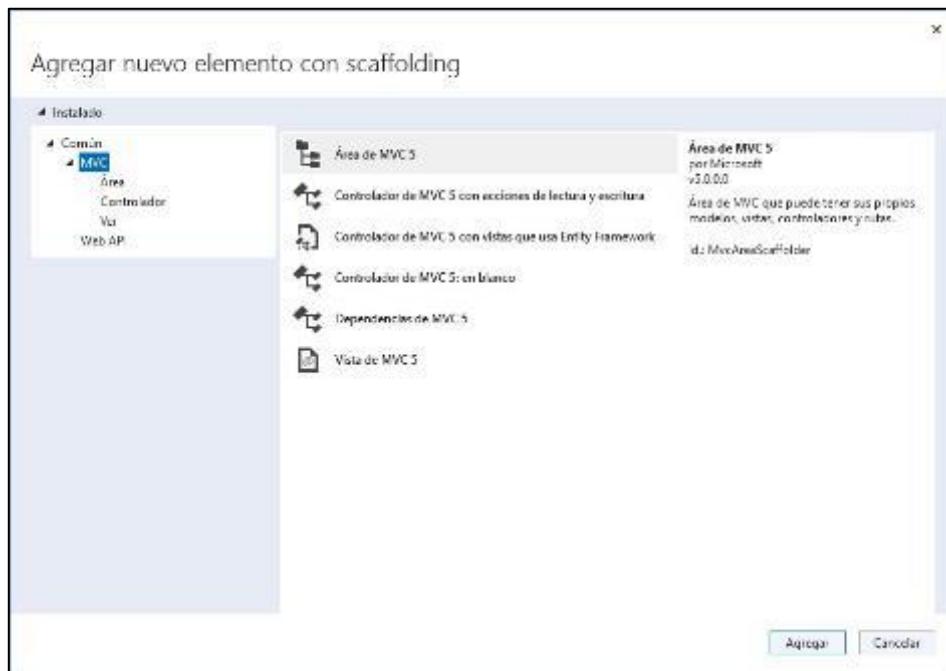
Scaffolding implica la creación de plantillas a través de los elementos del proyecto a través de un método automatizado.

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 seleccione la plantilla scaffold y el modelo de datos para generar los métodos de acción que se implementarán en el controlador.

Figura 37
Diseño de Controladores Scaffolding



Nota. Elaboración propia.

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 item.Nombre.

@{ código }: Permite ejecutar un código que no genera salida HTML.

Scaffolding

ASP.NET Scaffolding de MVC es un marco de generación de código para aplicaciones web, incluye generadores de código preinstalados para proyectos MVC y Web API. Agrega scaffolding a su proyecto cuando desea agregar rápidamente código que interactúe con los modelos de datos. El uso de scaffolding puede reducir la cantidad de tiempo para desarrollar operaciones de datos estándar en su proyecto.

Básicamente, es un marco de generación de código automatizado, genera código para operaciones CRUD en función de las clases de modelo de dominio proporcionadas y las conexiones de base de datos. Puede agregar scaffolding a su proyecto cuando desee agregar código que interactúe con el modelo de datos en el menor tiempo posible.

1.2.3. Vista Layout

Suponga que está desarrollando una aplicación web ASP.NET y desea mantener un aspecto coherente en todas las páginas dentro de su aplicación web. Luego tiene dos opciones, la primera es diseñar las secciones de encabezado, cuerpo y pie de página en cada página, en este enfoque, debe escribir más código en cada página.

La segunda opción es el diseño de "Páginas maestras" que ayudan a definir una plantilla de sitio común y luego heredar su apariencia en todas las vistas de su aplicación web.

En este diseño, utilizamos HTML y algunos otros métodos definidos por el sistema, así que veamos estos métodos uno por uno:

Url.Content (): es un método de la clase UrlHelper. Convierte una ruta virtual (relativa) en una ruta absoluta de la aplicación. Tiene un parámetro de tipo de cadena que es una ruta virtual del contenido. Devuelve la ruta absoluta de una aplicación. Si la ruta de contenido especificada (parámetro del método) no comienza con el carácter tilde (~), este método devuelve contentPath sin cambios. Url.Content () asegura que todos los enlaces funcionen sin importar si el sitio está en un directorio virtual o en la raíz del sitio web.

Html.ActionLink (): la forma más fácil de representar un enlace HTML es utilizar el HTML.ActionLink (). Crea un enlace a una acción del controlador. ActionLink () es un método de extensión de la clase HtmlHelper. Devuelve un elemento de anclaje (un elemento) que contiene la ruta virtual de la acción especificada. Cuando utiliza un método ActionLink (), debe pasar tres parámetros de cadena. Los parámetros son linkText (el texto interno del elemento de anclaje), actionName (el nombre de la acción) y controllerName (el nombre del controlador).

RenderSection (): especifica el nombre de la sección que queremos representar en esa ubicación en la plantilla de diseño. Si una sección es "requerida", Razor arrojará un error en tiempo de ejecución si esa sección no se implementa dentro de una plantilla de vista que se basa en el archivo de diseño (que puede hacer que sea más fácil rastrear errores de contenido). Devuelve el contenido HTML para renderizar.

RenderBody (): en las páginas de diseño, representa la parte de una página de contenido que no está dentro de una sección con nombre. Devuelve el contenido HTML para renderizar. Se requiere RenderBody ya que es lo que representa cada vista.

El archivo _ViewStart

El archivo "_ViewStart" en la carpeta Vistas contiene el siguiente contenido:

Figura 38

archivo _ViewStart



```
_ViewStart.cshtml
1 @{
2     Layout = "~/Views/Shared/_Layout.cshtml";
3 }
```

Nota. Elaboración propia.

Este código se agrega automáticamente a todas las vistas que muestra la aplicación. Si elimina este archivo, debe agregar esta línea a todas las vistas.

1.2.4. Vistas parciales

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

Creando una vista parcial

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

En la siguiente figura, se crea una vista parcial.

Figura 39
Diseño de una Vista Parcial



Nota. Elaboración propia.

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

Para invocar a la vista parcial con los distintos datos:

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

Para invocar a una vista parcial, la puede 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.

Figura 40
Invoker una Vista Parcial

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

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

Nota. Elaboración propia.

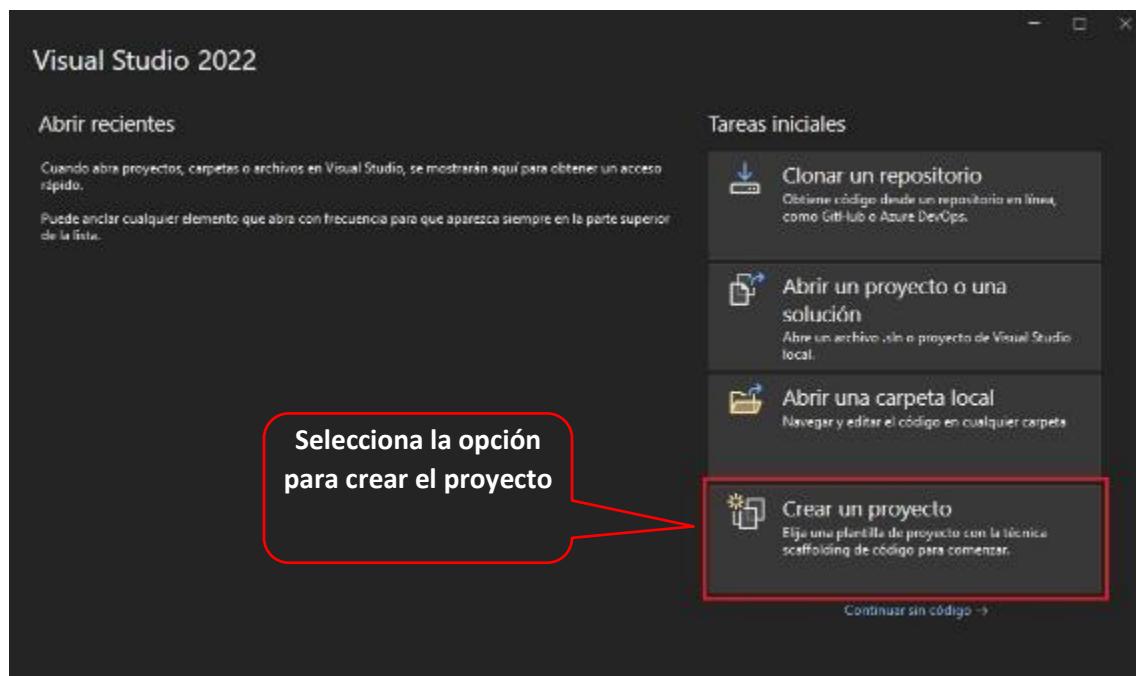
LABORATORIO 2.1.: Trabajando con Vistas Parciales

Se pide implementar una solución, donde permita listar los registros de los clientes registrados desde una fuente de datos (List) y filtrar los registros buscando por las iniciales del campo nombre del cliente.

Inicio del Proyecto

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

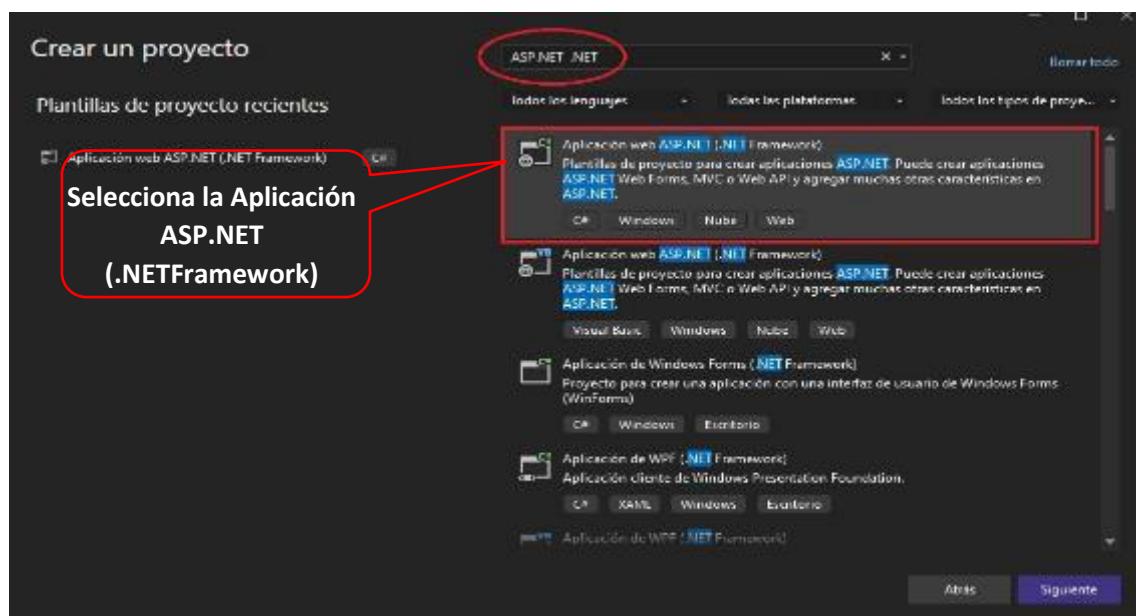
Figura 41
Desarrollo práctico



Nota. Elaboración propia.

Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción Siguiente.

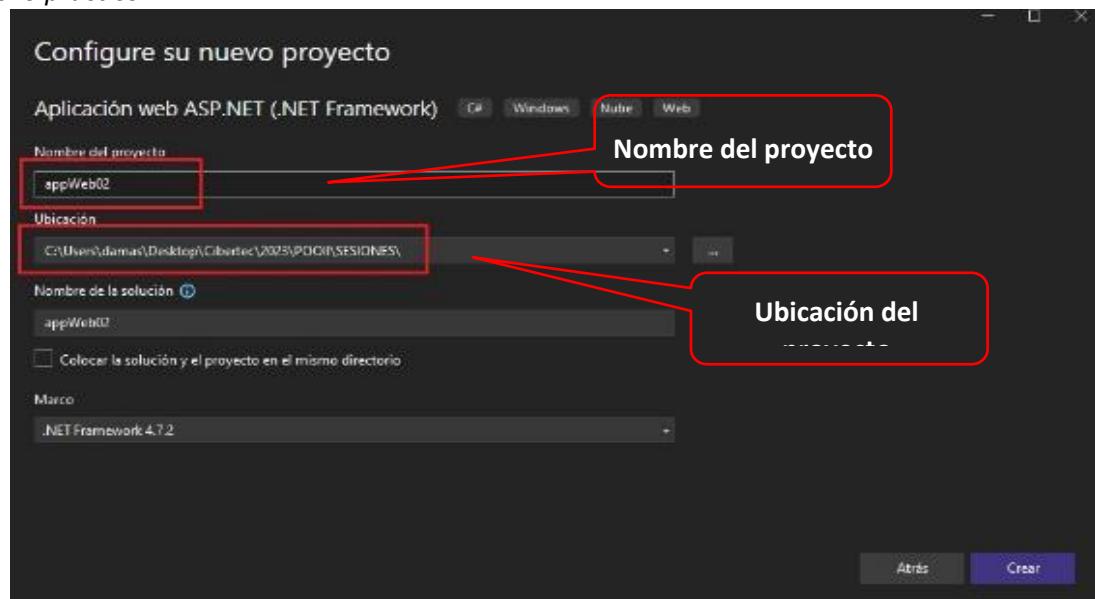
Figura 42
Desarrollo práctico



Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciónala ubicación del mismo, al terminar presiona la opción **Crear**.

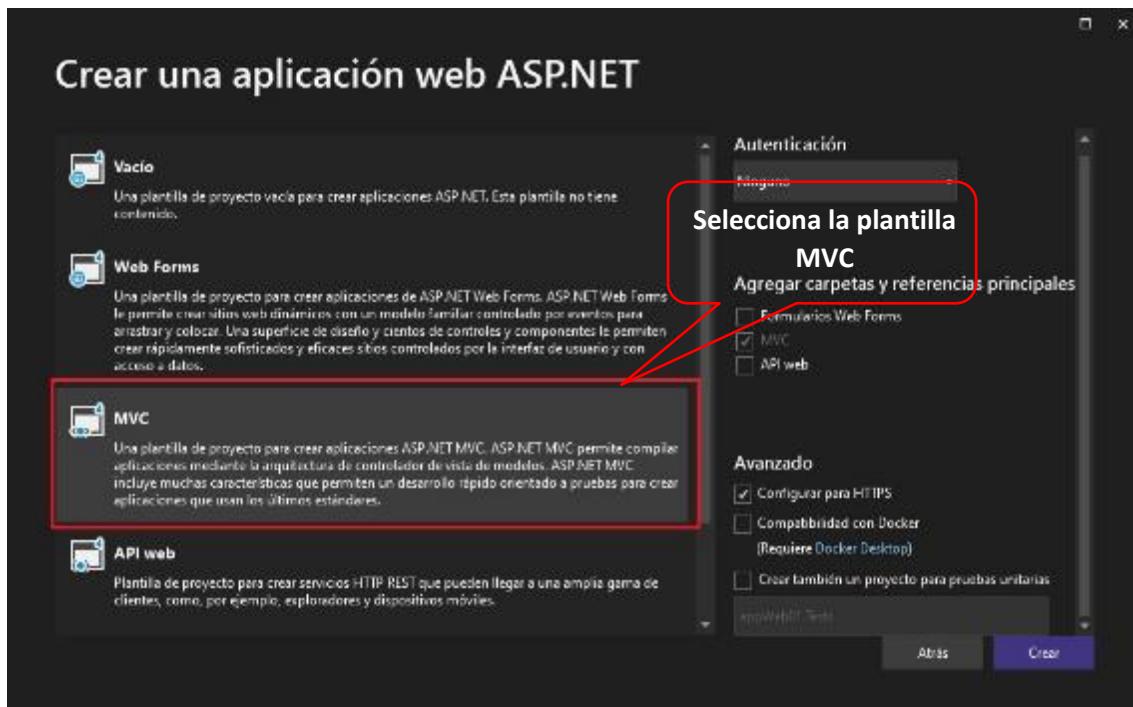
Figura 43
Desarrollo práctico



Nota. Elaboración propia.

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

Figura 44
Desarrollo práctico

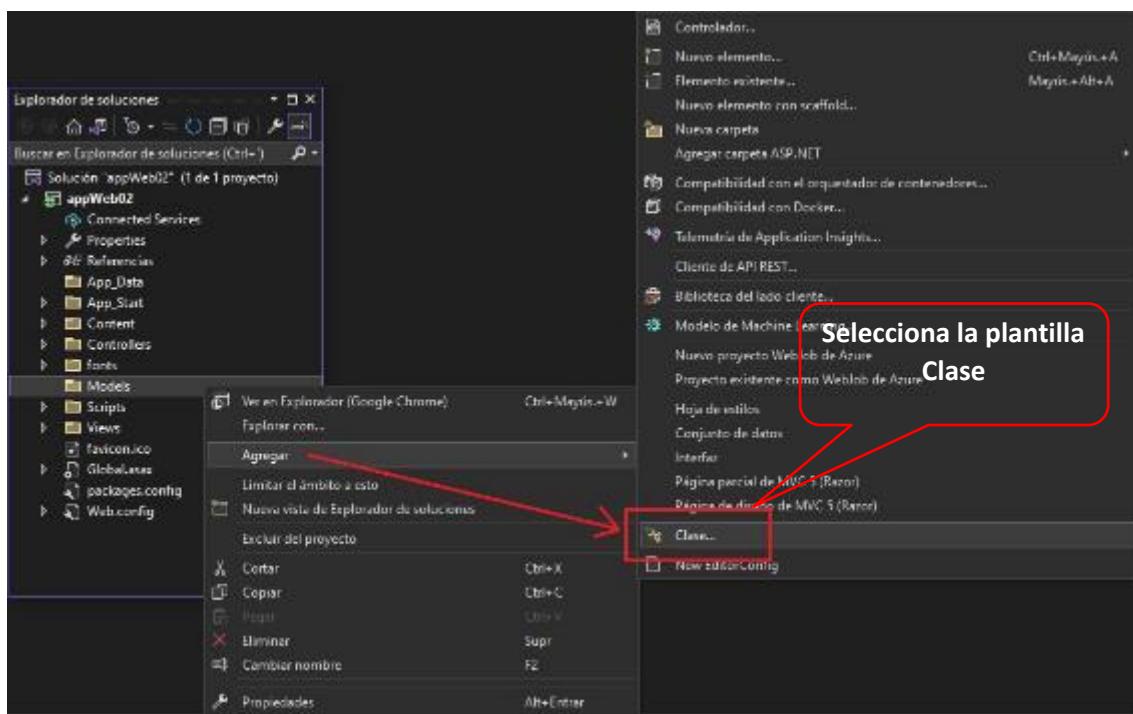


Nota. Elaboración propia.

Agregando una clase al proyecto

En la carpeta Models, hacer clic derecho seleccionar Agregar → Clase, tal como se muestra.

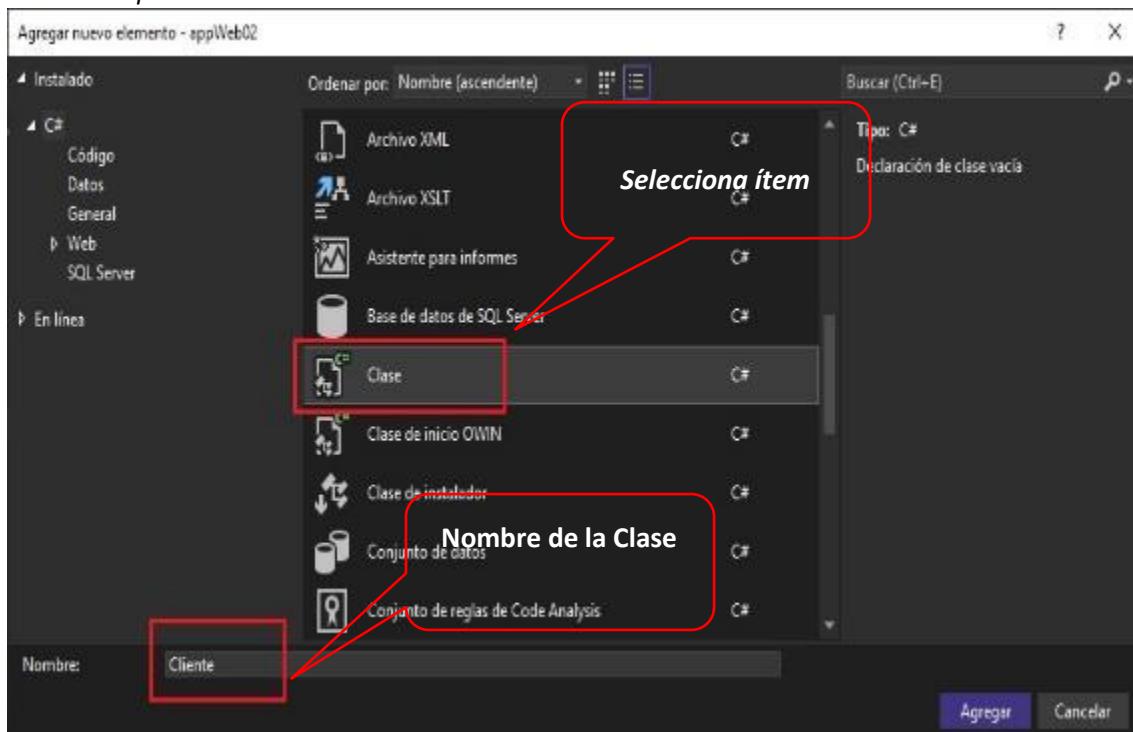
Figura 45
Desarrollo práctico



Nota. Elaboración propia.

Selecciona el elemento Clase, asigne el nombre del elemento: Cliente, tal como se muestra.

Figura 46
Desarrollo práctico



Nota. Elaboración propia.

Agregar la librería System.ComponentModel.DataAnnotations y los atributos de la clase Cliente, tal como se muestra.

Figura 47
Desarrollo práctico

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.ComponentModel.DataAnnotations;
6  namespace appWeb02.Models
7  {
8      public class Cliente
9      {
10         [Display(Name = "Id Cliente")]
11         public string idcliente { get; set; }
12         [Display(Name = "Nombre Cliente")]
13         public string nombre { get; set; }
14         [Display(Name = "Direccion")]
15         public string direccion { get; set; }
16         [Display(Name = "Email Cliente")]
17         public string email { get; set; }
18         [Display(Name = "Telefono Cliente")]
19         public string telefono { get; set; }
20     }
21 }

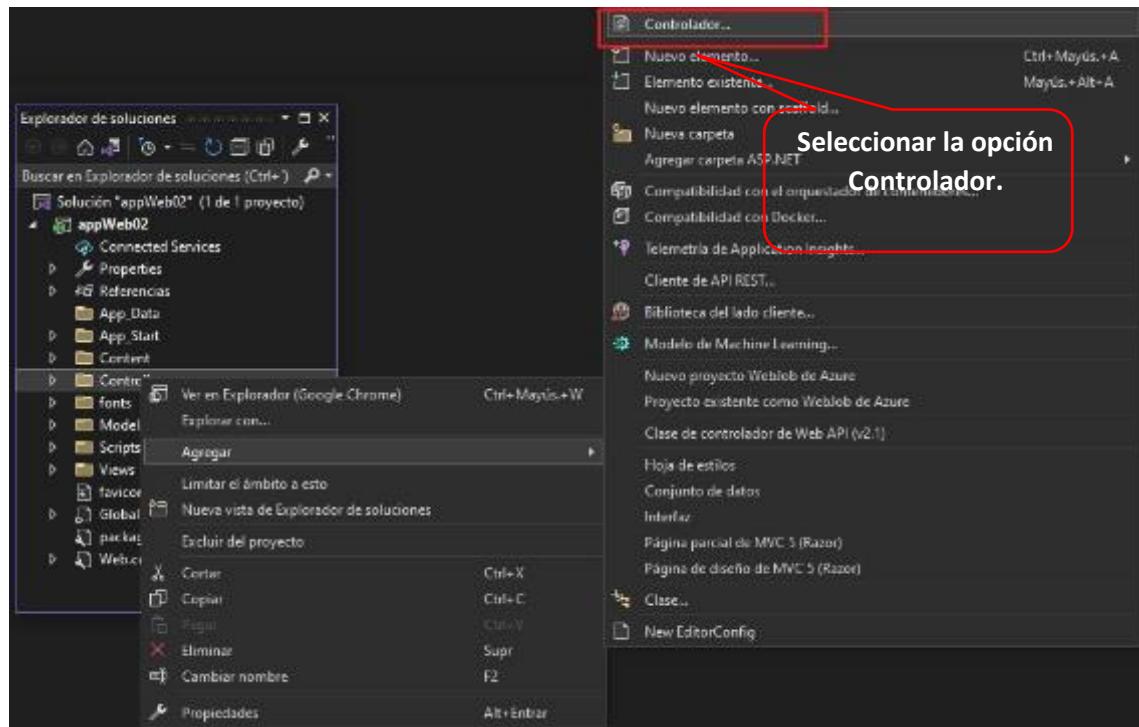
```

Nota. Elaboración propia.

Trabajando con el Controlador

A continuación, en la carpeta Controllers, agregamos un controlador, tal como se muestra:
Controllers → Agregar → Controlador.

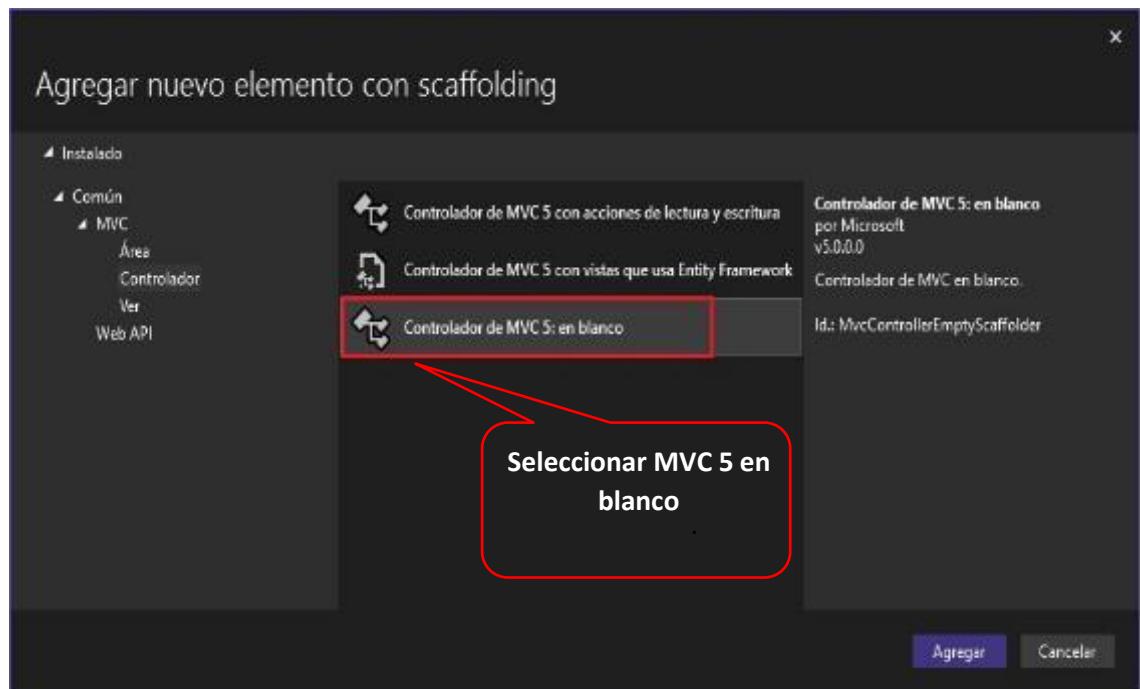
Figura 48
Desarrollo práctico



Nota. Elaboración propia.

Selecciona Controlador MVC5 en blanco, tal como se muestra.

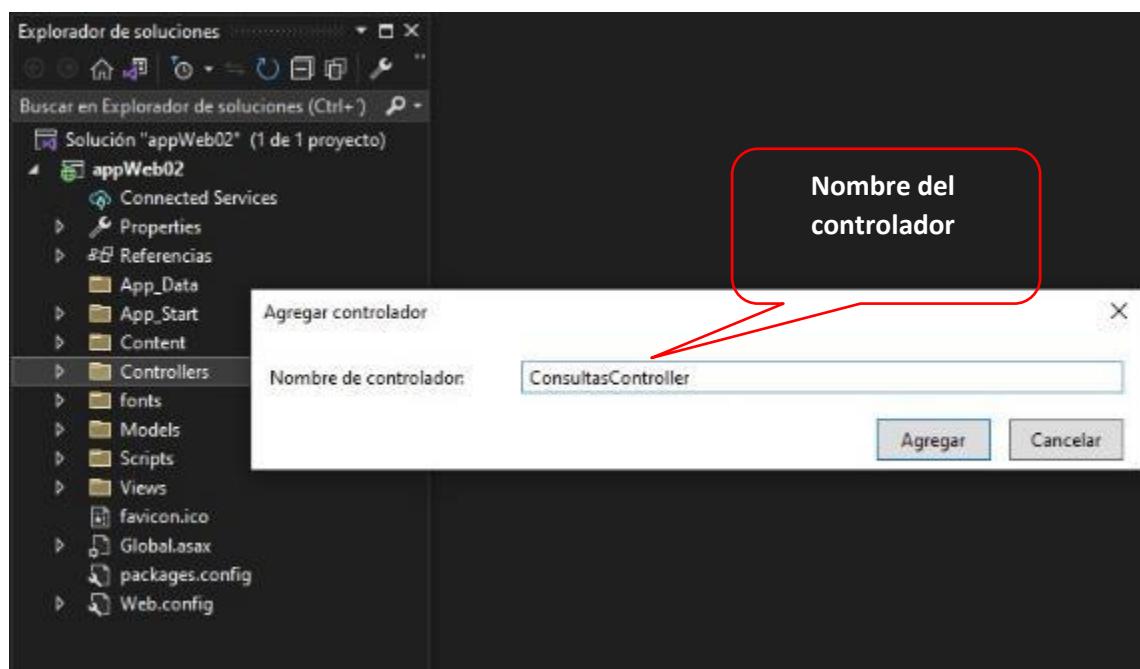
Figura 49
Desarrollo práctico



Nota. Elaboración propia.

Asigne el nombre de **ConsultasController**, tal como se muestra. Presiona el botón Agregar.

Figura 50
Desarrollo práctico



Nota. Elaboración propia.

Programando el Controlador

Importar la carpeta Models, donde se encuentra almacenado la clase.

Figura 51

Desarrollo práctico

```

ConsultasController.cs  X
appWeb02              -> appWeb02.Controllers.ConsultasController
Index()

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using appWeb02.Models;
7  namespace appWeb02.Controllers
8  {
9      public class ConsultasController : Controller
10     {
11         public ActionResult Index()
12     }
13 }

```

Nota. Elaboración propia.

En el Controlador, defina una lista de Cliente llamado clientes, agregar un conjunto de objetos Cliente en la lista, tal como se muestra.

Figura 52

Desarrollo práctico

```

ConsultasController.cs  X
appWeb02              -> appWeb02.Controllers.ConsultasController
Clients

1  using ...
7  namespace appWeb02.Controllers
8  {
9      public class ConsultasController : Controller
10     {
11         public List<Cliente> clientes= new List<Cliente>()
12         {
13             new Cliente(){idcliente="123",nombre="Juan",direccion="Lima",email="juan@gmail.com",fono="5675848"},
14             new Cliente(){idcliente="124",nombre="Hugo",direccion="Ate",email="hugo@gmail.com",fono="5455848"},
15             new Cliente(){idcliente="125",nombre="Ana",direccion="Rinac",email="ana@gmail.com",fono="5565848"},
16             new Cliente(){idcliente="126",nombre="Luis",direccion="Lima",email="luis@gmail.com",fono="5685848"},
17             new Cliente(){idcliente="127",nombre="JuanCa",direccion="Callao",email="juanca@gmail.com",fono="5995848"},
18             new Cliente(){idcliente="128",nombre="Ines",direccion="Ate",email="ines@gmail.com",fono="5675846"},
19         };
20
21         public ActionResult Index()
22     }
23 }

```

Nota. Elaboración propia.

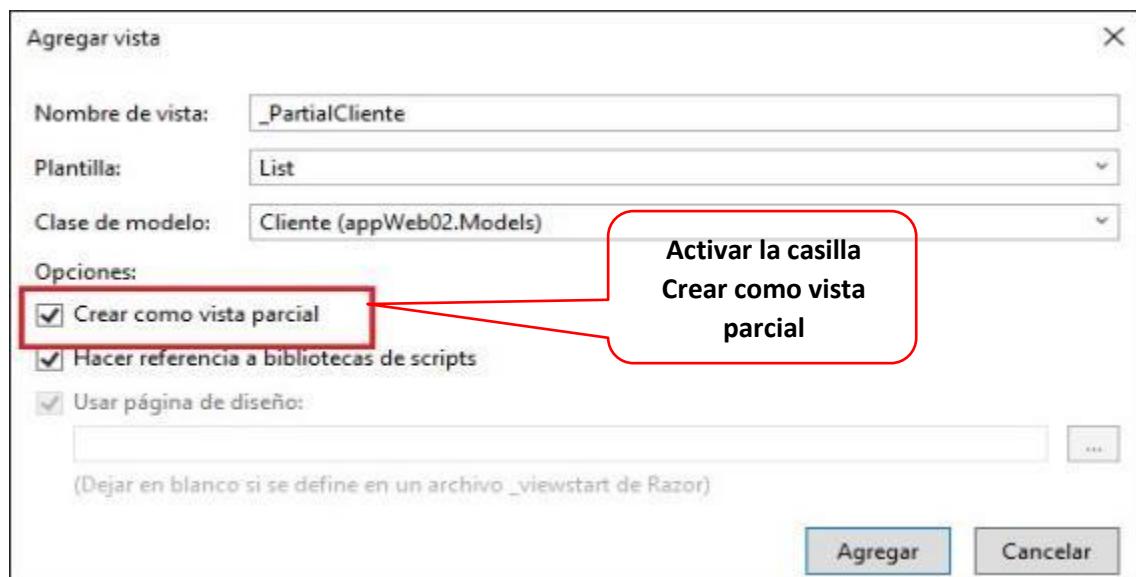
Agregando una Vista Parcial utilizando el Scaffolding

En el controlador, agregamos una Vista la cual nos lleva a la ventana Agregar Vista.

Ingrese el nombre de la Vista **_PartialCliente**, Plantilla: **List** y clase de modelo: **Cliente** Marcar la casilla **Crear como vista parcial**, para convertirla en dicha vista.

Figura 53

Desarrollo práctico



Nota. Elaboración propia.

Diseño de la Vista Parcial

La vista al crearse define una variable @model, que representa la lista de clientes de la cual recibe la colección de clientes para imprimirse en la página.

Figura 54
Desarrollo práctico

```

@model IEnumerable<appWeb02.Models.Cliente>



| @Html.DisplayNameFor(model => model.idcliente) | @Html.DisplayNameFor(model => model.nombre) | @Html.DisplayNameFor(model => model.direccion) | @Html.DisplayNameFor(model => model.email) | @Html.DisplayNameFor(model => model.fono) |                                                                                                                                                                                                                        |
|------------------------------------------------|---------------------------------------------|------------------------------------------------|--------------------------------------------|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @Html.DisplayFor(modelItem => item.idcliente)  | @Html.DisplayFor(modelItem => item.nombre)  | @Html.DisplayFor(modelItem => item.direccion)  | @Html.DisplayFor(modelItem => item.email)  | @Html.DisplayFor(modelItem => item.fono)  | @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ })   @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ })   @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ }) |


```

Nota. Elaboración propia.

Trabajando con los ActionResult Index

El primer ActionResult es Index() el cual envía a la vista la lista de clientes.

Figura 55
Desarrollo práctico

```

using System;
namespace appWeb02.Controllers
{
    public class ConsultasController : Controller
    {
        public List<Cliente> clientes= new List<Cliente>();
        public ActionResult Index()
        {
            return View(clientes);
        }
    }
}

```

Nota. Elaboración propia.

A continuación, agregamos la vista del ActionResult Index(), selecciona la plantilla Empty (sin modelo), tal como se muestra.

Figura 56
Desarrollo práctico



Nota. Elaboración propia.

En la ventana de la vista, agregamos:

- a) Using para referenciar la carpeta Models
- b) Variable Model que recibe la lista Enumerada de Cliente
- c) Html.Partial(), donde invoca a la vista parcial _PartialCliente

Figura 57
Desarrollo práctico

```

Index.cshtml  s x ConsultasController.cs
using appWeb02.Models
@model IEnumerable<Cliente>
{
    ViewBag.Title = "Index";
}

<h2>Listado de Clientes</h2>
<div>
    @Html.Partial("_PartialCliente")
</div>

```

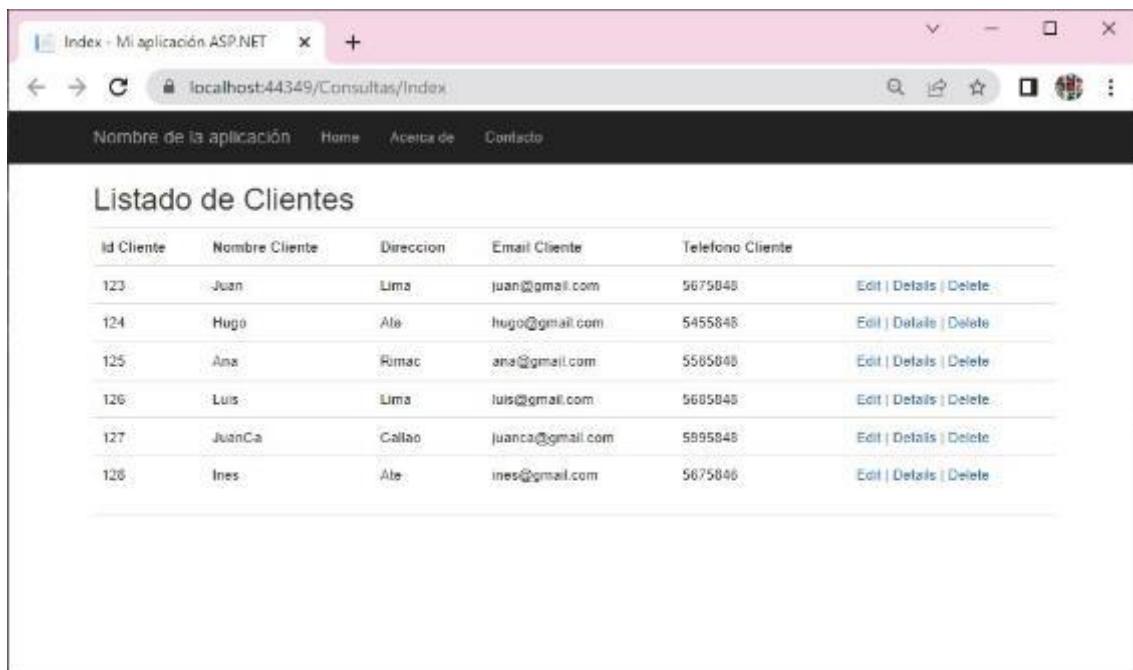
@model recibe la lista de cliente

Imprimir la vista parcial

Nota. Elaboración propia.

Presiona la tecla F5, sobre la vista Index(), la cual visualiza la lista de los clientes, tal como se muestra.

Figura 58
Desarrollo práctico



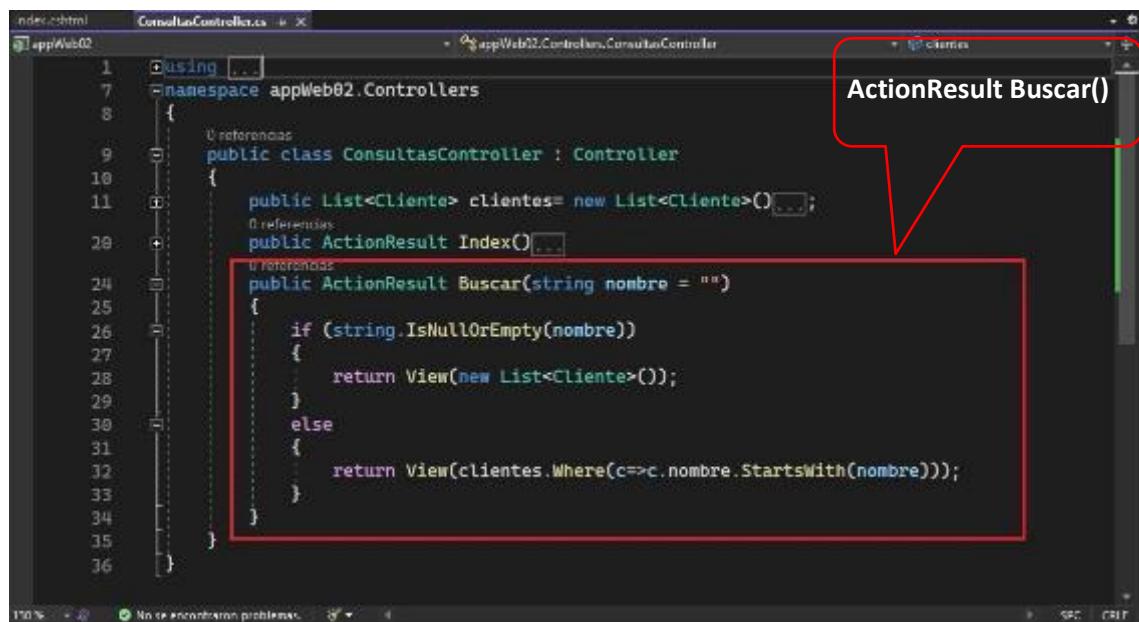
Nota. Elaboración propia.

Trabajando con el ActionResult Buscar

El segundo ActionResult es Buscar() el cual ingresa las iniciales del parámetro “nombre” y envía a la vista la lista de clientes filtrando por sus iniciales.

Figura 59

Desarrollo práctico

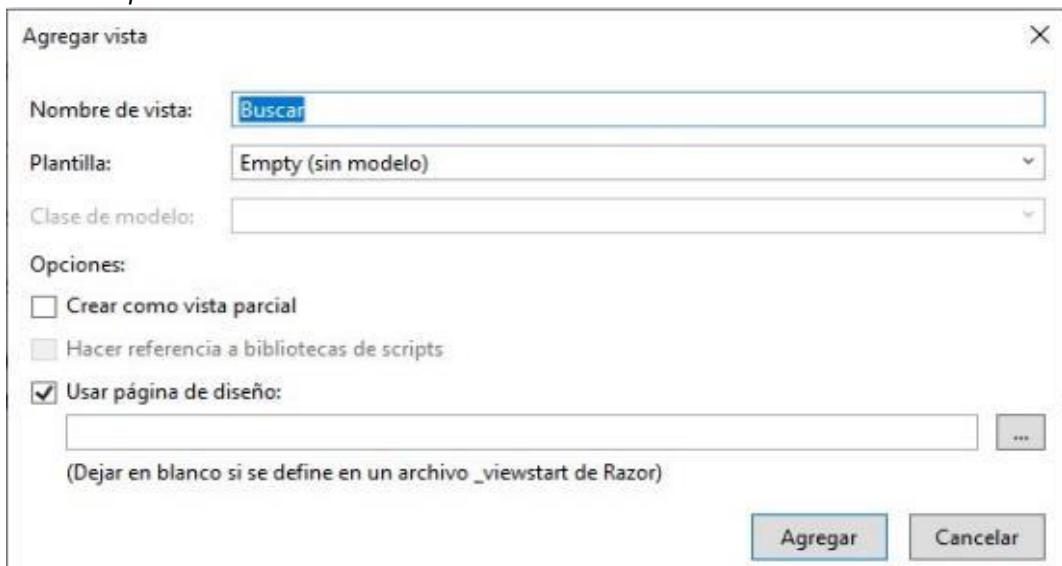


```
index.cshtml    ConsultasController.cs + X
appWeb02        appWeb02.Controllers
1  using ...
7  namespace appWeb02.Controllers
8  {
9      public class ConsultasController : Controller
10     {
11         public List<Cliente> clientes= new List<Cliente>();
12         public ActionResult Index()
13         {
14             return View(clientes);
15         }
16     }
17 }
18
19
20
21
22
23
24     public ActionResult Buscar(string nombre = "")
25     {
26         if (string.IsNullOrEmpty(nombre))
27         {
28             return View(new List<Cliente>());
29         }
30         else
31         {
32             return View(clientes.Where(c=>c.nombre.StartsWith(nombre)));
33         }
34     }
35 }
36 }
```

Nota. Elaboración propia.

A continuación, agregamos la vista a Buscar(), cuya plantilla es Empty (sin modelo), tal como se muestra.

Figura 60
Desarrollo práctico



Nota. Elaboración propia.

En la ventana de la vista, agregamos:

- Using para referenciar la carpeta Models
- Variable Model que recibe la lista Enumerada de Cliente
- Un formulario (Html.BeginForm), donde envía el valor de nombre a través de un TextBox
- Html.Partial(), donde invoca a la vista parcial _PartialCliente

Figura 61
Desarrollo práctico

```

@using appWeb02.Models
@model IEnumerable<Cliente>

@{
    ViewBag.Title = "Buscar";
}



## Buscar Clientes



@using (Html.BeginForm())
    {
        <input type="text" name="nombre" value="" placeholder="Ingrese las Iniciales del Nombre"/>
        <button type="submit">Consulta</button>
    }



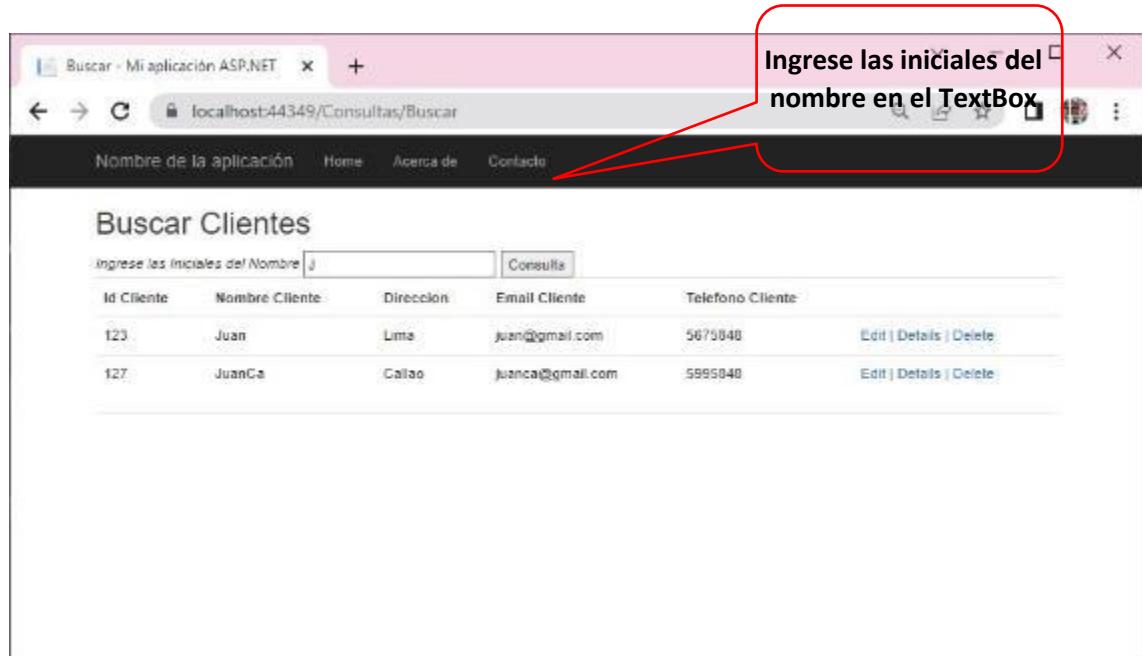
@Html.Partial("_PartialCliente")


```

Nota. Elaboración propia.

Presiona la tecla F5, ingrese las iniciales del nombre del cliente, al presionar el botón de Consulta, visualizamos los clientes que coincidan con dichos valores.

Figura 62
Desarrollo práctico



Nota. Elaboración propia.

Resumen

1. 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.
2. 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.
3. Scaffolding implica la creación de plantillas a través de los elementos del proyecto a través de un método automatizado.
4. 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.
5. Para mantener un aspecto coherente en todas las páginas dentro de su aplicación web es el diseño de "Páginas maestras" que ayudan a definir una plantilla de sitio común y luego heredar su apariencia en todas las vistas de su aplicación web.
6. Los métodos más utilizados en el diseño de vistas Layout: Url.Content(), Html.ActionLink(), RenderSection(), RenderBody().
7. Para que las vistas tengan el diseño de la vista Layout: @ { Diseño = "~ / Views / Shared / _Layout.cshtml" ; }
8. La implementación de Microsoft ASP.NET MVC proporciona una alternativa al modelo de formularios Web Forms de ASP.NET para crear aplicaciones web. ASP.NET MVC es un marco de presentación de poca complejidad y fácil de testear.
9. 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.
10. 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

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [https://learn.microsoft.com/en-us/previous-versions/aspnet/dd381412\(v=vs.108\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/aspnet/dd381412(v=vs.108)?redirectedfrom=MSDN)
- <https://www.forosdelweb.com/f179/jquery-c-mvc-crear-objeto-json-enviarlo-action-del-controlador-1082637/>
- <https://mug-it.org.ar/343016-Comunicando-cliente-y-servidor-con-jQuery-en-ASPnetMVC3.note.aspx>
- <https://desarrolloweb.com/articulos/pasar-datos-controladores-vistas-dotnet.html>
- <https://www.uv.mx/personal/ermeneses/files/2017/03/BDAClase14-MVC.pdf>
- <https://www.csharp.com/UploadFile/3d39b4/Asp-Net-mvc-4-layout-and-section-in-razor/>



TRABAJANDO CON DATOS EN ASP.NET MVC

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al finalizar la unidad, el alumno desarrolla aplicaciones con acceso a datos en el entorno Web utilizando el patrón MVC, implementando operaciones de consulta y actualización de datos.

TEMARIO

2.1 Tema 3 : Introducción a ADO.NET

- 2.1.1 : Arquitectura, proveedor de datos en ADO.NET
- 2.1.2 : Trabajando una conexión a un origen de datos
- 2.1.3 : Publicación de una cadena de conexión

2.2 Tema 4 : Recuperación de datos y paginación

- 2.2.1 : Clase DataReader: métodos y propiedades
- 2.2.2 : Consulta de datos con parámetros utilizando DataReader
- 2.2.3 : Trabajando con vistas parciales en una consulta de datos
- 2.2.4 : Paginación de datos recuperados

2.3 Tema 5 : Manipulación de datos

- 2.3.1 : Validaciones de datos: uso de DataAnnotations
- 2.3.2 : Operaciones de actualización sobre un origen de datos, manejo de la clase Command
- 2.3.3 : Trabajando con imágenes: uso de la clase File y HttpPostFile
- 2.3.4 : Manejo de transacciones, uso de la clase Transaction

2.4 Tema 6 : Arquitectura de capas con acceso a datos

- 2.4.1 : Introducción
- 2.4.2 : Implementando una arquitectura de capas en un proceso de actualización de datos

ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web utilizando en patrón de diseño Modelo Vista Controlador, para realizar procesos de consulta a una fuente de datos.
- Los alumnos desarrollan los laboratorios de esta semana.

2.1. INTRODUCCIÓN A ADO.NET

La mayoría de las aplicaciones desarrollado en Visual Basic y Visual C# giran en torno a la recuperación y actualización de datos de un origen de datos. Para que las aplicaciones se encuentren integradas aun determinado origen de datos, Visual Studio .NET ha desarrollado una nueva generación de tecnología de acceso a datos: ADO.NET.

ADO.NET es un conjunto de clases que exponen un conjunto de librerías y servicios de acceso a datos para programadores de .NET Framework. ADO.NET constituye una parte integral de .NET Framework y proporciona acceso a datos relacionales, XML y de aplicaciones. ADO.NET satisface diversas necesidades de desarrollo, como la creación de clientes de base de datos front-end y objetos empresariales de nivel medio que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet.

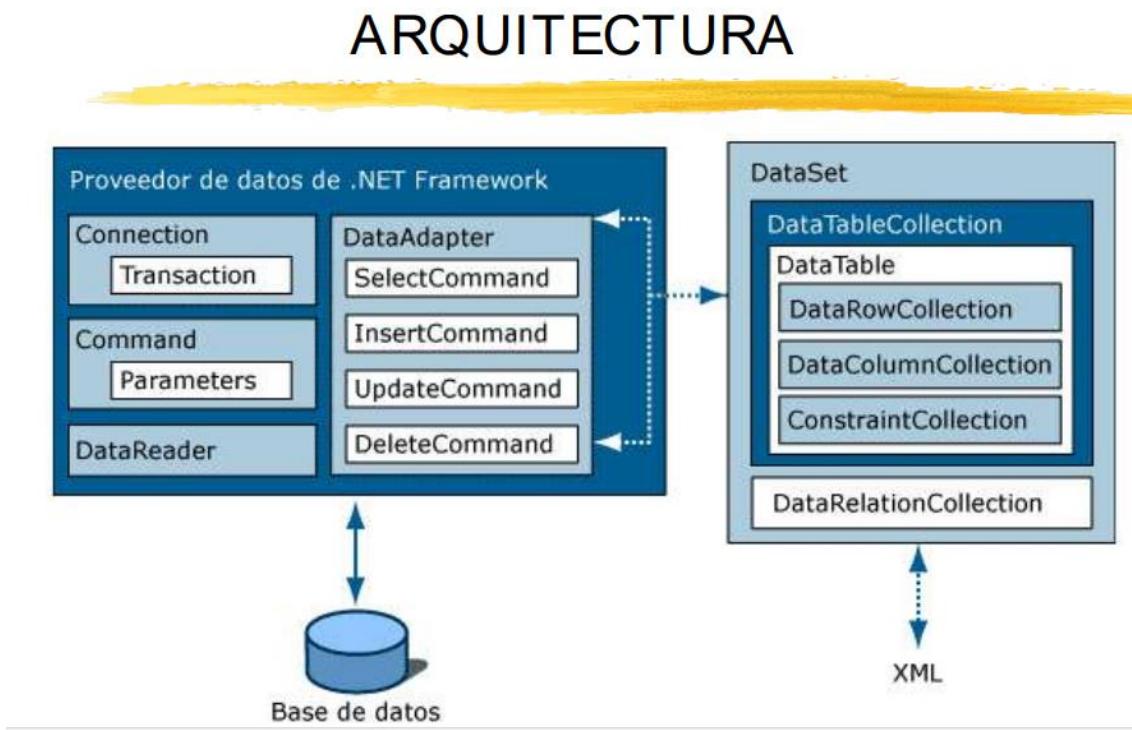
En la actualidad ADO.NET es parte del .NET Framework, esto quiere decir que es, de alguna manera, parte del sistema operativo y no más un redistribuible de 4 ó 5 MB que se necesita alojar junto al cliente o junto al instalador de una aplicación. Esto significa que nosotros, como desarrolladores, estaremos enfocados más al acceso a datos y a la lógica para manipular estos datos, y no a crear una librería para acceder a los datos.

Si nuestras aplicaciones van a tener un ciclo de vida largo, entonces debe considerar la posibilidad de rediseñar la tecnología de acceso a datos de la aplicación y utilizar ADO.NET en aplicaciones administradas. El uso de las tecnologías más modernas de acceso a datos reduce el tiempo de desarrollo, simplifica el código y proporciona un rendimiento excelente.

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.

2.1.1. Arquitectura, proveedor de datos en ADO.NET

Figura 63
Arquitectura ADO.NET



Nota. Tomado de *ADO.NET*, por desarrollodesoftwarelizana.blogspot.com, 2017, (<https://desarrollodesoftwarelizana.blogspot.com/>)

La arquitectura de ADO.NET consta en dos partes primarias:

1. Data provider o Proveedor de datos

“Estas clases proporcionan el acceso a una fuente de datos, como Microsoft SQL Server y Oracle. Cada fuente de datos tiene su propio conjunto de objetos del proveedor, pero cada uno tienen un conjunto común de clases de utilidad” (Alberca y Cuello, 2017).

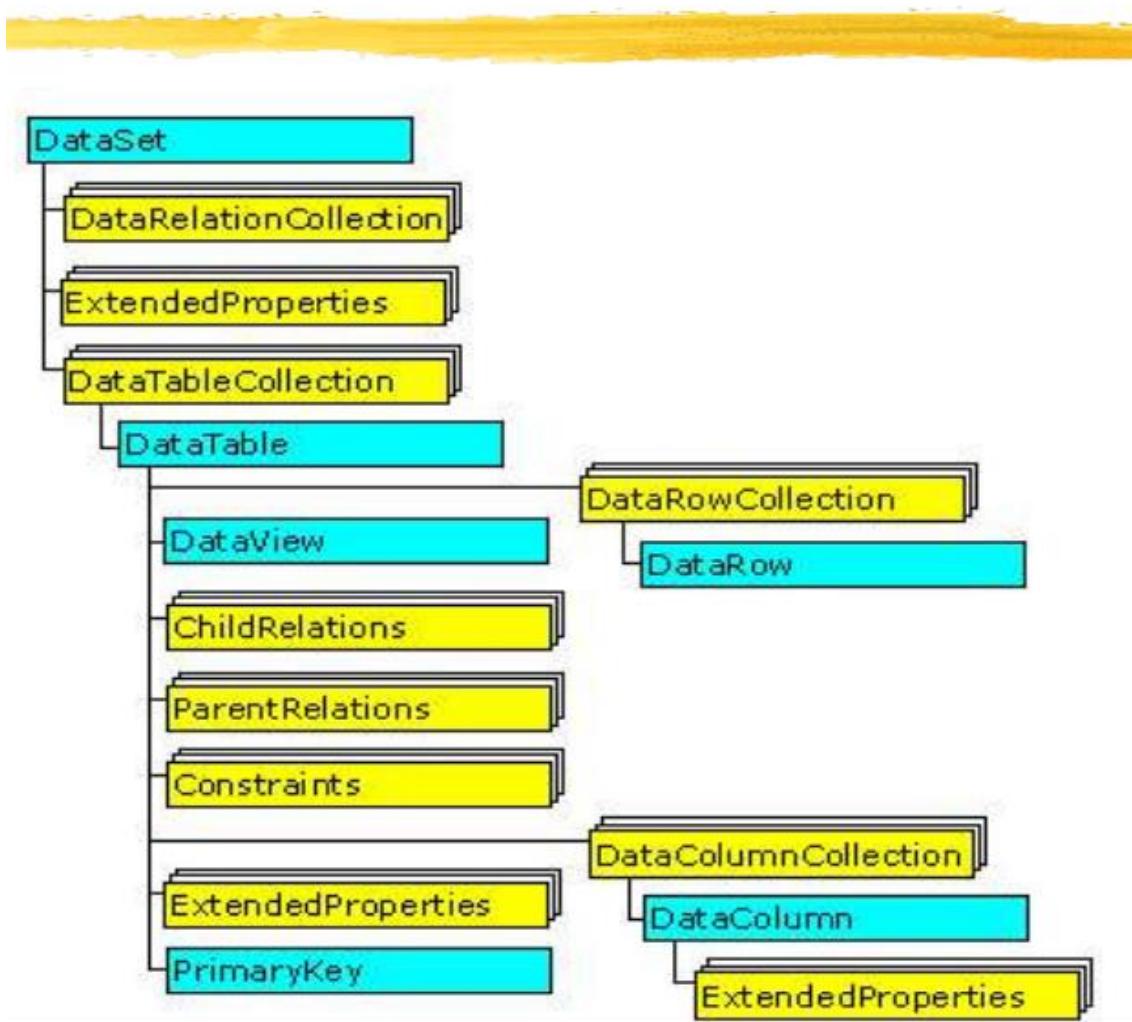
- **Connection:** proporciona una conexión usada para comunicarse con la fuente de datos. También actúa como Abstract Factory para los objetos command.
- **Command:** usado para realizar alguna acción en la fuente de datos, como lectura, actualización, o borrado de datos relacionales.
- **Parameter:** describe un simple parámetro para un command. Un ejemplo común es un parámetro para ser usado en un procedimiento almacenado.
- **DataAdapter:** “puente” utilizado para transferir data entre una fuente de datos y un objeto DataSet (ver abajo).
- **DataReader:** es una clase usada para procesar eficientemente una lista grande de resultados, un registro a la vez.

2. DataSets

Los objetos DataSets, son un grupo de clases que describen una simple base de datos relacional en memoria.

Figura 64
DataSet y sus componentes

DataSet de ADO.NET



Nota. Tomado de ADO.NET, por desarrollodesoftwarelizana.blogspot.com, 2017,
[\(https://desarrollodesoftwarelizana.blogspot.com/\)](https://desarrollodesoftwarelizana.blogspot.com/)

Las clases forman una jerarquía:

- DataSet: representa un esquema (o una base de datos entera o un subconjunto de una). Puede contener las tablas y las relaciones entre esas tablas.
- DataTable: representa una sola tabla en la base de datos. Tiene un nombre, filas, y columnas.
- DataView: vista de un DataTable y ordena los datos (como una cláusula "order by" de SQL) y, si se activa un filtro, filtra los registros (como una cláusula "where" del SQL).

Para facilitar estas operaciones se usa un índice en memoria. Todas las DataTables tienen un filtro por defecto, mientras que pueden ser definidos cualquier número de DataViews adicionales, reduciendo la interacción con la base de datos subyacente y mejorando así el desempeño.

- DataColumn: representa una columna de la tabla, incluyendo su nombre y tipo.
- DataRow: representa una sola fila en la tabla, y permite leer y actualizar los valores en esa fila, así como la recuperación de cualquier fila que esté relacionada con ella a través de una relación de clave primaria - clave foránea.
- DataRowView: representa una sola fila de un DataView, la diferencia entre un DataRow y el DataRowView es importante cuando se está interactuando sobre un resultset.
- DataRelation: es una relación entre las tablas, tales como una relación de clave primaria - clave foránea. Esto es útil para permitir la funcionalidad del DataRow de recuperar filas relacionadas.
- Constraint: describe una propiedad de la base de datos que se debe cumplir, como que los valores en una columna de clave primaria deben ser únicos. A medida que los datos son modificados cualquier violación que se presente causará excepciones.

Un DataSet es llenado desde una base de datos por un DataAdapter cuyas propiedades Connection y Command que han sido iniciados. Sin embargo, un DataSet puede guardar su contenido a XML (opcionalmente con un esquema XSD), o llenarse a sí mismo desde un XML, haciendo esto excepcionalmente útil para los servicios web, computación distribuida, y aplicaciones ocasionalmente conectadas desconectados.

Proveedores de datos en 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, o se colocan en un DataSet con el fin de que el usuario pueda visualizarlos 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

Tabla 2

Proveedores de datos .NET

Proveedores de datos .NET	Descripción
Proveedor de datos .NET para SQL Server	Proporciona el acceso a los datos para Microsoft SQL Server. Utiliza la librería System.Data.SqlClient
Proveedor de datos .NET para OLEDB	Proporciona el acceso a datos para los orígenes de datos que se exponen mediante OLE DB. Utiliza la librería System.Data.OleDb.
Proveedor de datos .NET para ODBC	Proporciona el acceso a datos para los orígenes de datos que se exponen mediante ODBC. Utiliza la librería System.Data.ODBC

Proveedor de datos .NET para Oracle	Proporciona el acceso a datos para la fuente de datos que se exponen mediante Oracle. Utiliza la librería Oracle.ManagedDataAccess.Client. La librería System.Data.OracleClient se encuentra en desuso.
Proveedor de datos .NET para MySQL	Proporciona el acceso a datos para la fuente de datos que se exponen mediante MySQL. Utiliza la librería MySql.Data.MySqlClient
Proveedor EntityClient	Proporciona el acceso a datos para las aplicaciones de Entity Data Model. Utiliza la librería System.Data.EntityClient

Nota. Adaptado de *Proveedores de datos de .NET Framework*, por Microsoft, 2023, (<https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/data-providers>)

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

Tabla 3
Proveedores de datos .NET

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 .

Nota. Adaptado de *Proveedores de datos de .NET Framework*, por Microsoft, 2023, (<https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/data-providers>)

Junto con las clases principales citadas en la tabla anterior, los proveedores de datos .NET incluyen los siguientes objetos:

Tabla 4*Objetos*

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 llenar 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 .
ClientPermission	Se proporciona para seguridad de acceso proveedores de datos.

Nota. Adaptado de *Proveedores de datos de .NET Framework*, por Microsoft, 2023, (<https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/data-providers>)

2.1.2. Trabajando una 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 DbConnection:

- Proveedor de datos para OLEDB incluye un objeto OleDbConnection
- Proveedor de datos para SQL Server incluye un objeto SqlConnection
- Proveedor de datos para ODBC incluye un objeto OdbcConnection
- Proveedor de datos para Oracle incluye un objeto OracleConnection

Conectarse 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.

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

```
SqlConnection cn = new SqlConnection ("server=(local); database=NorthWind; uid=sa;
pwd=sq1");
cn.Open();
```

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.

2.1.3. Publicación de una cadena de conexión

Es común definir una cadena de conexión a un origen de datos, en forma estática por cuanto consideramos que ésta no cambia durante las etapas de producción; pero cuando la aplicación se encuentra en la etapa de implementación seguramente sea necesaria su adaptación al entorno.

Por lo general, debemos buscar un lugar que impacte lo menos posible en el desarrollo: algunos desarrolladores tienden a crear una clase definiendo atributos y propiedades de retorno dentro del propio código; el problema está es que se requiere recompilar por completo el desarrollo, además de tener que actualizar cada cliente por un simple cambio de configuración.

Para el manejo de la conexión, utilizamos la propuesta que hace .net: publicamos la conexión en el archivo de configuración de la aplicación: web.config.

Entre las ventajas que este presenta se pueden encontrar:

- una lectura simple, ya que se basa en XML
- fácil acceso y modificación (se puede editar con el Notepad), por lo general este archivo se encuentra junto a la aplicación por lo que la seguridad debería permitir la escritura en esta carpeta.

Para definir una cadena de conexión, abrir el archivo web.config, defina la conexión utilizando la etiqueta <connectionStrings> (en plural porque se pueden definir varias conexiones), tal como se muestra:

Figura 65

Publicando la cadena de conexión

```
<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <add name="nombre_cadena"
      connectionString="server=(local); database=BD; uid=usuario; pwd=clave"/>
  </connectionStrings>
  <configSections>...</configSections>
  <system.web.webPages.razor>...</system.web.webPages.razor>
  <appSettings>...</appSettings>
  <system.webServer>...</system.webServer>
  <system.web>...</system.web>
</configuration>
```

Nota. Elaboración propia.

Para recuperar la cadena de conexión:

- Importamos la librería: System.Configuration;
- Recuperamos la cadena de conexión publicada en el web.config

```
ConnectionStringSettings cnset = ConfigurationManager.ConnectionStrings["cadena"];
SqlConnection cn = new SqlConnection(cnset.ConnectionString);
cn.Open();
```

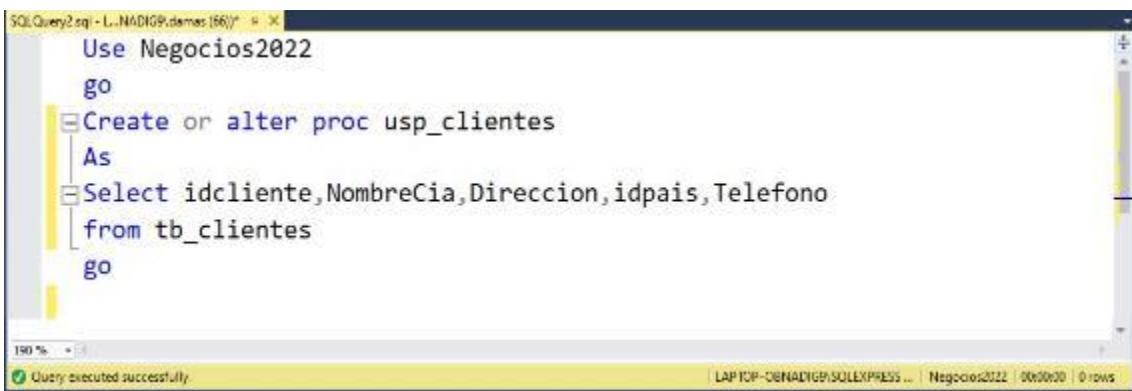
LABORATORIO 3.1.

Se desea implementar una Vista en ASP.NET MVC que permita listar los registros de la tabla tb_clientes almacenados en la base de datos Negocios2022

Creando Procedimiento Almacenado en el SQL Server

Abrir el Manejador del SQL Server, activar la base de datos Negocios2022 y crear el procedimiento almacenado usp_clientes, tal como se muestra.

Figura 66
Desarrollo de laboratorio



The screenshot shows a SQL query window in SSMS. The code is as follows:

```
SQLQuery2.sql - L..NADIGP\damas (66) * X
Use Negocios2022
go
Create or alter proc usp_clientes
As
Select idcliente,NombreCia,Direccion,idpais,Telefono
from tb_clientes
go
```

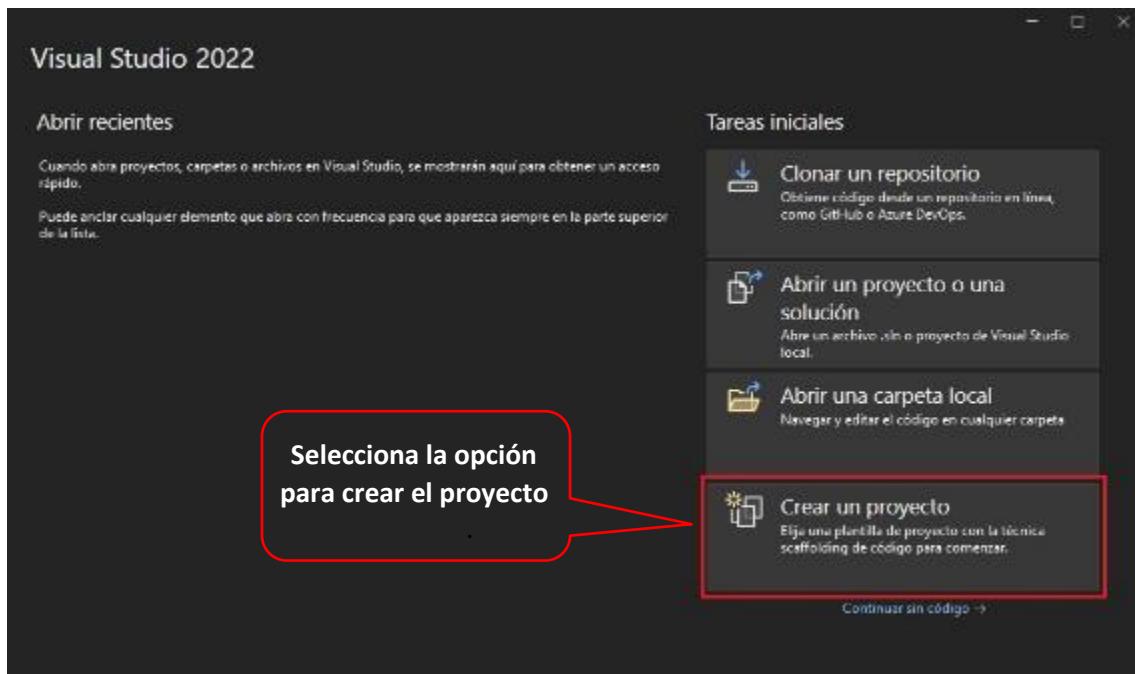
The status bar at the bottom indicates "Query executed successfully".

Nota. Elaboración propia.

Inicio del Proyecto

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

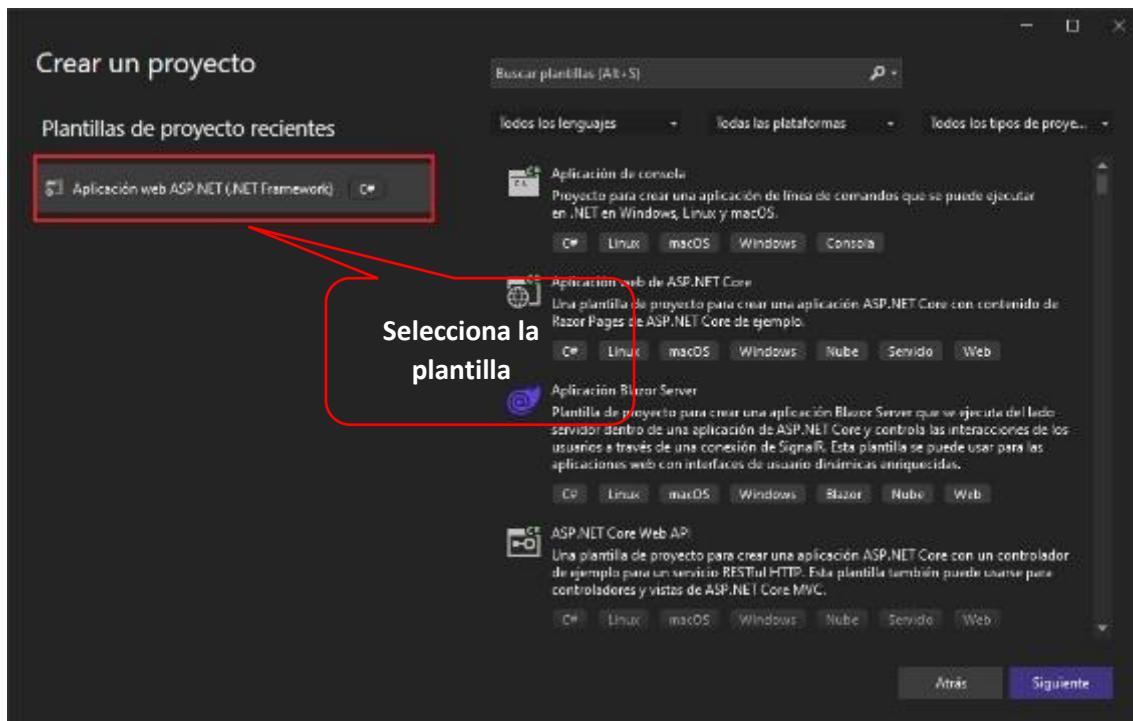
Figura 67
Desarrollo práctico



Nota. Elaboración propia.

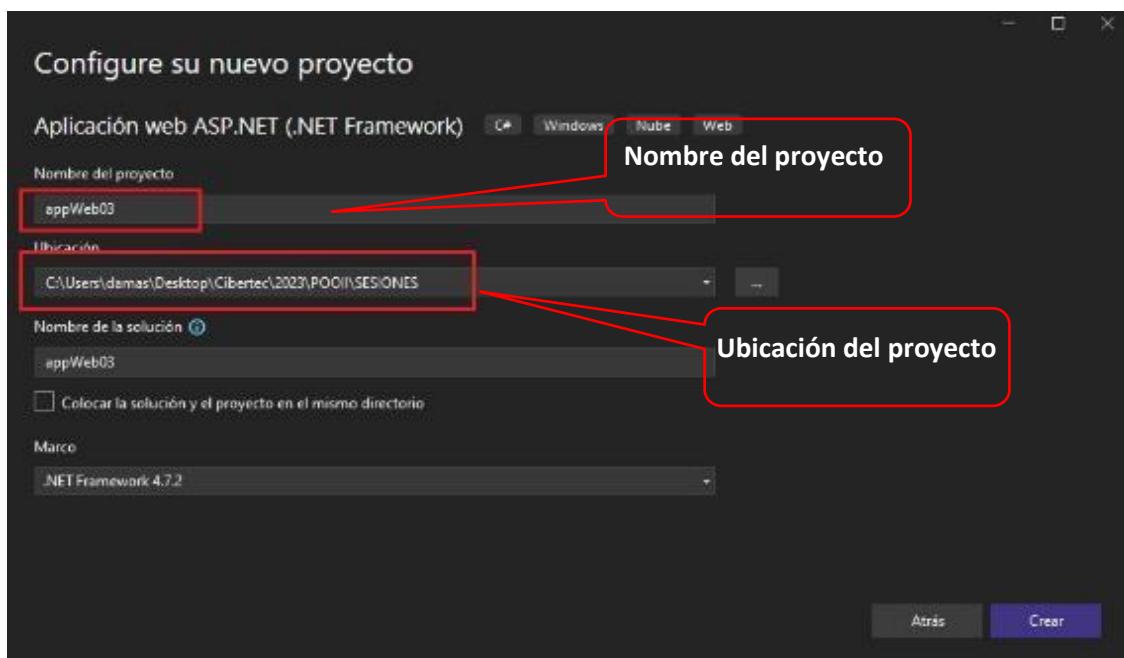
Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción Siguiente.

Figura 68
Desarrollo de Laboratorio



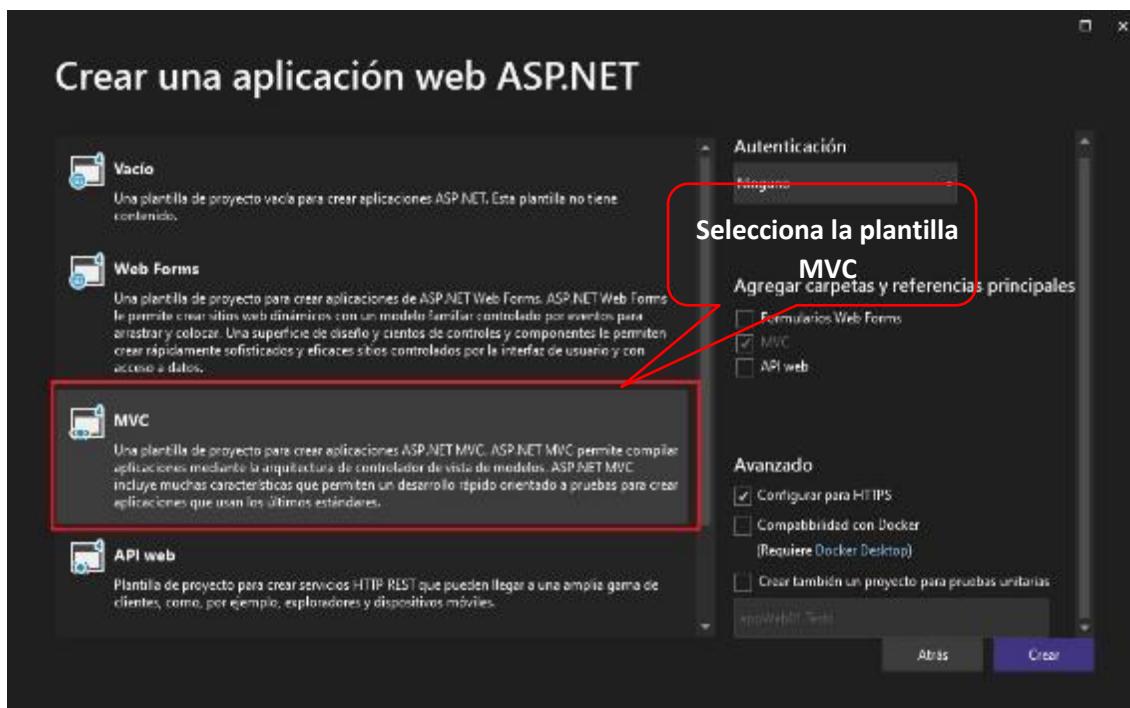
Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciona la ubicación del mismo, al terminar presiona la opción **Crear**.

Figura 69*Desarrollo de Laboratorio*

Nota. Elaboración propia.

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

Figura 70*Desarrollo de Laboratorio*

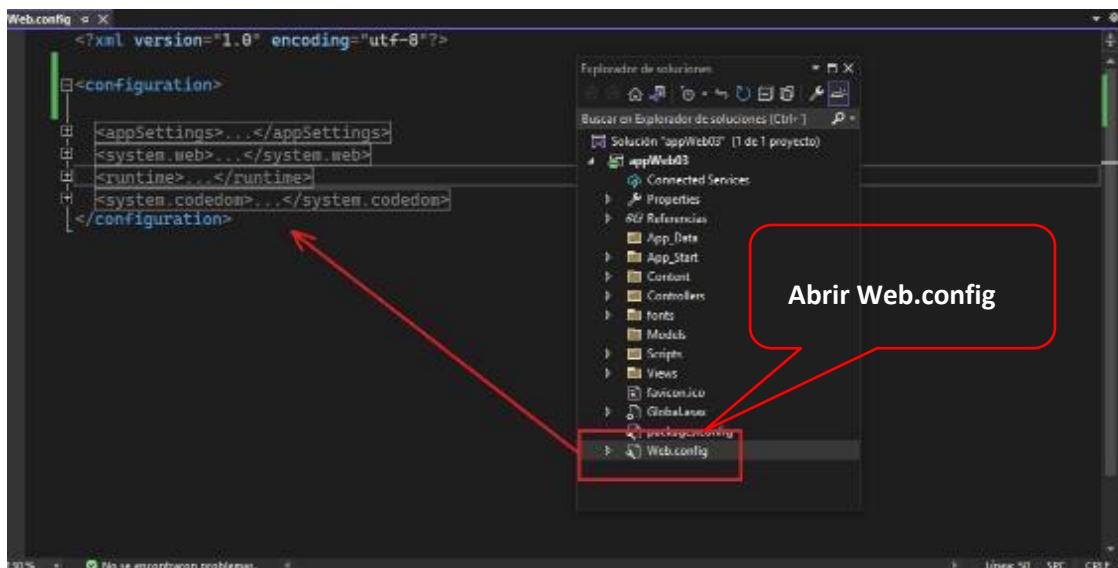
Nota. Elaboración propia.

Publicando la cadena de conexión

Desde el Explorador de proyecto, abrir el archivo Web.config, tal como se muestra.

Figura 71

Desarrollo de Laboratorio

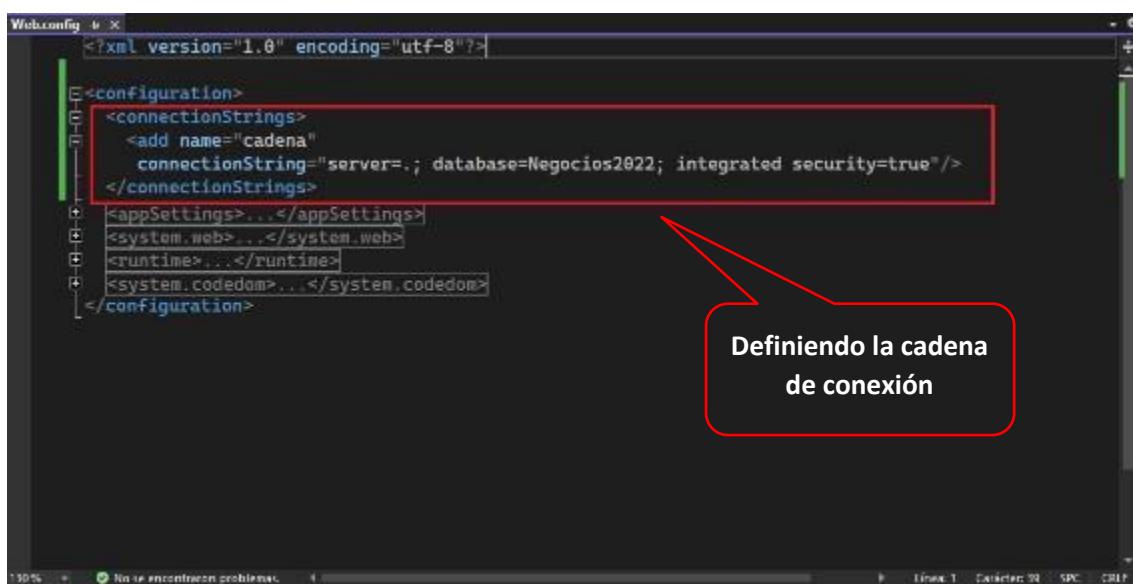


Nota. Elaboración propia.

Defina la etiqueta <connectionStrings> agregando una cadena <add> cuyo nombre es “cadena” y definiendo la cadena de conexión, tal como se muestra.

Figura 72

Desarrollo de Laboratorio



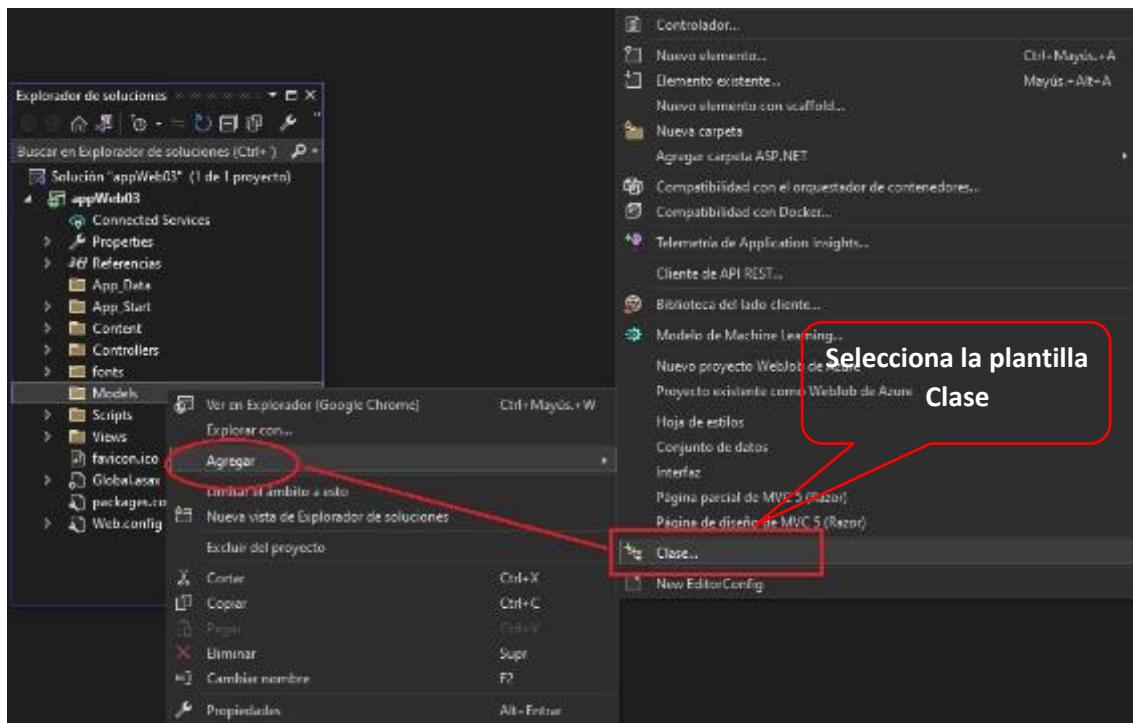
Nota. Elaboración propia.

Agregando la clase Cliente a la carpeta Models

En la carpeta Models, hacer clic derecho sobre la carpeta, seleccionar Agregar → Clase, tal como se muestra en la figura.

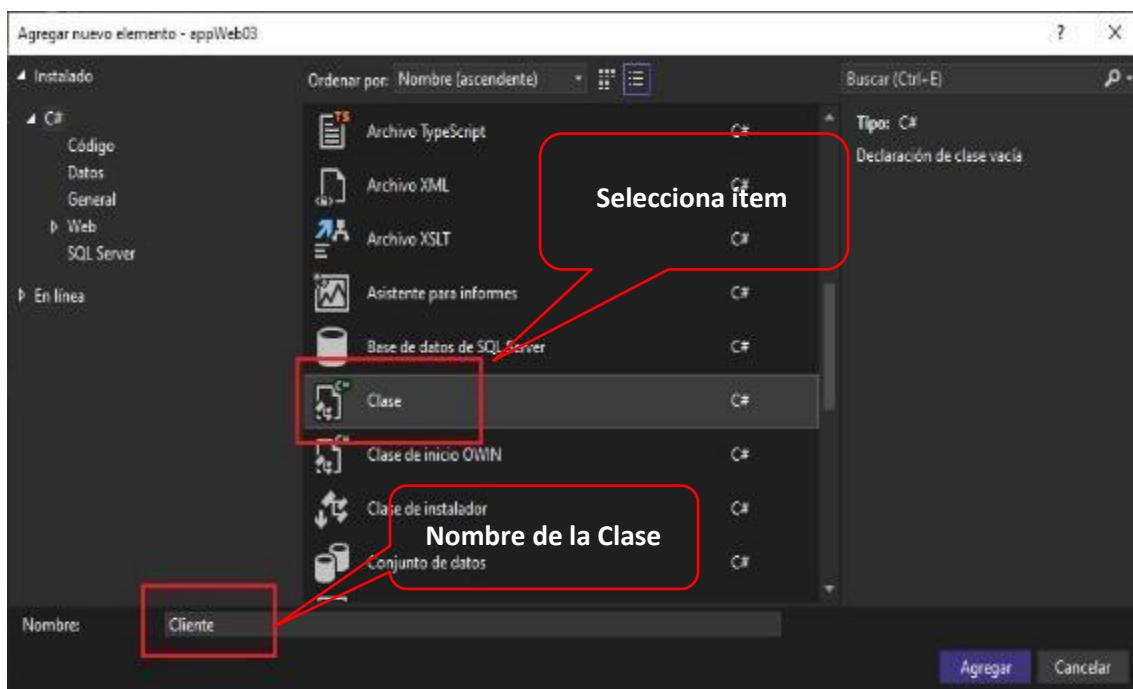
Figura 73

Desarrollo de Laboratorio



Nota. Elaboración propia.

Selecciona el elemento Clase, asigne el nombre del elemento: Cliente, tal como se muestra.

Figura 74*Desarrollo de Laboratorio**Nota.* Elaboración propia.

Agregar la librería System.ComponentModel.DataAnnotations y los atributos de la clase Cliente, tal como se muestra.

Figura 75*Desarrollo de Laboratorio*

```

    1  Busing System;
    2  using System.Collections.Generic;
    3  using System.Linq;
    4  using System.Web;
    5  using System.ComponentModel.DataAnnotations;
    6  Enamespace appWeb03.Models
    7  {
    8      public class Cliente
    9      {
    10         [Display(Name = "Id Cliente")]
    11         public string idcliente { get; set; }
    12         [Display(Name = "Nombre Cliente")]
    13         public string nombrecia { get; set; }
    14         [Display(Name = "Direccion")]
    15         public string direccion { get; set; }
    16         [Display(Name = "Id Pais")]
    17         public string idpais { get; set; }
    18         [Display(Name = "Telefono")]
    19         public string telefono { get; set; }
    20     }
    21 }

```

Importar la librería para utilizar la etiqueta [Display]

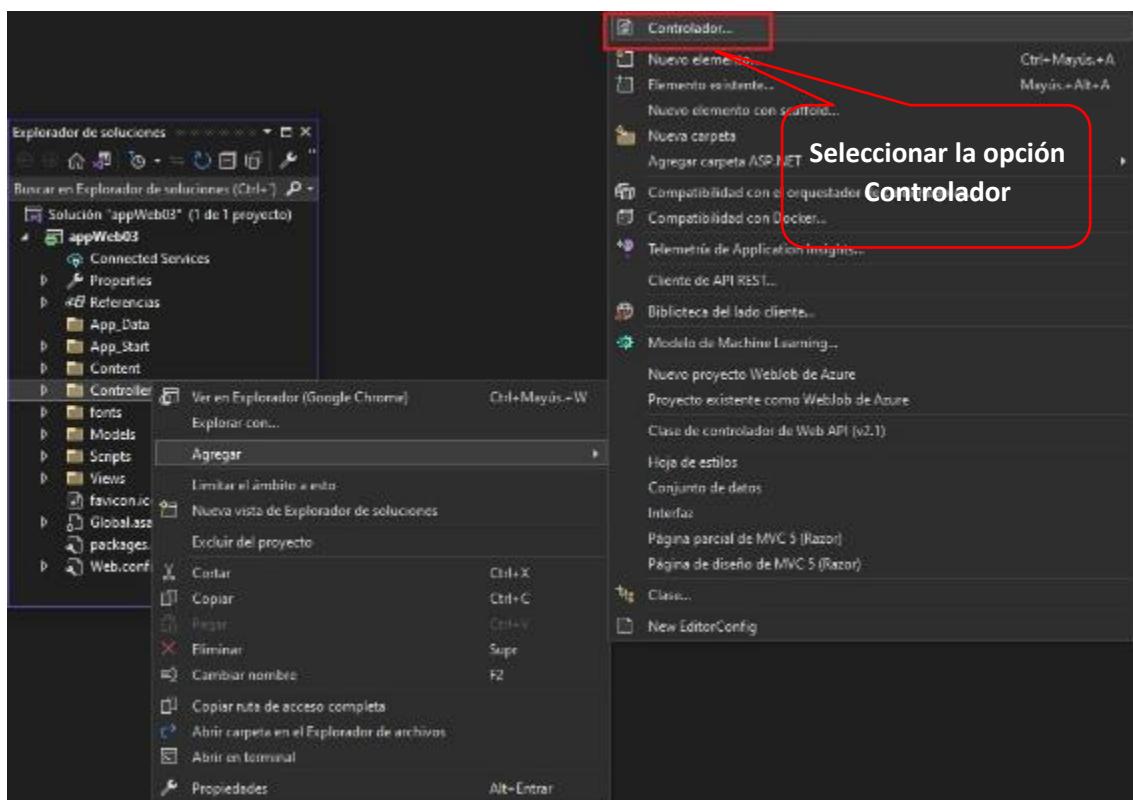
Nota. Elaboración propia.

Trabajando con el Controlador

A continuación, en la carpeta Controllers, agregamos un controlador, tal como se muestra:
Controllers → Agregar → Controlador

Figura 76

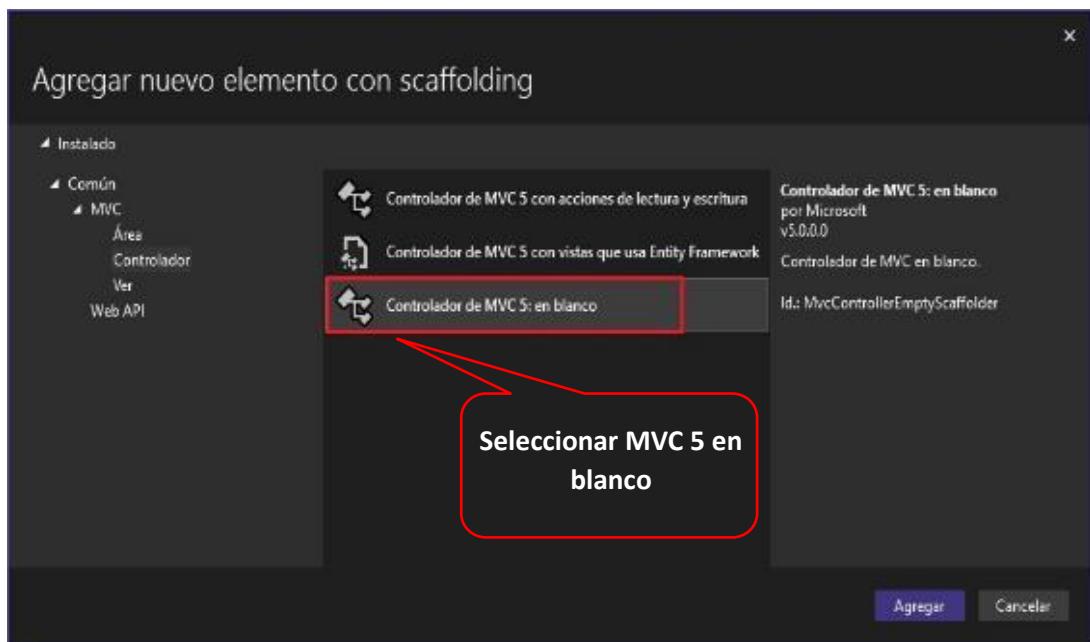
Desarrollo de Laboratorio



Nota. Elaboración propia.

Selecciona Controlador MVC5 en blanco, tal como se muestra.

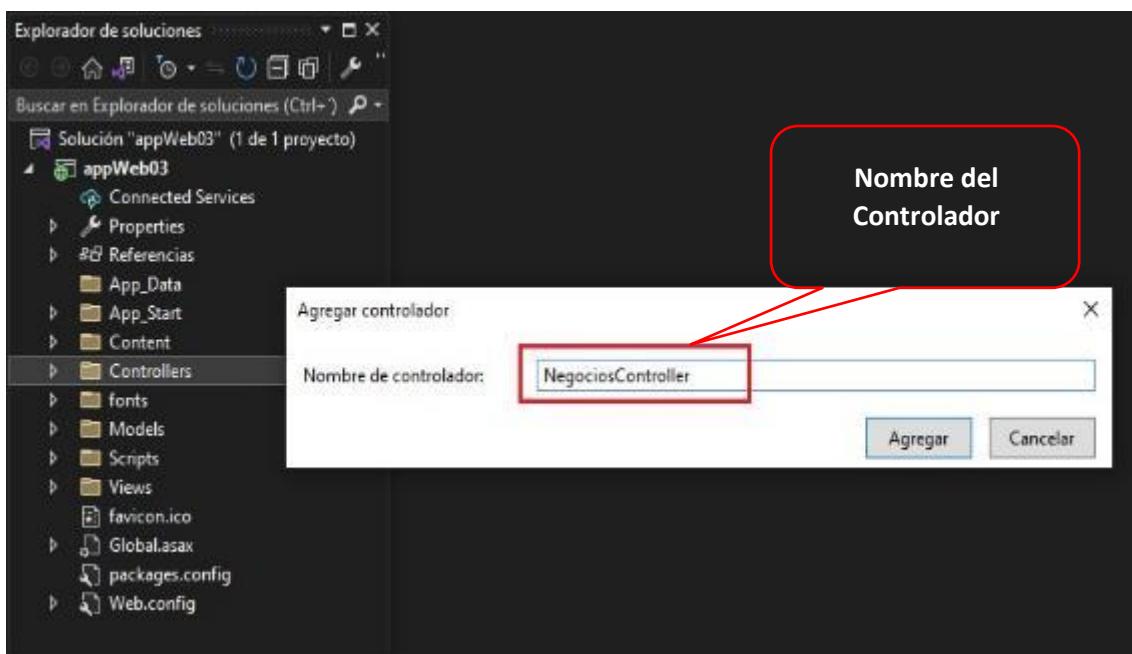
Figura 77
Desarrollo de Laboratorio



Nota. Elaboración propia.

Asigne el nombre de NegociosController, tal como se muestra. Presiona el botón Agregar.

Figura 78
Desarrollo de Laboratorio



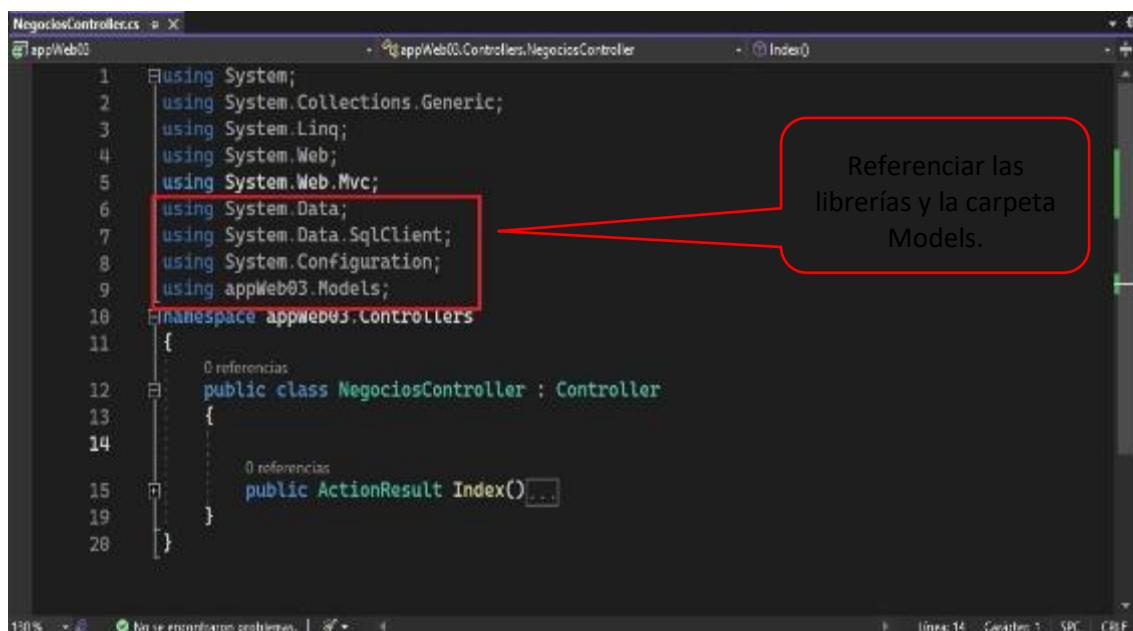
Nota. Elaboración propia.

Programando el Controlador

Importar las librerías y la carpeta Models, donde se encuentra almacenado la clase.

Figura 79

Desarrollo de Laboratorio



```
NegociosController.cs  X
appWeb03
  ↳ appWeb03.Controllers.NegociosController
    ↳ Index()
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6   using System.Data;
7   using System.Data.SqlClient;
8   using System.Configuration;
9   using appWeb03.Models;
10  namespace appWeb03.Controllers
11  {
12      public class NegociosController : Controller
13      {
14          public ActionResult Index()
15      }
16  }
```

Referenciar las librerías y la carpeta Models.

Nota. Elaboración propia.

1. En el Controlador, defina y codifique el método clientes() donde retorna la lista numerada de clientes: 1. Defina la conexión a través del SqlConnection, a través del bloque **using**.
2. Ejecute el procedure a través del SqlCommand, almacenando los resultados en el SqlDataReader.
3. 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.
4. Cerrar los objetos y enviar el objeto llamada temporal.

Figura 80
Desarrollo de Laboratorio

```

NegociosController.cs 12
13     public class NegociosController : Controller
14     {
15         [Referencia]
16         IEnumerable<Cliente> clientes()
17         {
18             List<Cliente> temporal = new List<Cliente>();
19             using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
20             {
21                 cn.Open();
22                 SqlCommand cmd = new SqlCommand("exec usp_clientes", cn);
23
24                 SqlDataReader dr = cmd.ExecuteReader();
25                 while (dr.Read())
26                 {
27                     temporal.Add(new Cliente()
28                     {
29                         idcliente=dr.GetString(0),
30                         nombrecia=dr.GetString(1),
31                         direccion=dr.GetString(2),
32                         idpais=dr.GetString(3),
33                         fono=dr.GetString(4)
34                     });
35                 }
36             }
37             dr.Close();
38             return temporal;
39         }
40     }

```

Nota. Elaboración propia.

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

Figura 81
Desarrollo de Laboratorio

```

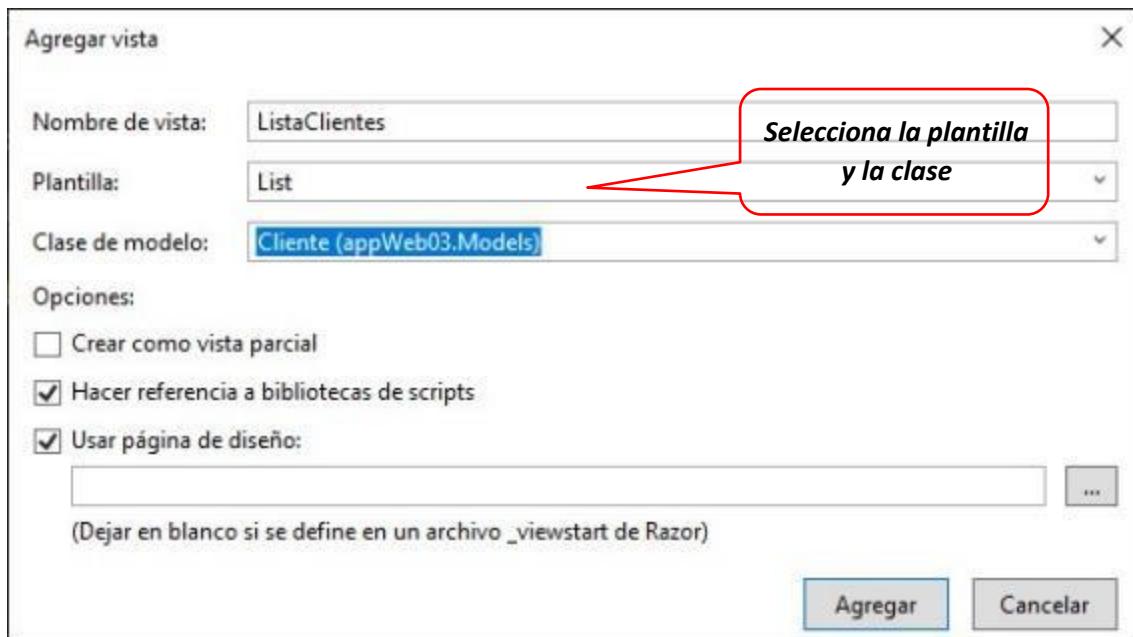
NegociosController.cs 10
11     namespace appWeb03.Controllers
12     {
13         [Referencia]
14         public class NegociosController : Controller
15         {
16             [Referencia]
17             IEnumerable<Cliente> clientes()...
18
19             [Referencia]
20             public ActionResult ListaClientes()
21             {
22                 return View(clientes());
23             }
24         }
25     }

```

Nota. Elaboración propia.

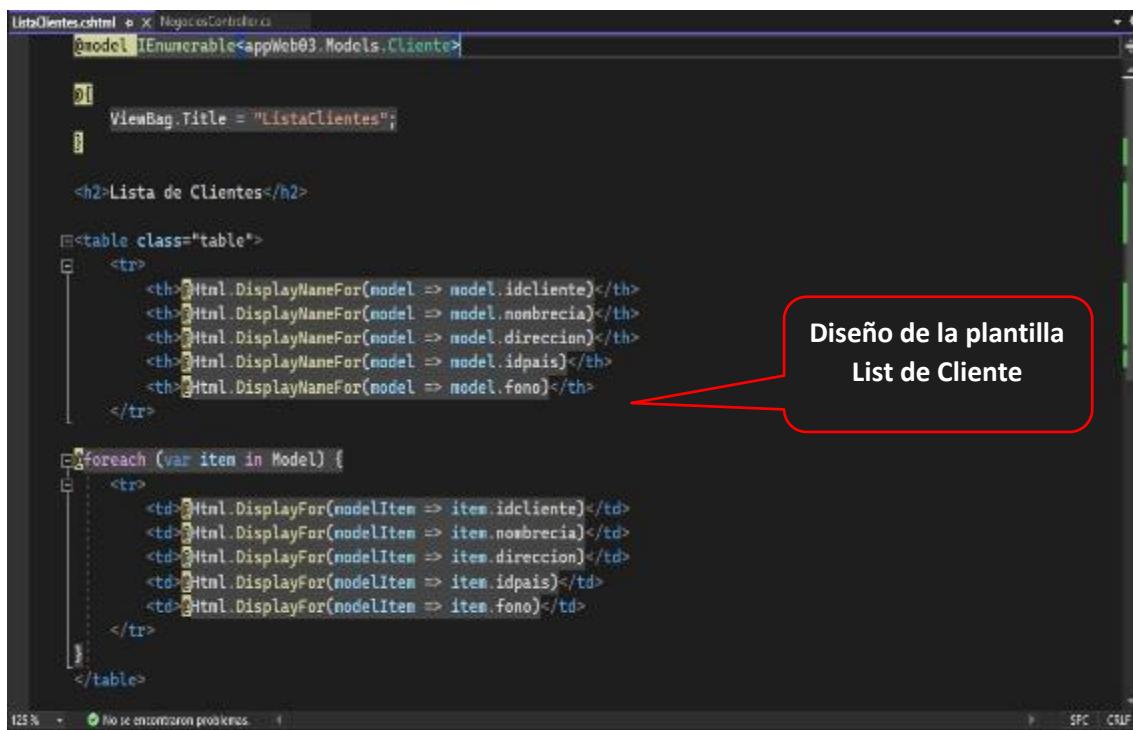
Trabajando con la Vista ListaClientes

- En el ActionResult, hacer clic derecho y selecciona, Agregar vista.
- En dicha ventana, seleccione la plantilla, la cual será List; y la clase de modelo la cual es Cliente, tal como se muestra.

Figura 82*Desarrollo de Laboratorio**Nota. Elaboración propia.*

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

Figura 83
Desarrollo de Laboratorio



```
ListaClientes.cshtml • X NegociosControlador.cs
model IEnumerable<appWeb03.Models.Cliente>

@{
    ViewBag.Title = "ListaClientes";
}



## Lista de Clientes



| @Html.DisplayNameFor(model => model.idcliente)                                                                                                                                                                                                                                                                                                                         | @Html.DisplayNameFor(model => model.nombreclia) | @Html.DisplayNameFor(model => model.direccion) | @Html.DisplayNameFor(model => model.idpais) | @Html.DisplayNameFor(model => model.fono) |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|------------------------------------------------|---------------------------------------------|-------------------------------------------|
| foreach (var item in Model) {         <td>@Html.DisplayFor(modelItem => item.idcliente)</td>         <td>@Html.DisplayFor(modelItem => item.nombreclia)</td>         <td>@Html.DisplayFor(modelItem => item.direccion)</td>         <td>@Html.DisplayFor(modelItem => item.idpais)</td>         <td>@Html.DisplayFor(modelItem => item.fono)</td>     </tr> } </table> |                                                 |                                                |                                             |                                           |


```

Nota. Elaboración propia.

Presiona la tecla F5 para ejecutar la Vista del Proyecto visualizando la lista de los registros de tb_clientes.

Figura 84*Desarrollo de Laboratorio*

Id Cliente	Nombre Cliente	Dirección	Id País	Teléfono
ALFKI	Alfreds Futterkiste	Obere Str. 57	002	030-0074321
ANATR	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	005	(5) 555-4729
ANTON	Antonio Moreno Taquería	Mataderos 2312	007	(5) 555-3932
AROUT	Around the Horn	120 Hanover Sq.	004	(71) 555-7788
BEROS	Berglunds snabbköp	Berguvsvägen 8	006	0921-12 34 65
BLAUS	Blauer See Delikatessen	Forsterstr. 57	001	0621-08460
BLONP	Blondel père et fils	24, place Kleber Estrasburgo	008	88 60 15 31
BOLID	Bolido Comidas preparadas	C/ Araquil, 67	009	(91) 555 91 99
BONAP	Bon app	12, rue des Bouchers	003	91 24 45 41
BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	003	(604) 555-3745
BSBEV	Bla Beverages	Fauntleroy Circus	009	(71) 555-1212
CACTU	Cactus Comidas para llevar	Cerillos 333	002	(1) 135-4892
GEMTC	Centro comercial Multicenter	Passeio das Coqueiras 8000	008	(61) 555-7008

Nota. Elaboración propia.

LABORATORIO 3.2.

Se desea implementar una Vista en ASP.NET MVC que permita listar los registros de la tabla tb_productos, ejecutando un procedimiento almacenado en la base de datos Negocios2022.

1. Creando el procedimiento almacenado

En el Manejador del Sql Server, activar la base de datos y crear el procedimiento almacenado usp_productos, tal como se muestra.

Figura 85*Desarrollo de Laboratorio*

```

SQLQuery1 - L-NAD-07.dbo... (32) < X
Use Negocios2022
go
Create or alter proc usp_productos
As
Select IdProducto,NombreProducto,NombreCategoria,PrecioUnidad,UnidadesEnExistencia
from tb_productos p join tb_categorias c on p.idCategoria=c.idCategoria
go

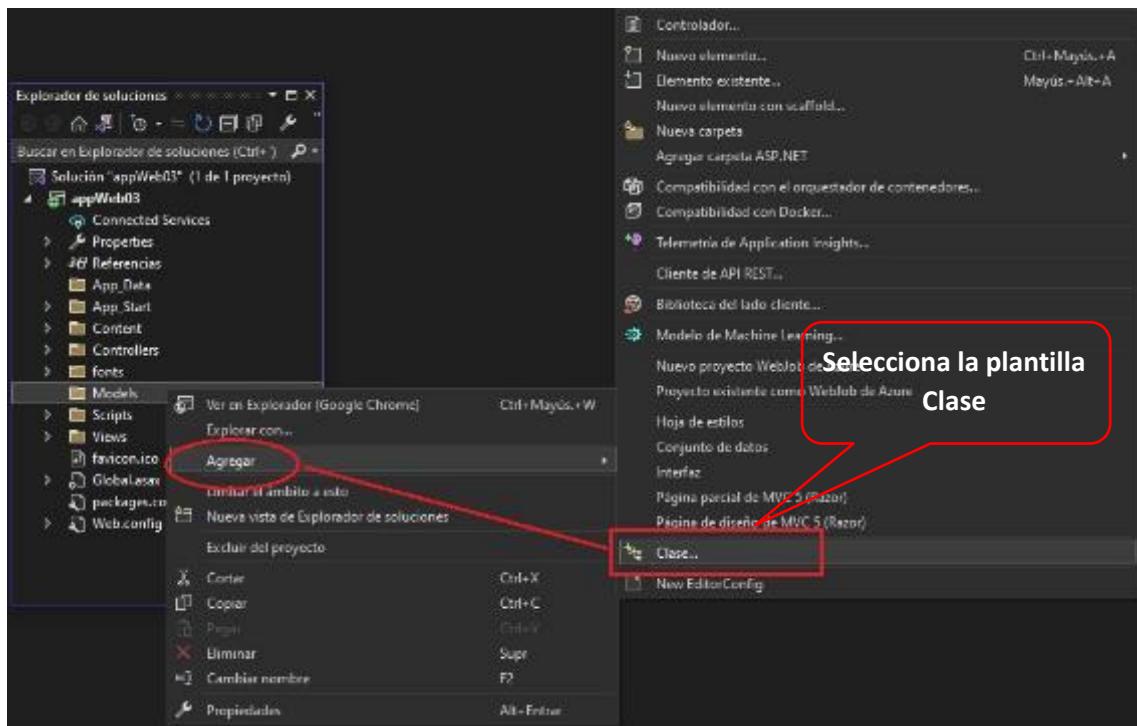
```

Nota. Elaboración propia.

Agregando la clase Producto a la carpeta Models

En la carpeta Models, hacer clic derecho sobre la carpeta, seleccionar Agregar → Clase, tal como se muestra en la figura.

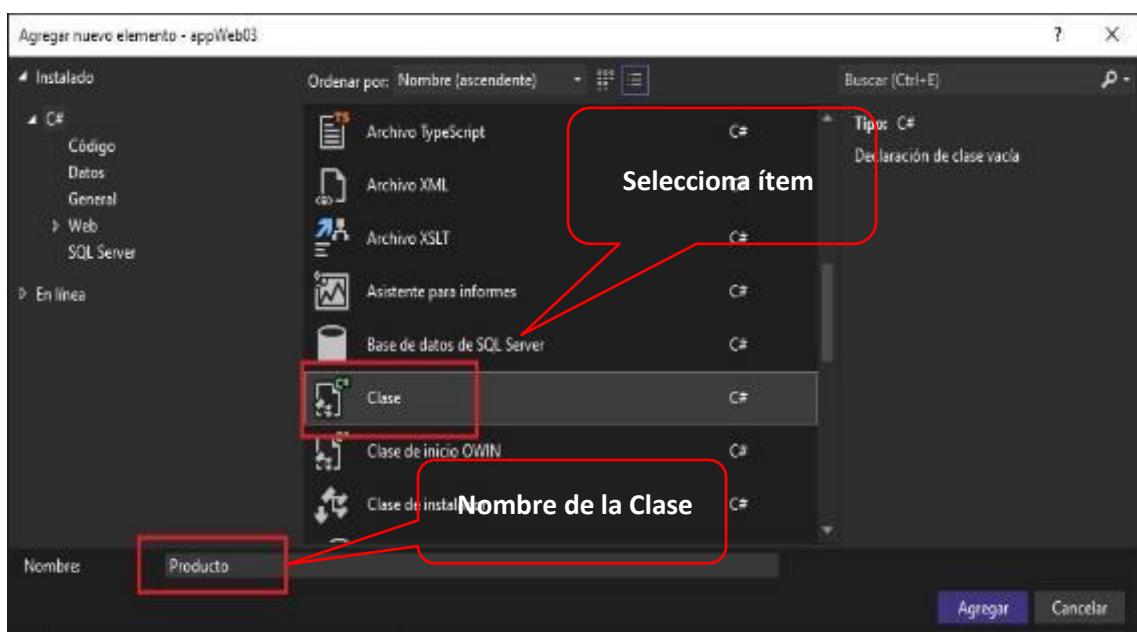
Figura 86
Desarrollo de Laboratorio



Nota. Elaboración propia.

Selecciona el elemento Clase, asigne el nombre del elemento: Producto, tal como se muestra.

Figura 87
Desarrollo de Laboratorio



Nota. Elaboración propia.

Agregar la librería System.ComponentModel.DataAnnotations y los atributos de la clase Producto, defina el atributo total el cual retorna el producto de precio por stock, tal como se muestra.

Figura 88

Desarrollo de Laboratorio

```

Product.cs  X
appWeb03          appWeb03.Models.Producto          iproducto
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.ComponentModel.DataAnnotations;
6  namespace appWeb03.Models
7  {
8      public class Producto
9      {
10         [Display(Name = "Id Producto")]
11         public int idproducto { get; set; }
12         [Display(Name = "Descripcion")]
13         public string descripcion { get; set; }
14         [Display(Name = "Categoria")]
15         public string categoria { get; set; }
16         [Display(Name = "Precio Unitario")]
17         public decimal precio { get; set; }
18         [Display(Name = "Unidades Disponibles")]
19         public Int16 stock { get; set; }
20         [Display(Name = "Total del Producto")]
21         public decimal total {
22             get { return precio * stock; } }
23     }
24 }

```

Nota. Elaboración propia.

Programando el método productos en el Controlador

1. En el Controlador Negocios, codifique el método productos()
2. Defina la conexión a través del SqlConnection, a través del bloque **using**.
3. Ejecute el procedure a través del SqlCommand, almacenando los resultados en el SqlDataReader.
4. 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.
5. Cerrar los objetos y enviar el objeto llamada temporal.

Figura 89*Desarrollo de Laboratorio*

```

NegociosController.cs 12
13
14     IEnumerable<Cliente> clientes();
15
16     IEnumerable<Producto> productos();
17
18     public class NegociosController : Controller
19     {
20
21         [HttpGet]
22         public ActionResult Index()
23         {
24             return View();
25         }
26
27         [HttpGet]
28         public ActionResult ListaClientes()
29         {
30             return View();
31         }
32
33         [HttpGet]
34         public ActionResult ListaProductos()
35         {
36             return View();
37         }
38
39         [HttpGet]
40         public ActionResult DetalleCliente(int id)
41         {
42             Cliente cliente = new Cliente();
43             cliente = clientes().Where(c => c.Id == id).FirstOrDefault();
44
45             if (cliente != null)
46             {
47                 return View(cliente);
48             }
49             else
50             {
51                 return RedirectToAction("Index");
52             }
53         }
54
55         [HttpGet]
56         public ActionResult DetalleProducto(int id)
57         {
58             Producto producto = new Producto();
59             producto = productos().Where(p => p.Id == id).FirstOrDefault();
60
61             if (producto != null)
62             {
63                 return View(producto);
64             }
65             else
66             {
67                 return RedirectToAction("Index");
68             }
69         }
70     }
71 
```

Nota. Elaboración propia.

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

Figura 90*Desarrollo de Laboratorio*

```

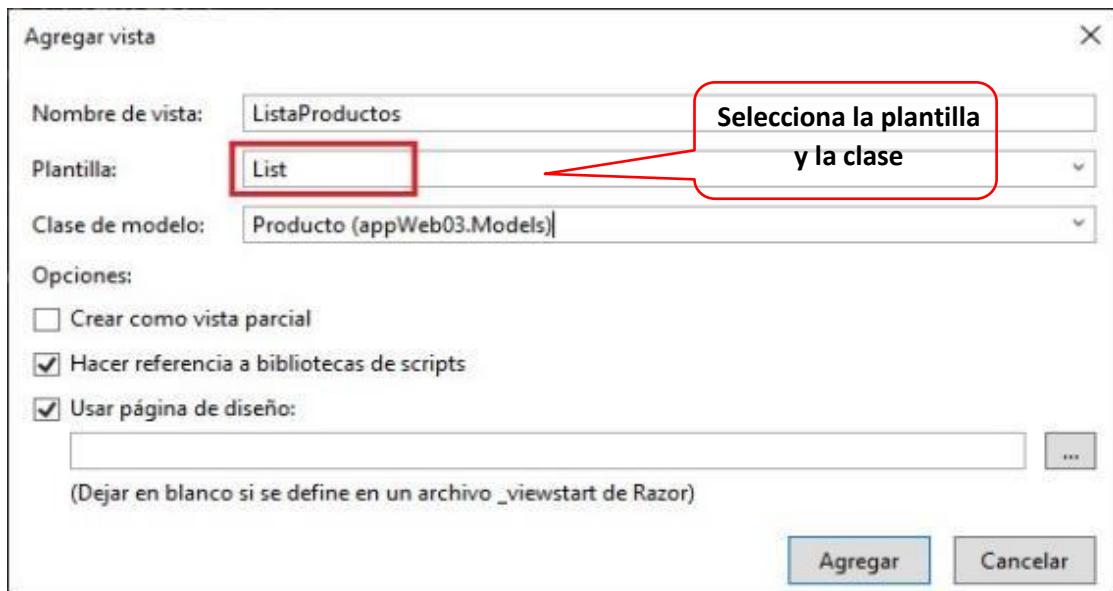
NegociosController.cs 12
13
14     IEnumerable<Cliente> clientes();
15
16     IEnumerable<Producto> productos();
17
18     public class NegociosController : Controller
19     {
20
21         [HttpGet]
22         public ActionResult Index()
23         {
24             return View();
25         }
26
27         [HttpGet]
28         public ActionResult ListaClientes()
29         {
30             return View();
31         }
32
33         [HttpGet]
34         public ActionResult ListaProductos()
35         {
36             return View();
37         }
38
39         [HttpGet]
40         public ActionResult DetalleCliente(int id)
41         {
42             Cliente cliente = new Cliente();
43             cliente = clientes().Where(c => c.Id == id).FirstOrDefault();
44
45             if (cliente != null)
46             {
47                 return View(cliente);
48             }
49             else
50             {
51                 return RedirectToAction("Index");
52             }
53         }
54
55         [HttpGet]
56         public ActionResult DetalleProducto(int id)
57         {
58             Producto producto = new Producto();
59             producto = productos().Where(p => p.Id == id).FirstOrDefault();
60
61             if (producto != null)
62             {
63                 return View(producto);
64             }
65             else
66             {
67                 return RedirectToAction("Index");
68             }
69         }
70     }
71 
```

Nota. Elaboración propia.

Trabajando con la Vista ListaProductos

- En el ActionResult, hacer clic derecho y selecciona, Agregar vista.
- En dicha ventana, seleccione la plantilla tipo **List**; y la clase de modelo **Producto**

Figura 91
Desarrollo de Laboratorio



Nota. Elaboración propia.

Vista generada por la plantilla List, agregar una fila donde totaliza (método Sum) el campo total.

Figura 92
Desarrollo de Laboratorio

```

@model IEnumerable<appWeb03.Models.Producto>



## Lista de Productos



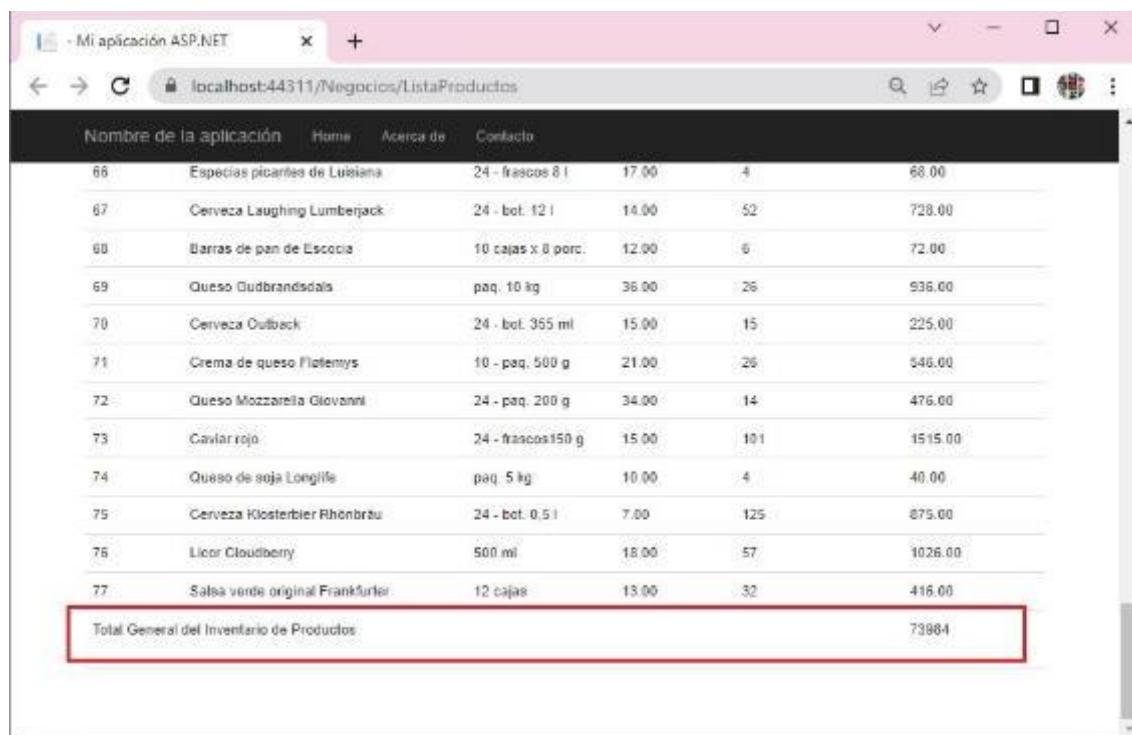
| @Html.DisplayNameFor(model => model.idproducto) | @Html.DisplayNameFor(model => model.descripcion) | @Html.DisplayNameFor(model => model.categoría) | @Html.DisplayNameFor(model => model.precio) | @Html.DisplayNameFor(model => model.stock) | @Html.DisplayNameFor(model => model.total) |
|-------------------------------------------------|--------------------------------------------------|------------------------------------------------|---------------------------------------------|--------------------------------------------|--------------------------------------------|
|                                                 |                                                  |                                                |                                             |                                            |                                            |


```

Nota. Elaboración propia.

Presiona la tecla F5 para ejecutar la Vista del Proyecto visualizando la lista de los registros de tb_productos, al final de los registros se imprime la suma de la columna Total.

Figura 93
Desarrollo de Laboratorio



The screenshot shows a Microsoft Edge browser window with the title "Mi aplicación ASP.NET" and the URL "localhost:44311/Negocios/ListaProductos". The page displays a table of product inventory data. The columns are: Nombre de la aplicación, Home, Acerca de, Contacto, ID, Nombre, Descripción, Precio, Stock, and Total. The last row of the table is highlighted with a red border and contains the text "Total General del Inventario de Productos" and the value "73984".

Nombre de la aplicación	Home	Acerca de	Contacto	ID	Nombre	Descripción	Precio	Stock	Total
66	Especies picantes de Luisiana	24 - frascos 8 l	17.00	4			68.00		
67	Cerveza Laughing Lumberjack	24 - bot. 12 l	14.00	52			728.00		
68	Barras de pan de Escocia	10 cajas x 8 porc.	12.00	6			72.00		
69	Queso Gudbrandsdals	paq. 10 kg	36.00	26			936.00		
70	Cerveza Outback	24 - bot. 355 ml	15.00	15			225.00		
71	Crema de queso Piolémys	10 - paq. 500 g	21.00	26			546.00		
72	Queso Mozzarella Giovanni	24 - paq. 200 g	34.00	14			476.00		
73	Gavilar rojo	24 - frascos 150 g	15.00	101			1515.00		
74	Queso de soja Longlife	paq. 5 kg	10.00	4			40.00		
75	Cerveza Klosterbier Rhenbräu	24 - bot. 0.5 l	7.00	125			875.00		
76	Licor Cloudberry	500 ml	18.00	57			1026.00		
77	Salsa verde original Frankfurter	12 cajas	13.00	32			416.00		
Total General del Inventario de Productos:									73984

Nota. Elaboración propia.

Resumen

1. 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.
2. Los dos componentes principales de ADO.NET para el acceso a datos y su manipulación son los proveedores de datos .NET Framework y DataSet.
3. Los proveedores de datos .NET Framework sirven para conectarse a una base de datos, ejecutar comandos y recuperar resultados. 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.
4. El proveedor de datos .NET Framework para SQL Server utiliza la librería System.Data.SqlClient.
5. Los principales componentes de un proveedor de datos .NET Framework son:
 - a. Connection
 - b. Command
 - c. DataReader
 - d. DataAdapter
6. El objeto DataSet es esencial para la compatibilidad con escenarios de datos distribuidos desconectados con ADO.NET. Representa un conjunto completo de datos que incluyen tablas relacionadas y restricciones; así como relaciones entre las tablas.
7. 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.
8. 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).
9. Debe cerrar siempre el objeto Connection cuando deje de usarlo. Esta operación se puede realizar mediante los métodos Close o Dispose del objeto Connection.

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

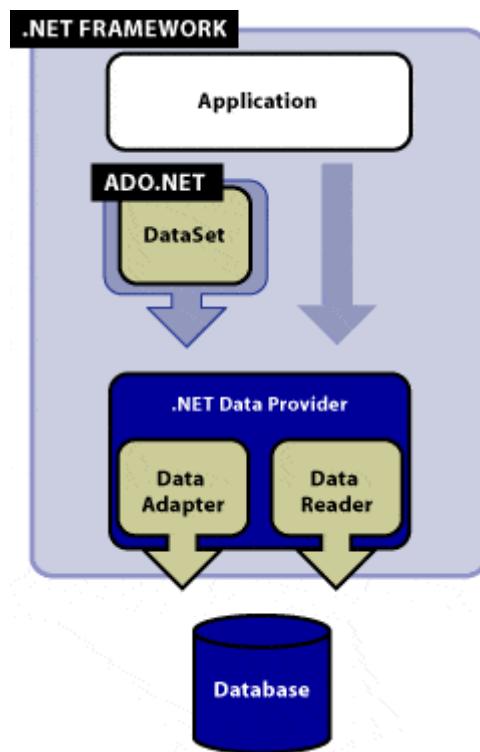
- <https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/ado-net-architecture?redirectedfrom=MSDN>
- <https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/connection-strings-and-configuration-files?redirectedfrom=MSDN>
- <https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/ado-net-architecture?redirectedfrom=MSDN>

2.2. RECUPERACIÓN DE DATOS Y PAGINACIÓN

La función principal de una aplicación que trabaje con un origen de datos es conectarse a dicha fuente de datos para realizar operaciones de consulta y actualización de los datos que se almacenan.

Los proveedores de .NET Framework para ADO.NET son utilizados por las aplicaciones para trabajar con un determinado origen de datos, permitiendo ejecutar instrucciones SQL para recuperar datos mediante el DataAdapter o el DataReader.

Figura 94
DataAdapter y DataReader



Nota. Tomado de *How does ado.net work*, por Progress, 2023, progress.com, (<https://www.progress.com/faqs/datadirect-ado-net-faqs/how-does-ado-net-work>)

2.2.1. Clase DataReader: métodos y propiedades

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 95
DataReader



Nota. Tomado de *Tổng Quan về ADO.NET*, por giakhoa.wordpress, 2011, (<https://giakhoa.wordpress.com/wp-content/uploads/2011/10/01-tong-quan-ado.pdf>)

Se 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:

Tabla 5
DataReader

Proveedor	Descripción
<code>OleDbDataReader</code>	Proveedor de datos para OLEDB
<code>SqlDataReader</code>	Proveedor de datos para SQL Server
<code>OdbcDataReader</code>	Proveedor de datos para ODBC
<code>OracleDataReader</code>	Proveedor de datos para Oracle

Nota. Adaptado de *Recuperación de datos con un objeto DataReader*, por Microsoft, 2023, (<https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/retrieving-data-using-a-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` llenado 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**.

2.2.2. Consulta de datos con parámetros utilizando DataReader

Cuando el DataReader ejecuta instrucciones de consulta con parámetros, se deben definir dentro del objeto Command, qué parámetros de entrada y de salida se deben crear. Para crear un parámetro, se utiliza el método: Parameters.Add() o Parameters.AddWithValue().

El método Parameters.Add() se debe especificar el nombre de columna: **Name**, tipo de datos: **SqlDbType**, tamaño: **Size** y asignando el valor del parámetro definido: **Value**. Por defecto la dirección del parámetro es de entrada **Input**, para indicar que es de salida debemos especificar en la propiedad **Direction**.

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

2.2.3. Trabajando con vistas parciales en una 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**.

Vistas parciales

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

Creando una vista parcial

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

Figura 96
Diseño de una Vista Parcial



Nota. Elaboración propia.

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

Para invocar a la vista parcial con los distintos datos:

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

Para invocar a una vista parcial, la puede 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.

Figura 97
Invoker una Vista Parcial

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

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

Nota. Elaboración propia.

2.2.4. Paginación de datos recuperados

La paginación en sitios web es una forma de estructurar el contenido, agrupándolo por una cantidad fija de elementos o registros. La paginación es simplemente el número de páginas que se muestran en la parte inferior de una página web que sirve para separar el contenido y facilitar la navegación.

Figura 98

Paginación



Nota. Tomado de *pagination*, por Sitechecker, 2020, sitechecker.pro (<https://sitechecker.pro/es/pagination/>)

A menudo es el elemento más olvidado en el diseño de páginas web, pero podría decirse que es uno de los más importantes.

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.

Cuando se trata de mostrar grandes cantidades de información en una web, ya sean productos de una tienda, resultados de una búsqueda o anuncios en una web de empleo, tradicionalmente se ha venido utilizando un recurso de interfaz conocido como paginación web. Esta técnica permite mantener el tamaño de cada página dentro de lo manejable en cuanto a navegación e interfaz. Al mismo tiempo, permite reducir el tamaño de la página la cantidad de información a transferir, ganando en velocidad.

Soluciones a nivel de concepto y de diseño sobre una paginación para una web concreta puede haber muchas. Este apunte pretende únicamente mostrar algunos ejemplos recogidos de ciertas webs: qué elementos son comunes y cuáles originales e interesantes. A partir de ahí, cada cual puede meditar la solución más adecuada para su web.

LABORATORIO 4.1.: Ejecutando consulta de datos con parámetro

Implemente un proyecto ASP.NET MVC, diseñe una Vista que permita listar los registros de la tabla tb_pedidoscabe de la base de datos Negocios2022 filtrando por un determinado año del campo FechaPedido ingresado como parámetro, ejecute el procedimiento almacenado usp_pedidos_year.

Creando el procedimiento almacenado

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

Figura 99
Procedimiento Almacenado

```

SQLQuery1.sql - L.MADIGA.dbo (36) < X
Use Negocios2022
go
create or alter proc usp_pedidos_year
@y int
As
Select idpedido, fechapedido,NombreCia,DireccionDestinatario,CiudadDestinatario
from tb_pedidoscab p join tb_clientes c on c.IdCliente=p.IdCliente
Where Year(FechaPedido)=@y
go

```

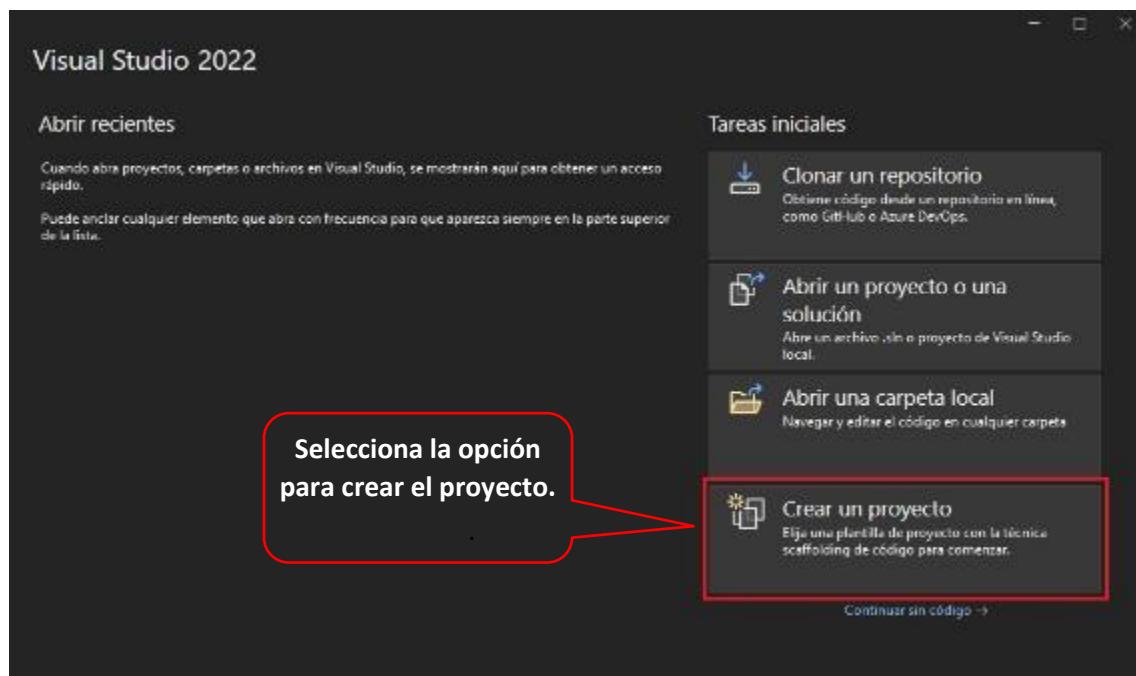
175 % 4 Query executed successfully. LARICP-DINAMICA\JULIAESPRESSO... LARICP-DINAMICA\camer... Negocios2022 Dbe3100 Dmowt

Nota. Elaboración propia.

Inicio del Proyecto

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

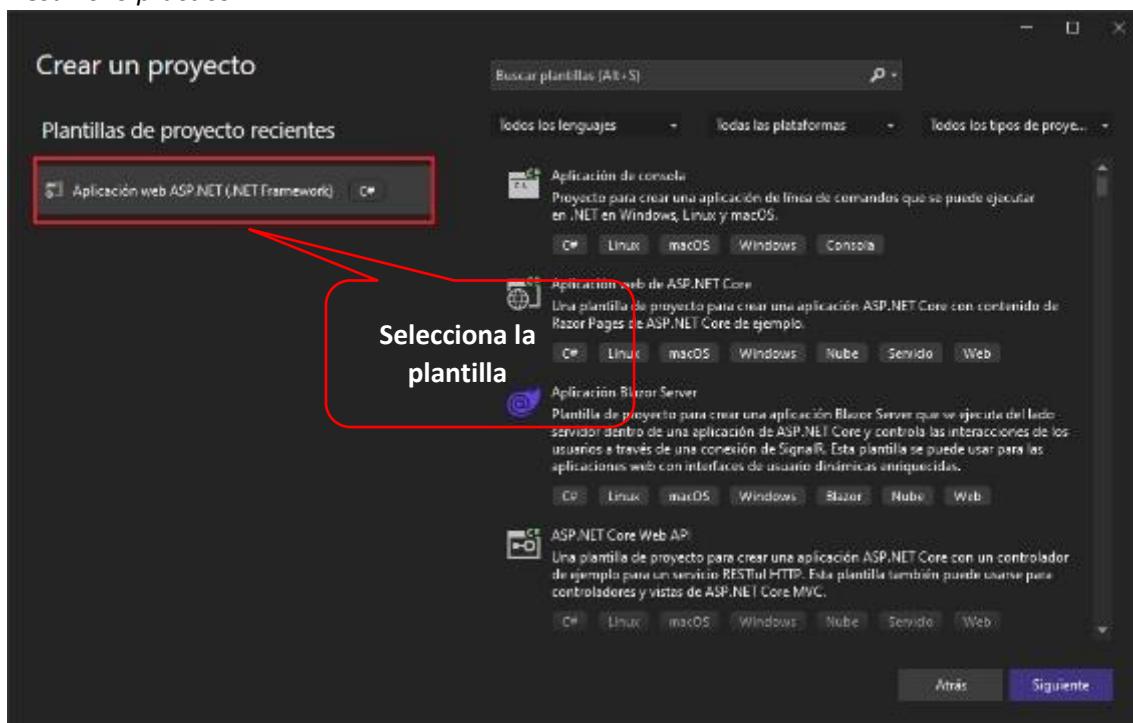
Figura 100
Desarrollo práctico



Nota. Elaboración propia.

Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción Siguiente.

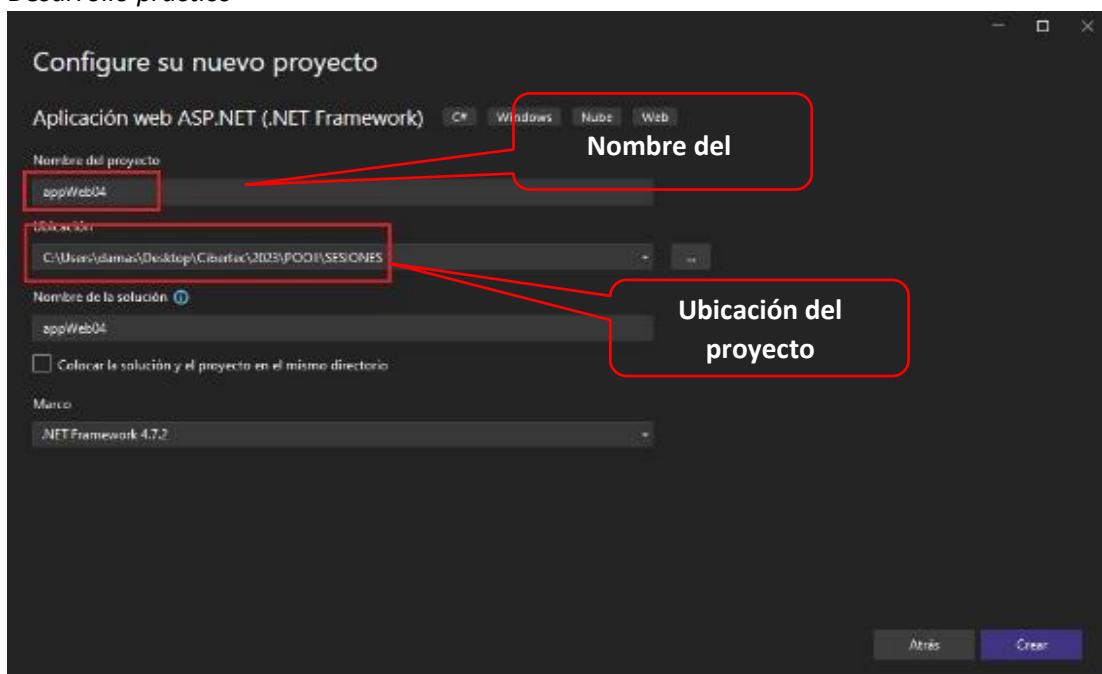
Figura 101
Desarrollo práctico



Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciónala ubicación del mismo, al terminar presiona la opción **Crear**.

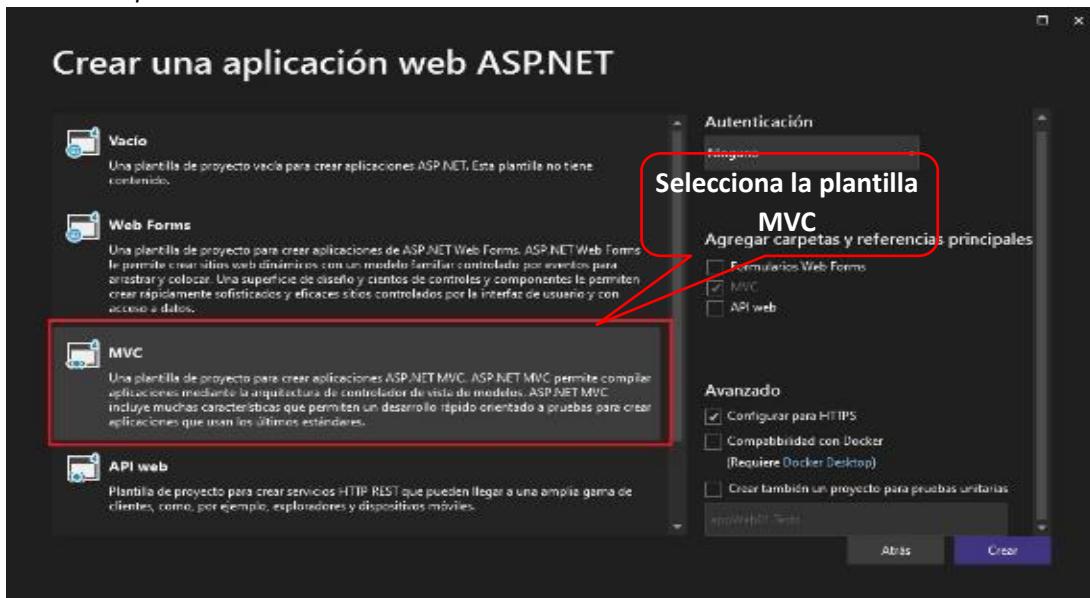
Figura 102
Desarrollo práctico



Nota. Elaboración propia.

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

Figura 103
Desarrollo práctico

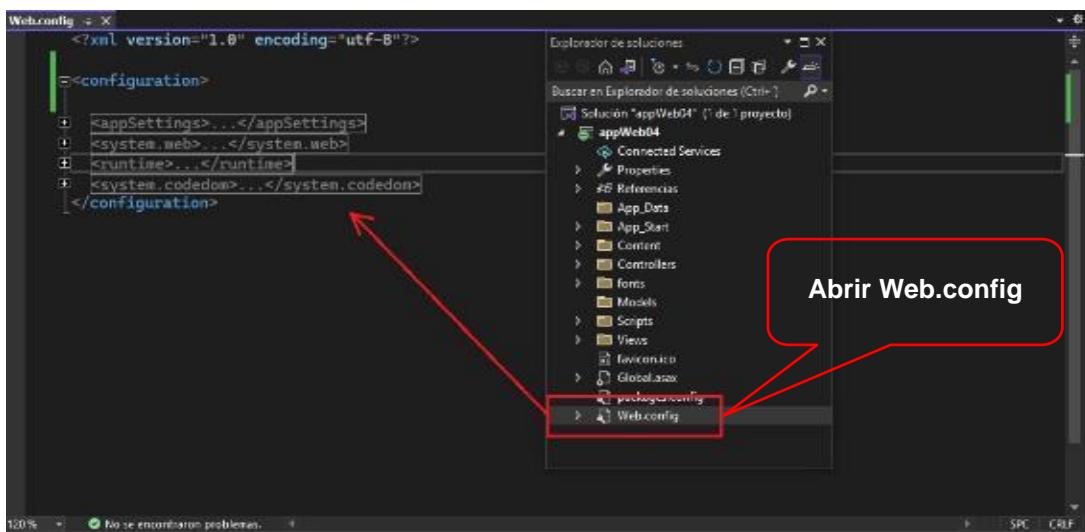


Nota. Elaboración propia.

Publicando la cadena de conexión

Desde el Explorador de proyecto, abrir el archivo Web.config, tal como se muestra.

Figura 104
Desarrollo práctico



Nota. Elaboración propia.

Defina la etiqueta <connectionStrings> agregando una cadena <add> cuyo nombre es “cadena” y definiendo la cadena de conexión, tal como se muestra.

Figura 105
Desarrollo práctico

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="cadena"
      connectionString="server=.; database=Negocios2022; integrated security=true"/>
  </connectionStrings>
  <appSettings>...</appSettings>
  <system.web>...</system.web>
  <runtime>...</runtime>
  <system.codedom>...</system.codedom>
</configuration>
```

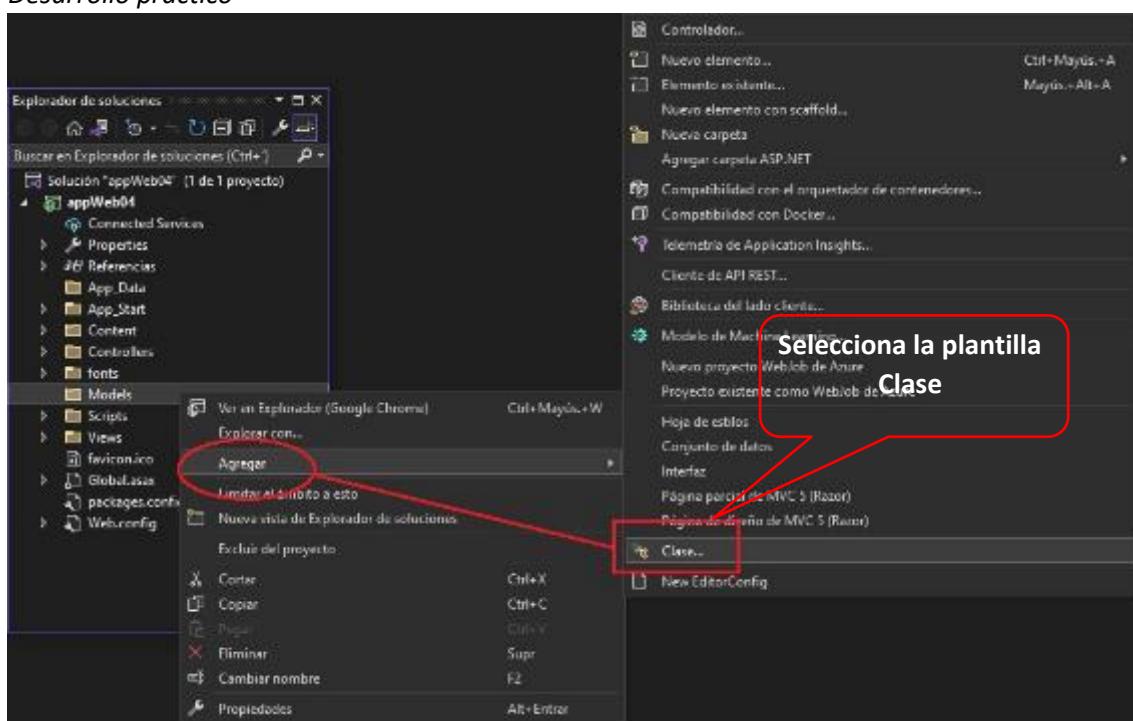
Definiendo la cadena de conexión

Nota. Elaboración propia.

Agregando la clase Pedido a la carpeta Models

En la carpeta Models, hacer clic derecho sobre la carpeta, seleccionar Agregar → Clase, tal como se muestra en la figura.

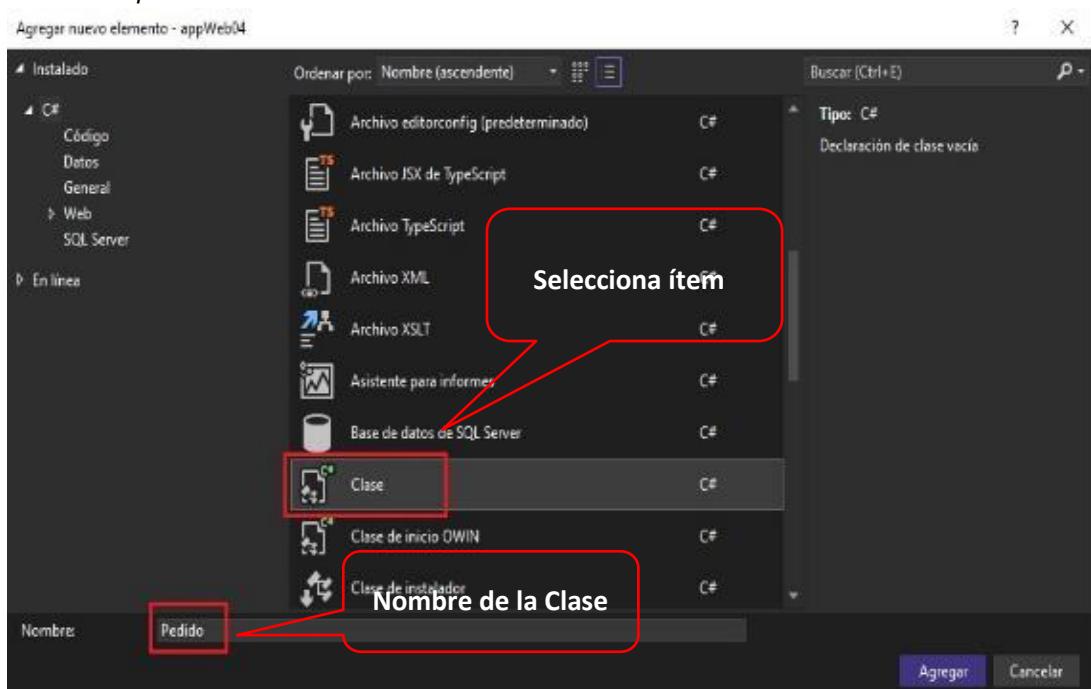
Figura 106
Desarrollo práctico



Nota. Elaboración propia.

Selecciona el elemento Clase, asigne el nombre del elemento: Pedido, tal como se muestra.

Figura 107
Desarrollo práctico



Nota. Elaboración propia.

Agregar la librería System.ComponentModel.DataAnnotations y los atributos de la clase Pedido, tal como se muestra.

Figura 108
Desarrollo práctico

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.ComponentModel.DataAnnotations;
6  namespace appWeb01.Models
7  {
8      public class Pedido
9      {
10         [Display(Name = "Id Pedido")]
11         public int idpedido { get; set; }
12         [Display(Name = "Fecha Pedido"), DisplayFormat(DataFormatString = "{0:dd/MM/yyyy}")]
13         public DateTime fecha { get; set; }
14         [Display(Name = "Cliente")]
15         public string cliente { get; set; }
16         [Display(Name = "Direccion Destino")]
17         public string direccion { get; set; }
18         [Display(Name = "Ciudad Destino")]
19         public string ciudad { get; set; }
20     }
21 }

```

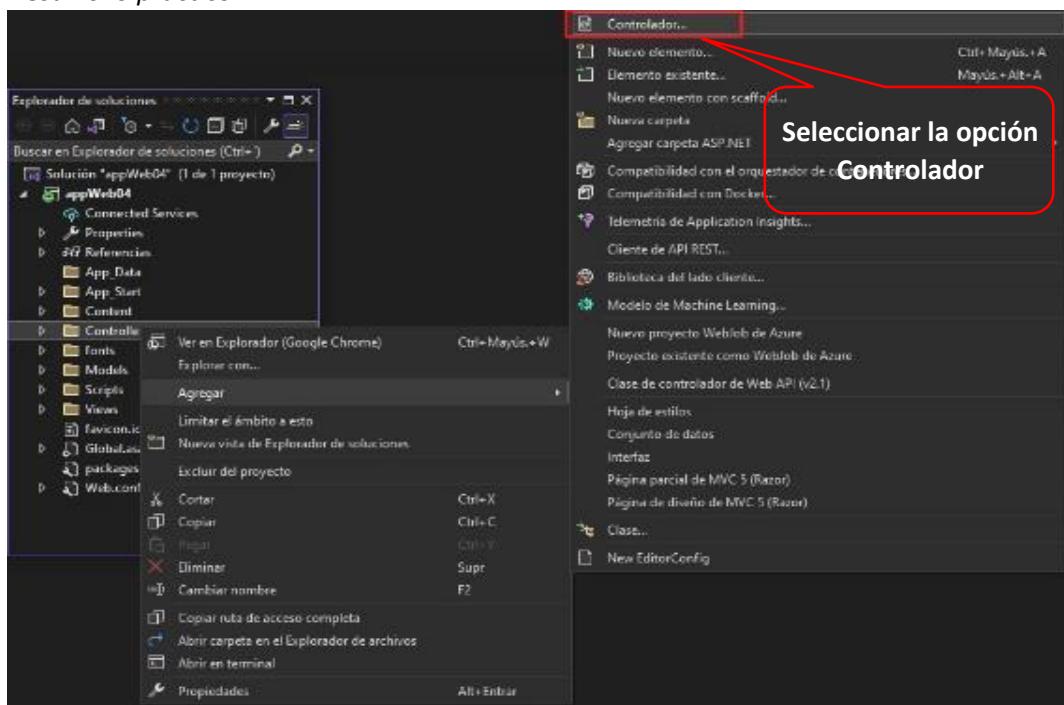
The screenshot shows the 'Pedido.cs' code editor in Visual Studio. The code defines a 'Pedido' class with several properties. A red box highlights the 'using System.ComponentModel.DataAnnotations;' directive. A callout bubble labeled 'Importar la librería para utilizar la etiqueta Display, DisplayFormat' points to this line.

Nota. Elaboración propia.

Trabajando con el Controlador

A continuación, en la carpeta Controllers, agregamos un controlador, tal como se muestra: Controllers → Agregar → Controlador

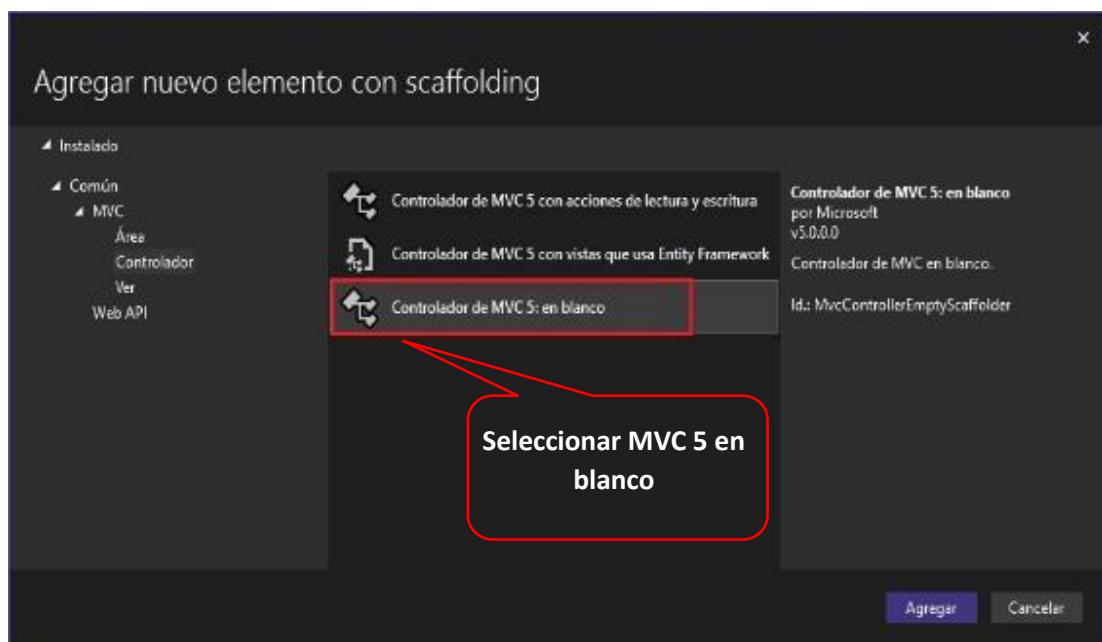
Figura 109
Desarrollo práctico



Nota. Elaboración propia.

Selecciona Controlador MVC5 en blanco, tal como se muestra.

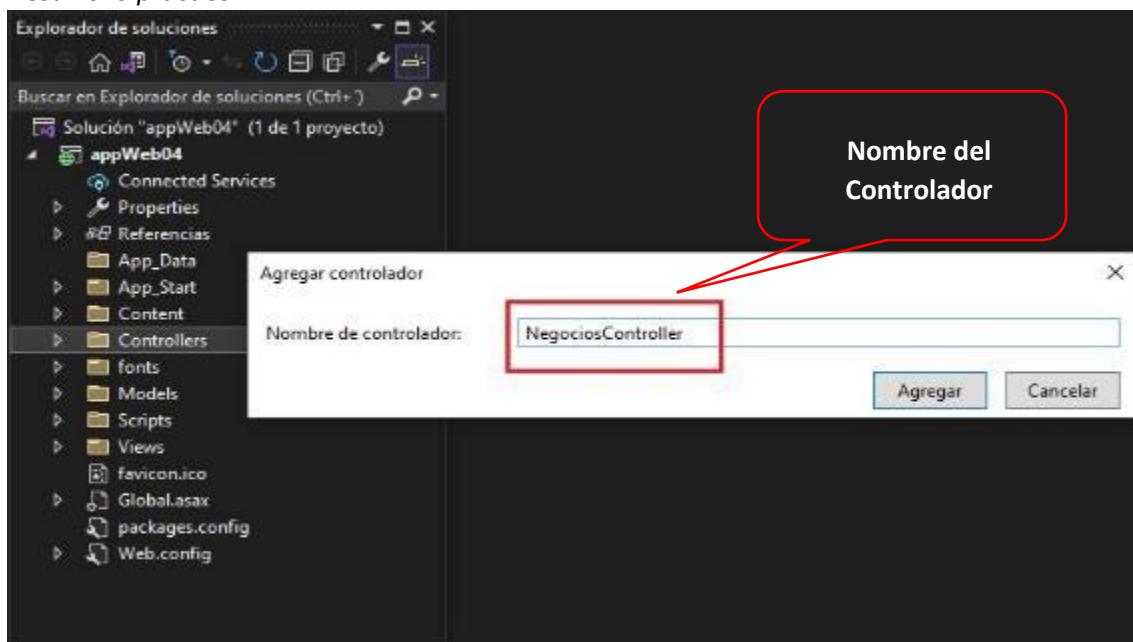
Figura 110
Desarrollo práctico



Nota. Elaboración propia.

Asigne el nombre de NegociosController, tal como se muestra. Presiona el botón Agregar.

Figura 11.1
Desarrollo práctico



Nota. Elaboración propia.

Programando el Controlador

Importar las librerías y la carpeta Models, donde se encuentra almacenado la clase.

Figura 11.2
Desarrollo práctico

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using System.Data;
7  using System.Data.SqlClient;
8  using System.Configuration;
9  using appWeb04.Models;
10 using appWeb04.Controllers;
11 {
12     public class NegociosController : Controller
13     {
14         public ActionResult Index()
15     }
16 }

```

The screenshot shows the code editor with the file 'NegociosController.cs' open. The code defines a controller 'NegociosController' that inherits from 'Controller'. It includes several 'using' statements at the top, such as 'System', 'System.Collections.Generic', 'System.Linq', 'System.Web', 'System.Web.Mvc', 'System.Data', 'System.Data.SqlClient', 'System.Configuration', and 'appWeb04.Models'. A red callout box with the text 'Referenciar las librerías y la carpeta Models' has an arrow pointing to the first few 'using' statements.

Nota. Elaboración propia.

En el Controlador, defina y codifique el método pedidosYear(int y) donde retorna la lista numerada de pedidos:

1. Defina la conexión a través del SqlConnection, a través del bloque **using**.

2. Ejecute el procedure a través del SqlCommand y agregue el valor al parámetro, almacenando los resultados en el SqlDataReader.
3. 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.
4. Cerrar los objetos y enviar el objeto llamada temporal.

Figura 113
Desarrollo práctico

```

12  public class NegociosController : Controller
13  {
14      IReferencia
15      IEnumerable<Pedido> pedidosYear(int y)
16      {
17          List<Pedido> temporal = new List<Pedido>();
18          using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
19          {
20              cn.Open();
21              SqlCommand cmd = new SqlCommand("exec usp_pedidos_year By", cn);
22              cmd.Parameters.AddWithValue("@y", y);
23              SqlDataReader dr = cmd.ExecuteReader();
24              while (dr.Read())
25              {
26                  temporal.Add(new Pedido()
27                  {
28                      idpedido = dr.GetInt32(0),
29                      fecha = dr.GetDateTime(1),
30                      cliente = dr.GetString(2),
31                      dirección = dr.GetString(3),
32                      ciudad = dr.GetString(4)
33                  });
34              }
35              dr.Close();
36          }
37          return temporal;
}

```

Nota. Elaboración propia.

Defina el ActionResult ConsultaPedidosYear(int y), el cual enviará a la Vista el resultado del método pedidosYear(y), tal como se muestra:

Figura 114
Desarrollo práctico

```

1  using [...]
2  namespace appWeb04.Controllers
3  {
4      IReferencia
5      public class NegociosController : Controller
6      {
7          IReferencia
8          IEnumerable<Pedido> pedidosYear(int y)...
9
10         IReferencia
11         ActionResult ConsultaPedidosYear(int y=0)
12         {
13             return View(pedidosYear(y));
14         }
15     }
16 }

```

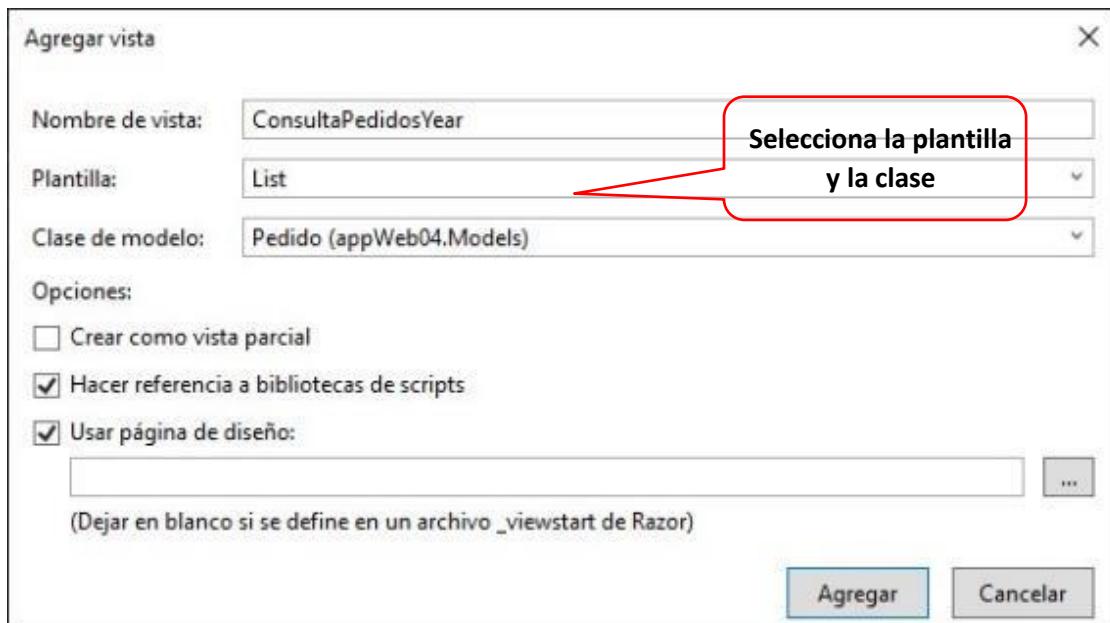
Nota. Elaboración propia.

Trabajando con la Vista ConsultaPedidosYear

- En el ActionResult, hacer clic derecho y selecciona, Agregar vista.
- En dicha ventana, seleccione la plantilla, la cual será List; y la clase de modelo la cual es Pedido, tal como se muestra.

Figura 115

Desarrollo de laboratorio



Nota. Elaboración propia.

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

Figura 116

Desarrollo práctico

```

@model IEnumerable<appWeb04.Models.Pedido>



## Consulta de Pedidos por Year



| @Html.DisplayNameFor(model => model.idpedido) | @Html.DisplayNameFor(model => model.fecha) | @Html.DisplayNameFor(model => model.cliente) | @Html.DisplayNameFor(model => model.direccion) | @Html.DisplayNameFor(model => model.ciudad) |
|-----------------------------------------------|--------------------------------------------|----------------------------------------------|------------------------------------------------|---------------------------------------------|
|-----------------------------------------------|--------------------------------------------|----------------------------------------------|------------------------------------------------|---------------------------------------------|


```

Nota. Elaboración propia.

Agregue un Formulario a la Vista, defina un helper TextBox para ingresar el valor al parámetro "y", defina el tipo a number, asigne valor min y max, tal como se muestra.

Figura 117

Desarrollo práctico

```

@model IEnumerable<appWeb04.Models.Pedido>


@using (Html.BeginForm())
    {
        <em>Ingrese el Año del Pedido:</em>
        @Html.TextBox("y", 0, new { type="number", min=1993, max=DateTime.Today.Year })
        <button>Consulta</button>
    }



|  |
|--|
|  |
|--|

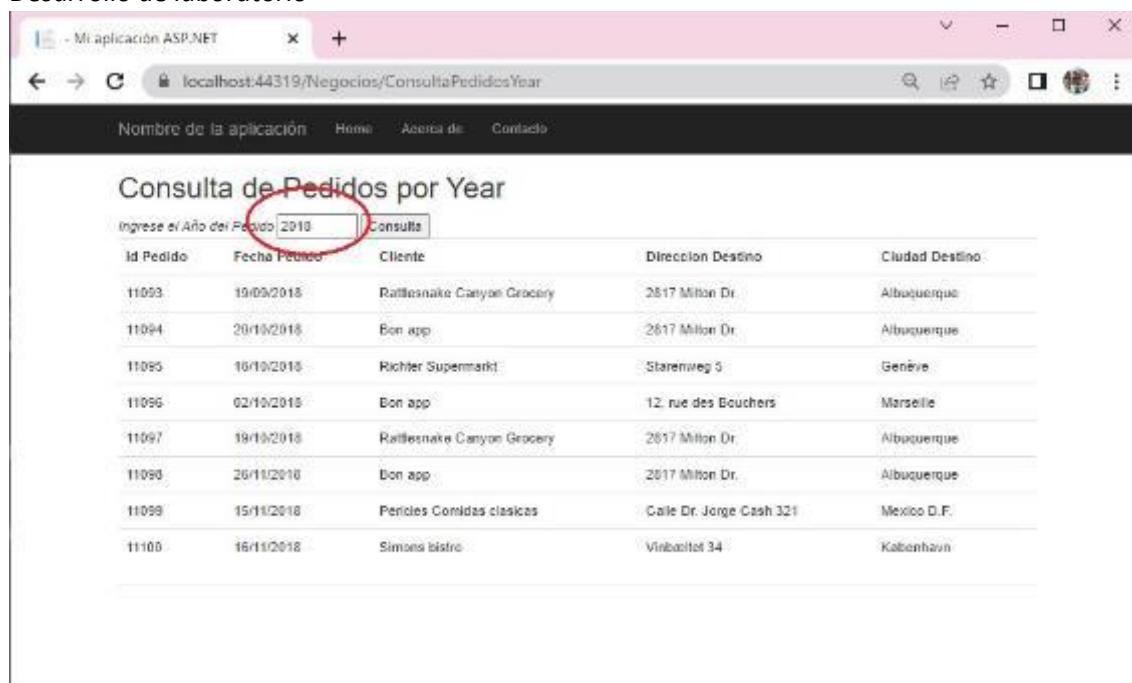

```

Nota. Elaboración propia.

Presiona la tecla F5 para ejecutar la Vista del Proyecto, ingrese el año en el TextBox, al presionar el botón Consulta visualizamos los pedidos registrados en ese año.

Figura 118

Desarrollo de laboratorio



Nota. Elaboración propia.

LABORATORIO 4.2.: Ejecutando consulta de datos con dos parámetros

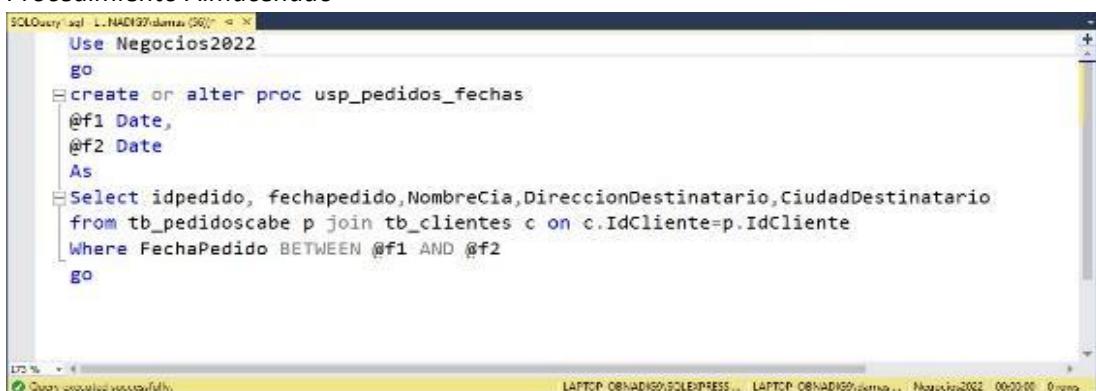
Implemente un proyecto ASP.NET MVC, diseñe una Vista que permita listar los registros de la tabla tb_pedidoscabe de la base de datos Negocios2022 filtrando entre dos fechas del campo FechaPedido ingresado como parámetro, ejecute el procedimiento almacenado usp_pedidos_fechas.

Creando el procedimiento almacenado

En el administrador del SQL Server, defina el procedimiento almacenado usp_pedidos_fechas, defina los parámetros @f1 y @f2 de tipo Date, tal como se muestra.

Figura 119

Procedimiento Almacenado



```

USE Negocios2022
go
create or alter proc usp_pedidos_fechas
@f1 Date,
@f2 Date
As
Select idpedido, fechapedido,NombreCia,DireccionDestinatario,CiudadDestinatario
from tb_pedidoscabe p join tb_clientes c on c.IdCliente=p.IdCliente
Where FechaPedido BETWEEN @f1 AND @f2
go

```

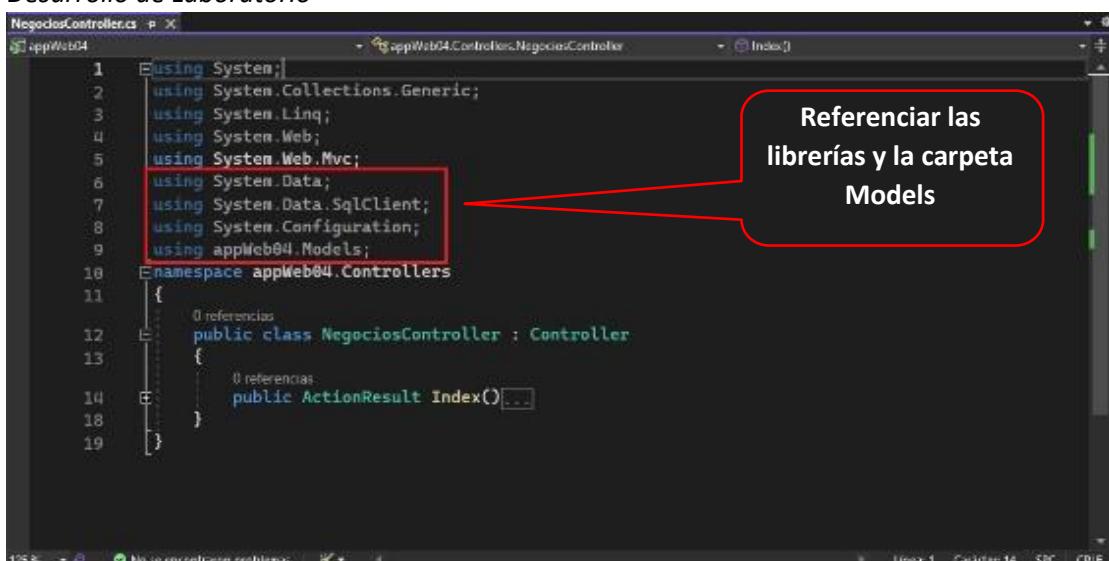
Nota. Elaboración propia.

Programando el Controlador NegociosController

En el Controlador creado en el laboratorio anterior, importar las librerías y la carpeta Models, donde se encuentra almacenado la clase.

Figura 120

Desarrollo de Laboratorio



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using System.Data;
7  using System.Data.SqlClient;
8  using System.Configuration;
9  using appWeb04.Models;
10 using appWeb04.Controllers
11 {
12     public class NegociosController : Controller
13     {
14         public ActionResult Index()
15     }
16 }

```

Referenciar las
librerías y la carpeta
Models

Nota. Elaboración propia.

En el Controlador, defina el método pedidosFechas con dos parámetros de tipo DateTime.

Figura 121

Desarrollo de Laboratorio

```

10     @namespace appWeb04.Controllers
11 {
12     public class NegociosController : Controller
13     {
14         [ referencia
15         IEnumerable<Pedido> pedidosYear(int y) ... ]
16         [ referencia
17         IEnumerable<Pedido> pedidosFechas(DateTime f1, DateTime f2)
18         {
19         }
20     }
21     [ referencia
22     ActionResult ConsultaPedidosYear(int y=0) ...
23     ]
24 }

```

Nota. Elaboración propia.

Codifique el método pedidosFechas donde retorna la lista numerada de pedidos:

1. Defina la conexión a través del SqlConnection, a través del bloque **using**.
2. Ejecute el procedure a través del SqlCommand y agregue el valor a los parámetros, almacenando los resultados en el SqlDataReader.
3. 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.
4. Cerrar los objetos y enviar el objeto llamada temporal.

Figura 122

Desarrollo de Laboratorio

```

14     IEnumerable<Pedido> pedidosYear(int y) ...
15
16     IEnumerable<Pedido> pedidosFechas(DateTime f1, DateTime f2)
17     {
18         List<Pedido> temporal = new List<Pedido>();
19         using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
20         {
21             cn.Open();
22             SqlCommand cmd = new SqlCommand("exec usp_pedidos_fechas @f1,@f2", cn);
23             cmd.Parameters.AddWithValue("@f1", f1);
24             cmd.Parameters.AddWithValue("@f2", f2);
25             SqlDataReader dr = cmd.ExecuteReader();
26             while (dr.Read())
27             {
28                 temporal.Add(new Pedido()
29                 {
30                     idpedido = dr.GetInt32(0),
31                     fecha = dr.GetDateTime(1),
32                     cliente = dr.GetString(2),
33                     direccion = dr.GetString(3),
34                     ciudad = dr.GetString(4)
35                 });
36             }
37             dr.Close();
38         }
39         return temporal;
40     }

```

Nota. Elaboración propia.

Defina el ActionResult ConsultaPedidosFechas(DateTime f1, DateTime f2), el cual enviará a la Vista el resultado del método pedidosFechas(f1,f2), tal como se muestra.

Figura 123
Desarrollo de Laboratorio

```

12     public class NegociosController : Controller
13     {
14         IReferencia
15         IEnumerable<Pedido> pedidosYear(int y=0);
16
17         IReferencia
18         IEnumerable<Pedido> pedidosFechas(DateTime? f1, DateTime? f2);
19
20         public ActionResult ConsultaPedidosYear(int y=0)
21         {
22             return View(pedidosYear(y));
23         }
24
25         public ActionResult ConsultaPedidosFechas(DateTime? f1=null, DateTime? f2=null)
26         {
27             DateTime _f1 = f1 == null ? DateTime.Today : (DateTime)f1;
28             DateTime _f2 = f2 == null ? DateTime.Today : (DateTime)f2;
29
30             return View(pedidosFechas(_f1,_f2));
31         }
32     }

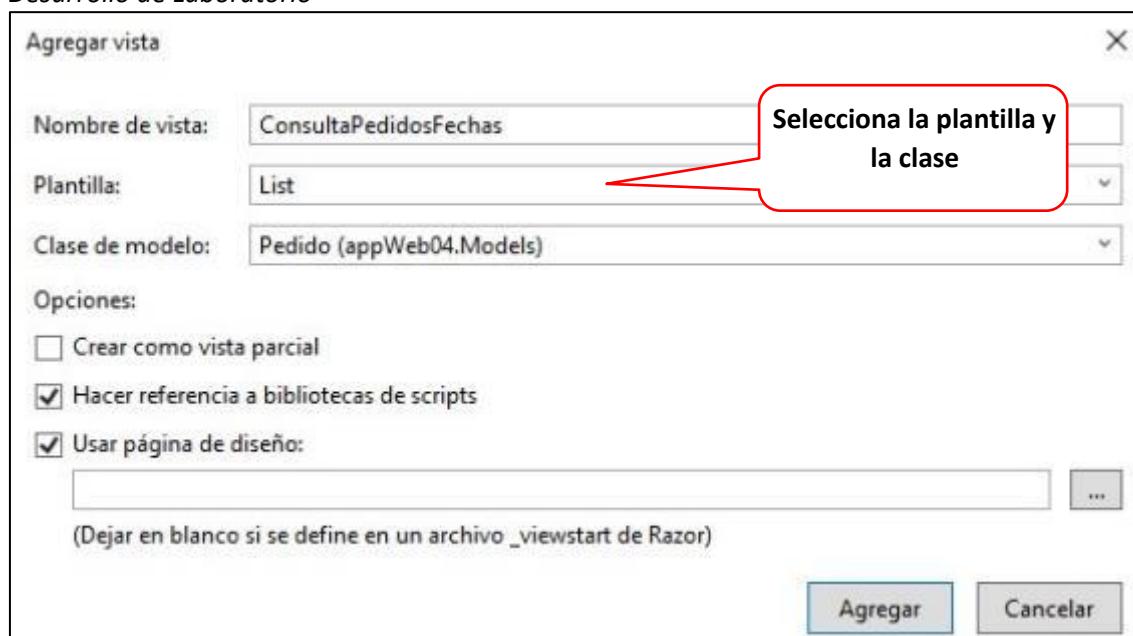
```

Nota. Elaboración propia.

Trabajando con la Vista ConsultaPedidosFechas

- En el ActionResult, hacer clic derecho y selecciona, Agregar vista.
- En dicha ventana, seleccione la plantilla, la cual será List; y la clase de modelo la cual es Pedido, tal como se muestra.

Figura 124
Desarrollo de Laboratorio



Nota. Elaboración propia.

Agregue un Formulario a la Vista, defina dos helper TextBox para ingresar el valor a los parámetros “f1” y “f2” de tipo date, tal como se muestra.

Figura 125

Desarrollo de Laboratorio

```

@model IEnumerable<appWeb04.Models.Pedido>

ViewBag.Title = "ConsultaPedidosFechas";



## Consulta de Pedidos entre dos Fechas



Using (Html.BeginForm())
    {
        <en>Desde:</en>
        @Html.TextBox("f1", "", new { type = "date" })
        <en>Hasta:</en>
        @Html.TextBox("f2", "", new { type = "date" })
        <button>Consulta</button>
    }



|           |
|-----------|
| <tr></tr> |
|-----------|

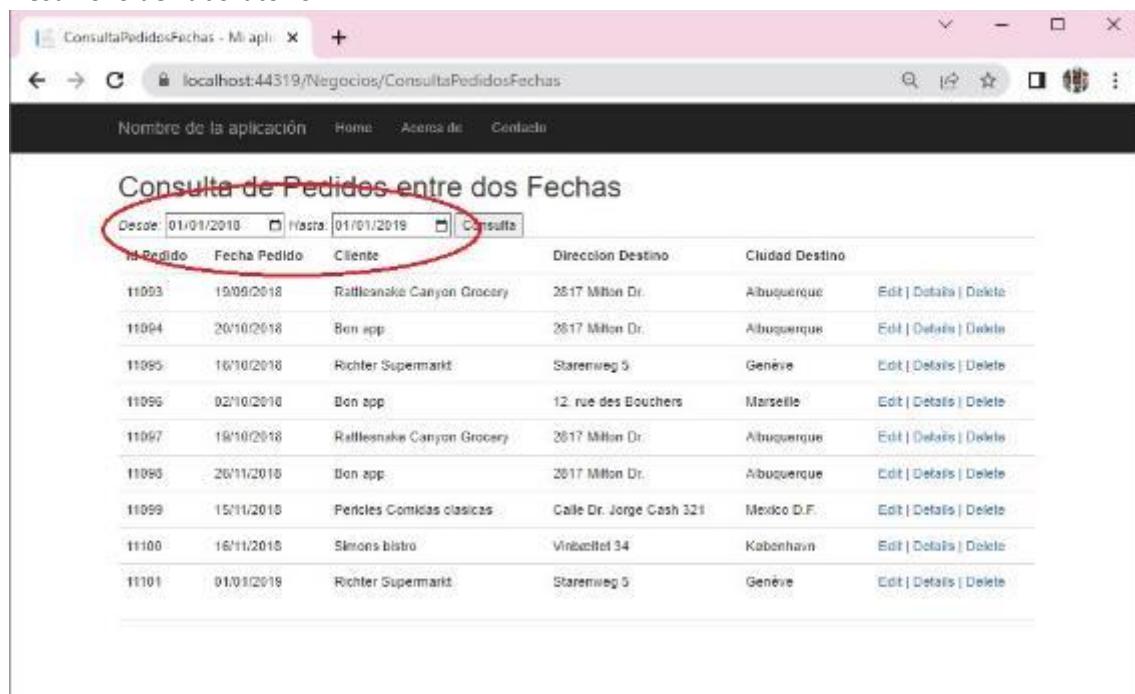

```

Nota. Elaboración propia.

Presiona la tecla F5 para ejecutar la Vista del Proyecto, selecciona las fechas en cada TextBox, al presionar el botón Consulta visualizamos los pedidos registrados entre las fechas seleccionadas.

Figura 126

Desarrollo de Laboratorio



Nota. Elaboración propia.

LABORATORIO 4.3.: Paginación de los registros recuperados

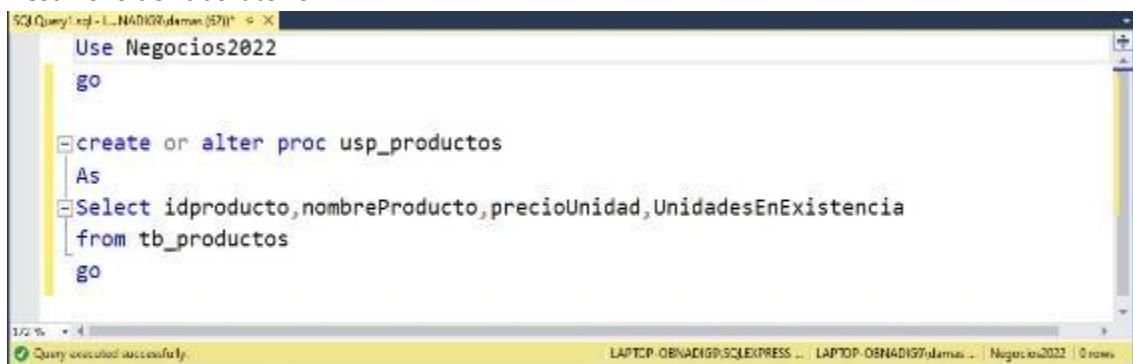
Implemente un proyecto ASP.NET MVC, diseñe una página que permita listar y paginate los registros de la tabla tb_productos de la base de datos Negocios2022 y ejecute el procedimiento almacenado usp_productos.

Creando el Procedimiento Almacenado en el Sql Server

En el manejador del SQL Server, creamos el procedimiento almacenado usp_productos.

Figura 127

Desarrollo de Laboratorio



```
Use Negocios2022
go

create or alter proc usp_productos
As
Select idproducto,nombreProducto,precioUnidad,UnidadesEnExistencia
from tb_productos
go
```

Query executed successfully.

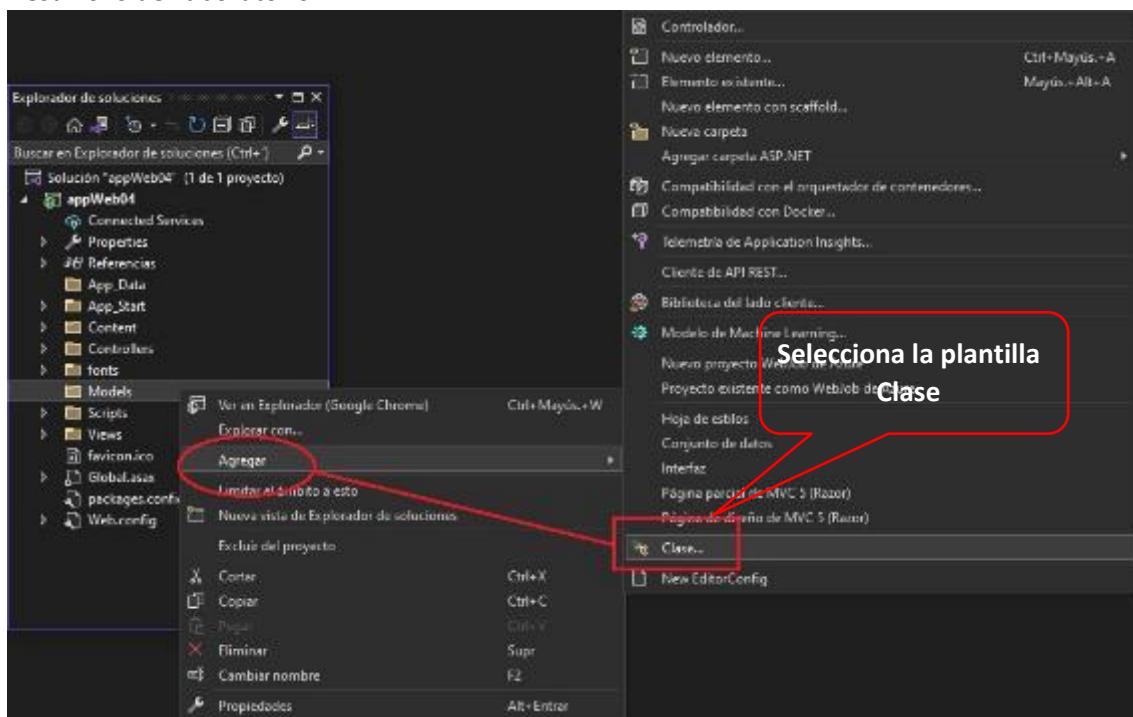
Nota. Elaboración propia.

Agregando la clase Producto a la carpeta Models

En la carpeta Models, hacer clic derecho sobre la carpeta, seleccionar Agregar → Clase, tal como se muestra en la figura.

Figura 128

Desarrollo de Laboratorio

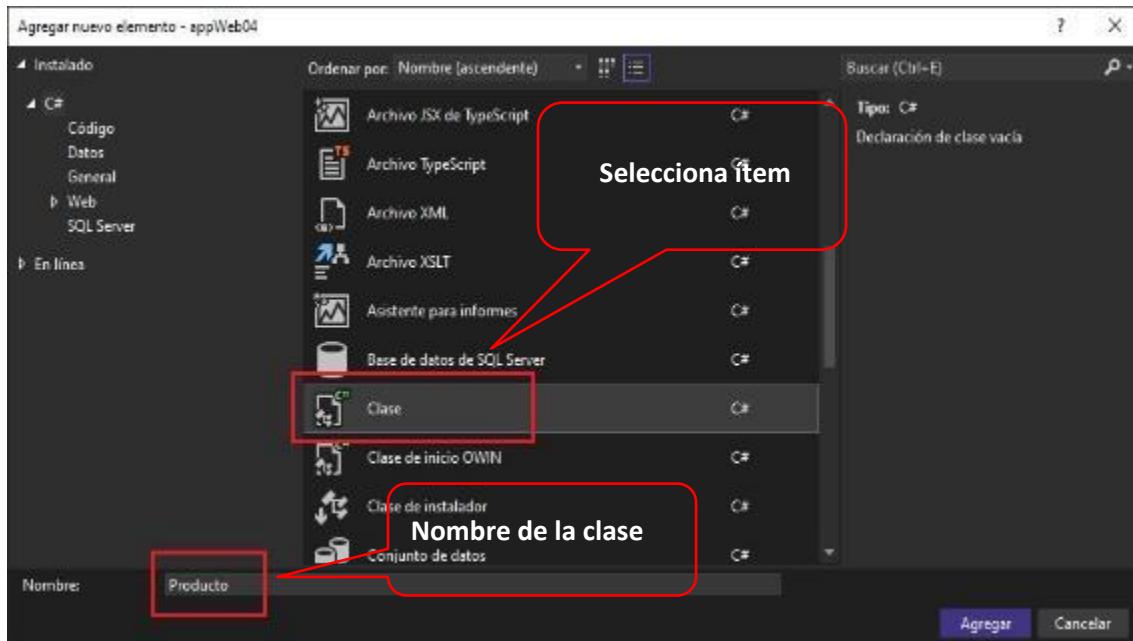


Nota. Elaboración propia.

Selecciona el elemento Clase, asigne el nombre del elemento: Producto, tal como se muestra.

Figura 129

Desarrollo de Laboratorio



Nota. Elaboración propia.

Agregar la librería System.ComponentModel.DataAnnotations y los atributos de la clase Producto, tal como se muestra.

Figura 130

Desarrollo de Laboratorio

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.ComponentModel.DataAnnotations;
6  namespace appWeb04.Models
7  {
8      public class Producto
9      {
10         [Display(Name = "Id Producto")]
11         public int idproducto { get; set; }
12         [Display(Name = "Descripción Producto")]
13         public string descripcion { get; set; }
14         [Display(Name = "Precio Unitario")]
15         public decimal preuni { get; set; }
16         [Display(Name = "Unidades Disponibles")]
17         public Int16 stock { get; set; }
18         [Display(Name = "SubTotal")]
19         public decimal subtotal { get { return preuni * stock; } }
20     }
21 }

```

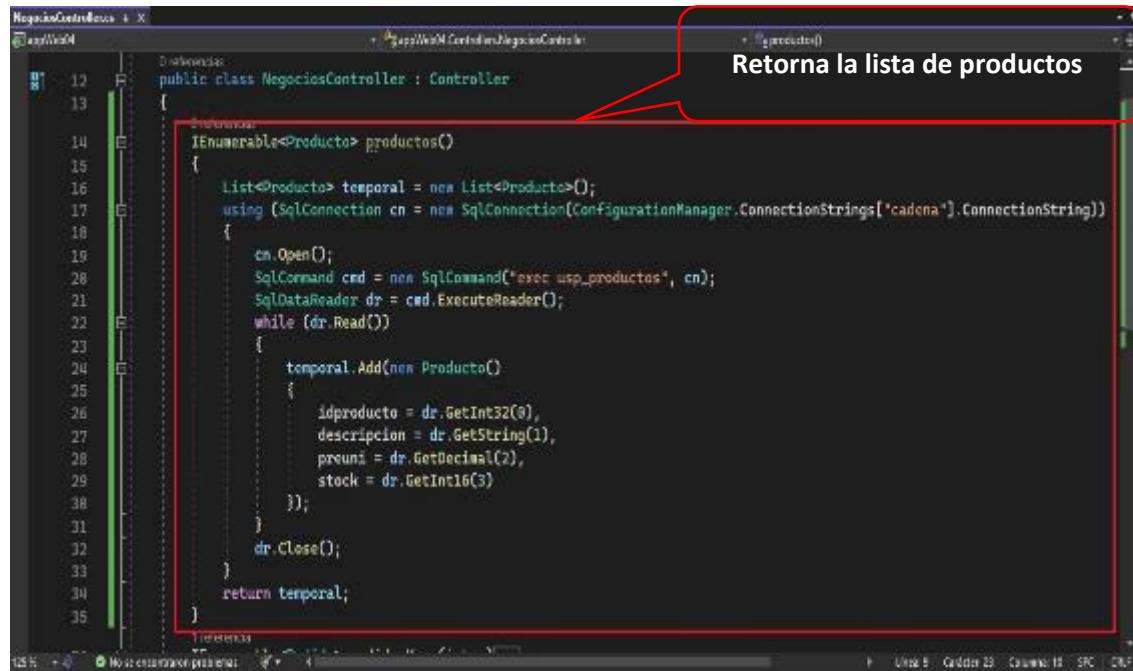
Nota. Elaboración propia.

Trabajando con el Controlador

En el proyecto desarrollado en el laboratorio anterior, agregue un método de tipo IEnumerable llamado productos() donde retorna los registros de la tabla tb_productos.

Figura 131

Desarrollo de Laboratorio



```

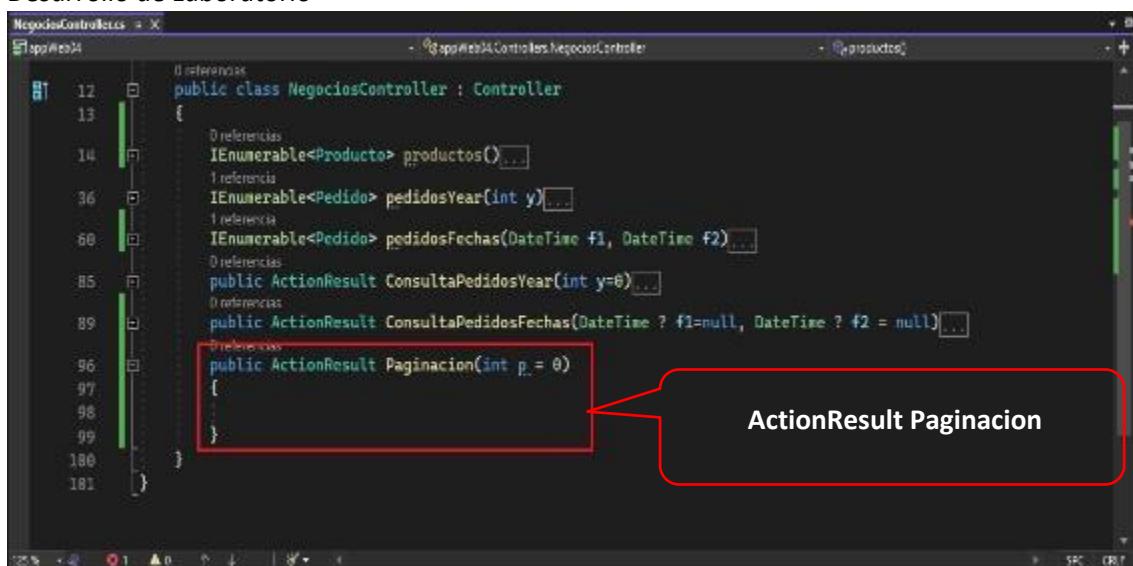
NegociosController.cs + X
@aplicativ24
D:\referencias
public class NegociosController : Controller
{
    // Consultas
    IEnumerable<Producto> productos()
    {
        List<Producto> temporal = new List<Producto>();
        using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
        {
            cn.Open();
            SqlCommand cmd = new SqlCommand("exec spc_productos", cn);
            SqlDataReader dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                temporal.Add(new Producto()
                {
                    idproducto = dr.GetInt32(0),
                    descripción = dr.GetString(1),
                    preuni = dr.GetDecimal(2),
                    stock = dr.GetInt16(3)
                });
            }
            dr.Close();
        }
        return temporal;
    }
}
  
```

Nota. Elaboración propia.

A continuación, agregamos un ActionResult llamado Paginacion() definiendo un parámetro llamado p (tipo entero) el cual representa el número de la página.

Figura 132

Desarrollo de Laboratorio



```

NegociosController.cs + X
@aplicativ24
D:\referencias
public class NegociosController : Controller
{
    // Consultas
    IEnumerable<Producto> productos()...
    IEnumerable<Pedido> pedidosYear(int y)... 
    IEnumerable<Pedido> pedidosFechas(DateTime f1, DateTime f2)...
    public ActionResult ConsultaPedidosYear(int y=0)...
    public ActionResult ConsultaPedidosFechas(DateTime ? f1=null, DateTime ? f2 = null)...
    // Nuevas
    public ActionResult Paginacion(int p = 0)
    {
    }
}
  
```

Nota. Elaboración propia.

En el ActionResult, defina la variable “c” que almacena la cantidad de productos; la variable “f” se le asigna 15 (15 registros por página), la variable “npags”, que representa la cantidad de páginas.

Figura 133
Desarrollo de Laboratorio

```

NegociosController.cs  ~ X
appWeb04              .\appWeb04.Controllers.NegociosController      @_ productos()
12
13
14     1 referencia
15         IEnumarable<Producto> productos();
16
17     0 referencias
18     public ActionResult Paginacion(int p = 0)
19     {
20         int c = productos().Count();
21         int f = 15;
22
23         int npags = c % f == 0 ? c / f : c / f + 1;
24     }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44

```

Nota. Elaboración propia.

A continuación, definimos 2 ViewBag; los cuales almacena el número de página (ViewBag.p) y la cantidad de páginas (ViewBag.npags). El ActionResult retorna la cantidad de registros según la página seleccionada.

Figura 134
Desarrollo de Laboratorio

```

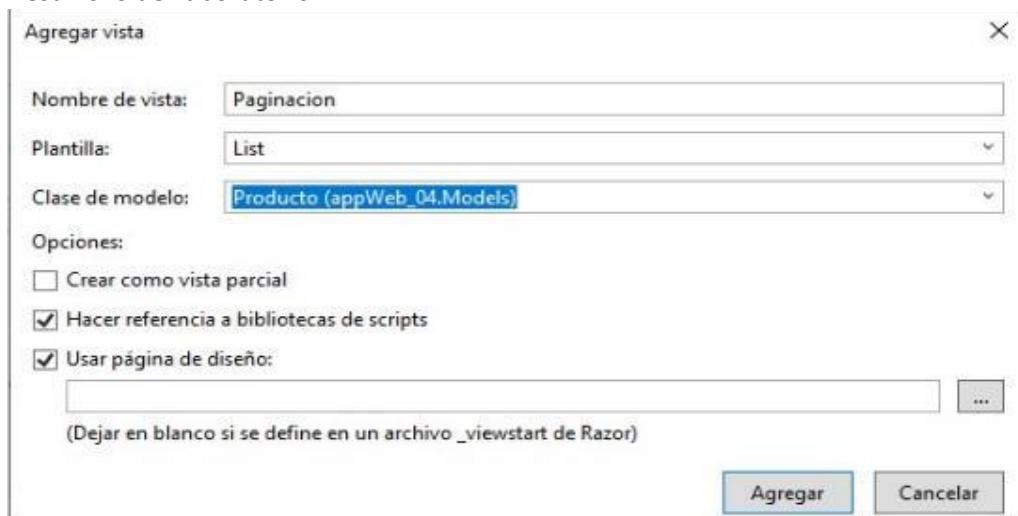
NegociosController.cs  ~ X
appWeb04              .\appWeb04.Controllers.NegociosController      @_ productos()
12
13
14     2 referencias
15         IEnumarable<Producto> productos();
16
17     0 referencias
18     public ActionResult Paginacion(int p = 0)
19     {
20         int c = productos().Count();
21         int f = 15;
22
23         int npags = c % f == 0 ? c / f : c / f + 1;
24
25         ViewBag.p = p;
26         ViewBag.npags=npags;
27
28         return View(productos().Skip(f*p).Take(f));
29     }
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

Envía a la vista los registros de la paginación.

Nota. Elaboración propia.

Agregar la vista de Paginacion: lista de Producto.

Figura 135*Desarrollo de Laboratorio**Nota. Elaboración propia.*

En la vista, agregamos un bloque <div>, donde a través de un for, vamos a imprimir ActionLink los cuales representan al número de página a seleccionar y visualizar. En el ActionLink se coloca: título, controlador, variable p con el valor de i, y el diseño de class

Figura 136*Desarrollo de Laboratorio*

```

@model IEnumerable<appWeb04.Models.Producto>

@{
    ViewBag.Title = "Paginacion";
}



## Paginacion de Productos



<table class="table">
    <tr></tr>
    @foreach (var item in Model)
    {
        <tr>
            <td>@item.Nombre </td>
            <td>@item.Cantidad </td>
            <td>@item.Precio </td>
            <td>
                <a href="#">Edit</a>
                <a href="#">Details</a>
                <a href="#">Delete</a>
            </td>
        </tr>
    }
</table>


```

```

<div>
    <for(int i=0; i<(int)ViewBag.npages; i++)>
        <div>
            <a href="#">@i.ToString()</a>
        </div>
    </for>
</div>
```

Nota. Elaboración propia.

Presiona la tecla F5, donde la vista visualiza la primera página. Al hacer clic en un Action (la cual se representa por la numeración que está en la parte inferior) visualiza los registros de la página seleccionada.

Figura 137
Desarrollo de Laboratorio

Paginacion de Productos				
Creado por:				
ID Producto	Descripción Producto	Precio Unitario	Unidades Disponibles	SubTotal
16	Pasta de manzana Pavlova	17.00	29	493.00
17	Galletas Alice Borlina	30.00	0	0.00
18	Lenguitas Agre Colavita	62.00	42	2394.00
19	Pasta de té de chocolate	9.00	25	225.00
20	Viennetta de Sir Rodney's	81.00	40	3240.00
21	Bollos de Sir Rodney's	10.00	40	400.00
22	Pan de canela con piña en la Glastate	21.00	164	3494.00
23	Pan Fre	9.00	81	549.00
24	Rahmboos Galletas Mandarinas	4.20	20	83.00
25	Crema de chocolate y naranja NaNigie	14.00	18	184.00
26	Dulces de goma Gumbé	21.00	15	315.00
27	Chocolate Schogg	45.00	48	2160.00
28	Col fermentada Riesle	45.00	26	1170.00
29	Bolachas Thuringer	121.00	0	0.00
30	Ungüento de coco del nordeste	25.00	16	250.00

Nota. Elaboración propia.

LABORATORIO 4.4.: Búsqueda y paginación de los registros recuperados

Implemente una Vista en un proyecto ASP.NET MVC, que permita listar y paginar los registros de la tabla tb_pedidoscabe de la base de datos Negocios2022 filtrando por un determinado año del campo FechaPedido, defina el procedimiento almacenado usp_pedidos_Year para implementar el proceso

Creando el procedimiento almacenado

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

Figura 138
Procedimiento Almacenado

```

Use Negocios2022
go
create or alter proc usp_pedidos_year
@y int
As
Select idpedido, fechapedido, NombreCia, DireccionDestinatario, CiudadDestinatario
from tb_pedidoscabe p join tb_clientes c on c.IdCliente=p.IdCliente
Where Year(FechaPedido)=@y
go

```

Nota. Elaboración propia.

Trabajando con el Controlador

En el Controlador hemos creado el método pedidosYear(int y) donde retorna la lista numerada de pedidos por un determinado año del campo fechapedido, revisar el laboratorio 4.1.

Figura 139

Desarrollo de Laboratorio

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using appWeb04.Models;
using System.Data.SqlClient;

namespace appWeb04.Controllers
{
    public class NegociosController : Controller
    {
        // GET: Negocios
        public ActionResult Index()
        {
            return View();
        }

        public IEnumerable<Pedido> pedidosYear(int y)
        {
            List<Pedido> temporal = new List<Pedido>();
            using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
            {
                cn.Open();
                SqlCommand cmd = new SqlCommand("exec usp_pedidos_year By", cn);
                cmd.Parameters.AddWithValue("@y", y);
                SqlDataReader dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                    temporal.Add(new Pedido()
                    {
                        idpedido = dr.GetInt32(0),
                        fecha = dr.GetDateTime(1),
                        cliente = dr.GetString(2),
                        direccion = dr.GetString(3),
                        ciudad = dr.GetString(4)
                    });
                }
                dr.Close();
            }
            return temporal;
        }
    }
}

```

Nota. Elaboración propia.

En el Controlador, agregamos un ActionResult llamado PáginaPedidos() definiendo los parámetros llamado **p** (tipo entero) el cual representa el número de la página, **y** de tipo int, el cual filtra los registros por año.

Figura 140

Desarrollo de Laboratorio

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using appWeb04.Models;
using System.Data.SqlClient;

namespace appWeb04.Controllers
{
    public class NegociosController : Controller
    {
        // GET: Negocios
        public ActionResult Index()
        {
            return View();
        }

        public IEnumerable<Pedido> pedidosYear(int y)
        {
            ...
        }

        public ActionResult PáginaPedidos(int p=0, int y=0)
        {
            ...
        }
    }
}

```

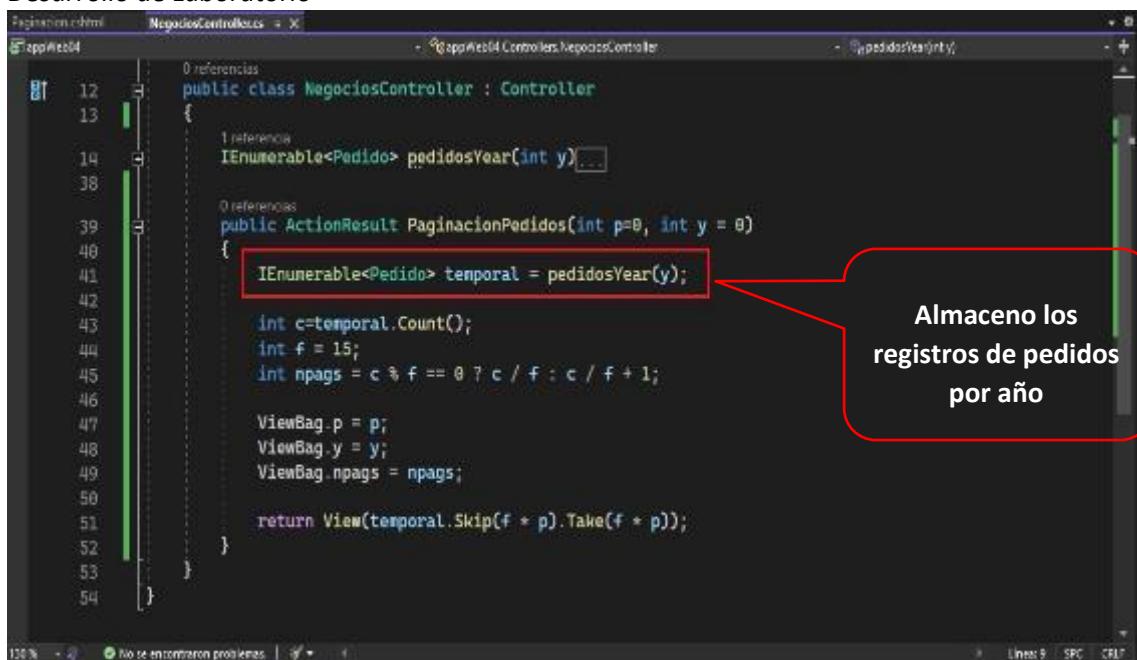
Nota. Elaboración propia.

En el ActionResult PaginacionPedidos

- Almaceno la búsqueda en una lista numerada llamada temporal.
- Defina la variable c, que almacena la cantidad de productos de la búsqueda; la variable filas se le asigna 15 (15 registros por página), la variable npags, que representa la cantidad de páginas.
- Definimos 3 ViewBag; los cuales almacena el número de página (ViewBag.p), la cantidad de páginas (ViewBag.npags) y el valor del parámetro nombre.
- El ActionResult retorna la cantidad de registros según la página seleccionada.

Figura 141

Desarrollo de Laboratorio



```

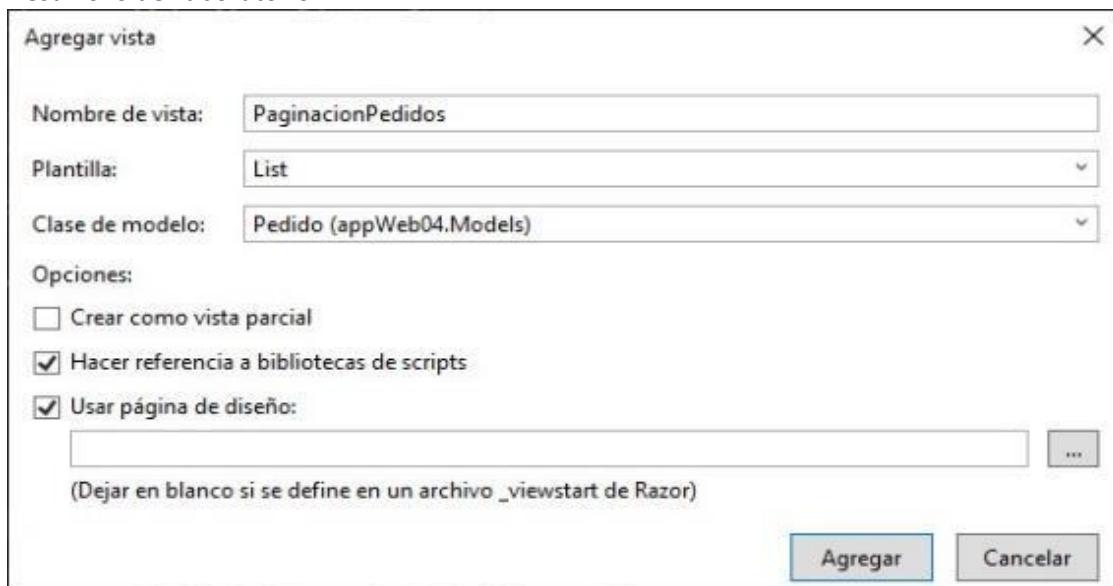
12
13
14  public class NegociosController : Controller
15  {
16      1 referencia
17      IEnumerable<Pedido> pedidosYear(int y)...]
18
19      0 referencias
20      public ActionResult PaginacionPedidos(int p=0, int y = 0)
21      {
22          IQueryable<Pedido> temporal = pedidosYear(y); // Linea resaltada
23
24          int c=temporal.Count();
25          int f = 15;
26          int npags = c % f == 0 ? c / f : c / f + 1;
27
28          ViewBag.p = p;
29          ViewBag.y = y;
30          ViewBag.npags = npags;
31
32          return View(temporal.Skip(f * p).Take(f * p));
33      }
34  }

```

Nota. Elaboración propia.

Agregar la vista de PaginacionPedidos: lista de Pedido

Figura 142
Desarrollo de Laboratorio



Nota. Elaboración propia.

En la vista defina un helper de Formulario, para ingresar el año del pedido en el parámetro “y”, tal como se muestra.

Figura 143
Desarrollo de Laboratorio

```

PáginaPedidos.cshtml  X  Resumen      Nuevos Controles
Model: IEnumerable<appWeb04.Models.Pedido>

ViewBag.Title = "PáginaPedidos";

## Página de Pedidos por Año



using (Html.BeginForm())
{
    <em>Ingrese el Año</em>
    <input type="number" name="y" value="DateTime.Today.Year" min="1990" max="DateTime.Today.Year"/>
    <button>Consulta</button>
}



| Detalle                            |
|------------------------------------|
| <i>foreach (var item in Model)</i> |


```

Nota. Elaboración propia.

En la vista, agregamos un bloque `<div>`, donde a través de un `for`, vamos a imprimir `ActionLink` los cuales representan al número de página a seleccionar y visualizar. En el `ActionLink` se coloca: título, controlador, las variables: `p` con el valor de `i`, `“y”` con el valor de `ViewBag.y`, y el select de `class`.

Figura 144
Desarrollo de Laboratorio

```

PáginaPedidos.cshtml  × NegocioController.cs
@model IEnumerable<appWeb04.Models.Pedido>

@{ ViewBag.Title = "PáginaPedidos"; }

<h2>Página de Pedidos por Año</h2>

<div>
    <table class="table">...</table>

    <div>
        <for (int i = 0; i < (int)ViewBag.npags; i++)>
        {
            @Html.ActionLink(
                (i + 1).ToString(),
                "PáginaPedidos",
                new { p = i, y=ViewBag.y },
                new { @class = "btn btn-danger" })
        }
    </div>

```

A través de un for agregamos ActionLink que representan los números de la paginación

Nota. Elaboración propia.

Presiona la tecla F5, ingrese el año. Al hacer clic en el button Consulta filtramos los registros, y al finalizar la lista visualizamos los links de la paginación.

Figura 145
Desarrollo de Laboratorio

Id Pedido	Fecha Pedido	Cliente	Dirección Destino	Ciudad Destino	
11093	19/09/2018	Ratlesnake Canyon Grocery	2817 Milton Dr.	Albuquerque	Edit Details Delete
11094	20/10/2018	Bon app	2817 Milton Dr.	Albuquerque	Edit Details Delete
11095	16/11/2018	Richter Supermarkt	Starenweg 5	Génova	Edit Details Delete
11096	02/12/2018	Bon app	12, rue des Bouchans	Marsella	Edit Details Delete
11097	19/12/2018	Ratlesnake Canyon Grocery	2817 Milton Dr.	Albuquerque	Edit Details Delete
11098	26/11/2018	Bon app	2817 Milton Dr.	Albuquerque	Edit Details Delete
11099	15/11/2018	Pericles Comidas clásicas	Calle Dr. Jorge Cash 321	Méjico D.F.	Edit Details Delete
11100	16/11/2018	Sin nombre	Vivaldi 34	Köln	Edit Details Delete

Nota. Elaboración propia.

Resumen

1. 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.
2. 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.
3. 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.
4. Hay HTML Helpers para toda clase de controles de formulario Web: check boxes, hidden fields, password boxes, radio buttons, text boxes, text areas, DropDownList lists y list boxes.
5. Hay también un HTML Helper para el elemento Label, los cuales nos asocian descripciones de texto a un formulario Web. Cada HTML Helper nos da una forma rápida de crear HTML válido para el lado del cliente.
6. 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.

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- [https://learn.microsoft.com/en-us/previous-versions/windows/apps/jj883732\(v=win.10\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/windows/apps/jj883732(v=win.10)?redirectedfrom=MSDN)
- <https://juanmalao.wordpress.com/2011/01/30/asp-net-mvc-3-sintaxis-de-razor-y/>
- [https://learn.microsoft.com/en-us/previous-versions/aspnet/dd410123\(v=vs.100\)?redirectedfrom=MSDN](https://learn.microsoft.com/en-us/previous-versions/aspnet/dd410123(v=vs.100)?redirectedfrom=MSDN)

2.3. MANIPULACIÓN DE DATOS

2.3.1 Validaciones de datos: uso del DataAnnotations

El espacio de nombres **System.ComponentModel.DataAnnotations**, nos permite llevar a cabo las validaciones de datos de acuerdo a nuestras definiciones del dominio de cada estructura. 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 atributo validador para realizar la validación. El namespace **System.ComponentModel.DataAnnotations** incluye los siguientes atributos de validación:

- **Range** - Permite validar si el valor de una propiedad se sitúa entre un rango específico de valores.
- **RegularExpression** - Permite validar si el valor de una propiedad coincide con un patrón de expresión regular especificada.
- **Required** - Le permite marcar a que el atributo debe ser de un campo obligatorios, este atributo puede ser utilizado junto a `ErrorMessage` para indicar un mensaje personalizado de error en caso de que no se cumpla con la validación.
- **StringLength** - Le permite especificar una longitud máxima para una propiedad de cadena. Este atributo puede ir en conjunción con **MinimunLength** para indicar el tamaño mínimo del campo string.
- **DataType** – Indica un nombre de un tipo adicional que debe asociarse a un campo de datos.
- **CustomValidation** – permite validar a través de validaciones personalizadas.

Tipos de atributos

ErrorMessage es una propiedad de esta clase que heredarán 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 código del cliente")]
public string idcliente { get; set; }
```

Esta propiedad tiene un '*FormatString*' implícito por el cual mediante la notación estándar de '*FormatString*' puede 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 *MaxLengthAttribute* y *MinLengthAttribute* son utilizados por EF para la validación en el lado del servidor y estos se diferencian del *StringLengthAttribute*, que utilizan un atributo para cada validación, y no solo sirven para validar tamaños de string, sino que también validan tamaños de arrays.

RegularExpressionAttribute

Especifica una restricción mediante una expresión regular

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

RequiredAttribute

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

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

RangeAttribute

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

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

DataTypeAttribute

Especifica el tipo de dato del atributo.

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

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

2.3.2. Operaciones de actualización sobre un origen de datos, manejo de la clase Command

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, retornando la cantidad de registros afectados en el proceso. Para ejecutar comandos que ejecuta las operaciones de CRUD, cree un objeto **Command** definiendo el comando SQL adecuado y una connection a la fuente de datos, incluidos los Parameters necesarios. El comando se debe ejecutar con el método **ExecuteNonQuery** del objeto **Command**.

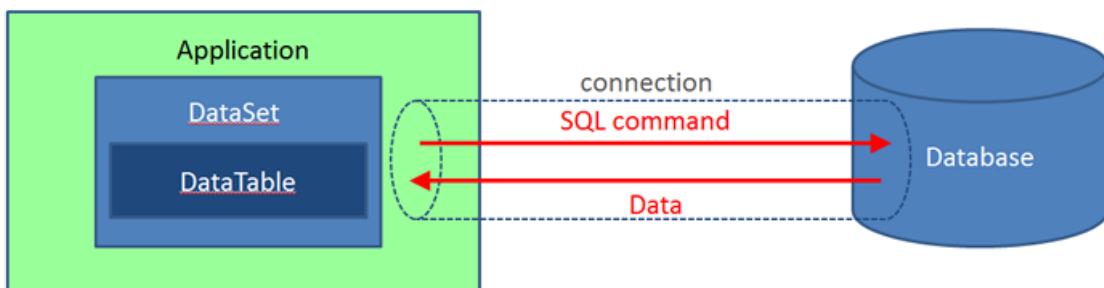
El método **ExecuteNonQuery** devuelve un entero que representa el número de filas que se ven afectadas por la instrucción o por el procedimiento almacenado que se haya ejecutado. Si se ejecutan varias instrucciones, el valor devuelto es la suma de los registros afectados por todas las instrucciones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

Aunque el Manejador de base de datos de destino es SQL Server, las mismas técnicas se pueden aplicar a otros manejadores de base de datos porque la sintaxis de consulta utilizada es SQL estándar que generalmente es compatible con todos los sistemas de bases de datos relacionales.

Manejo de la clase Command: propiedades y métodos

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

Figura 146
Diagrama del objeto Command



Nota. Tomado de *ADO.NET*, por Computer Science Bytes, s.f., dotnetero.com, (<https://www.computersciencebytes.com/object-oriented-programming/ado-net/>)

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**:

Tabla 6
Objeto Command

Proveedor	Descripción
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

Nota. Adaptado de *DbCommand Class*, por Microsoft, s.f., (<https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbcommand?view=netframework-4.8>)

Ejecución 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:

Tabla 7
Valor de retorno

Comando	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 .

Nota. Adaptado de *DbCommand Class*, por Microsoft, s.f., (<https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbcommand?view=netframework-4.8>)

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:

Tabla 8
CommandType

Comando	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.

Nota. Adaptado de *DbCommand Class*, por Microsoft, s.f., (<https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbcommand?view=netframework-4.8>)

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

A continuación, se muestra la descripción de los enumerables del CommandBehavior:

Tabla 9
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

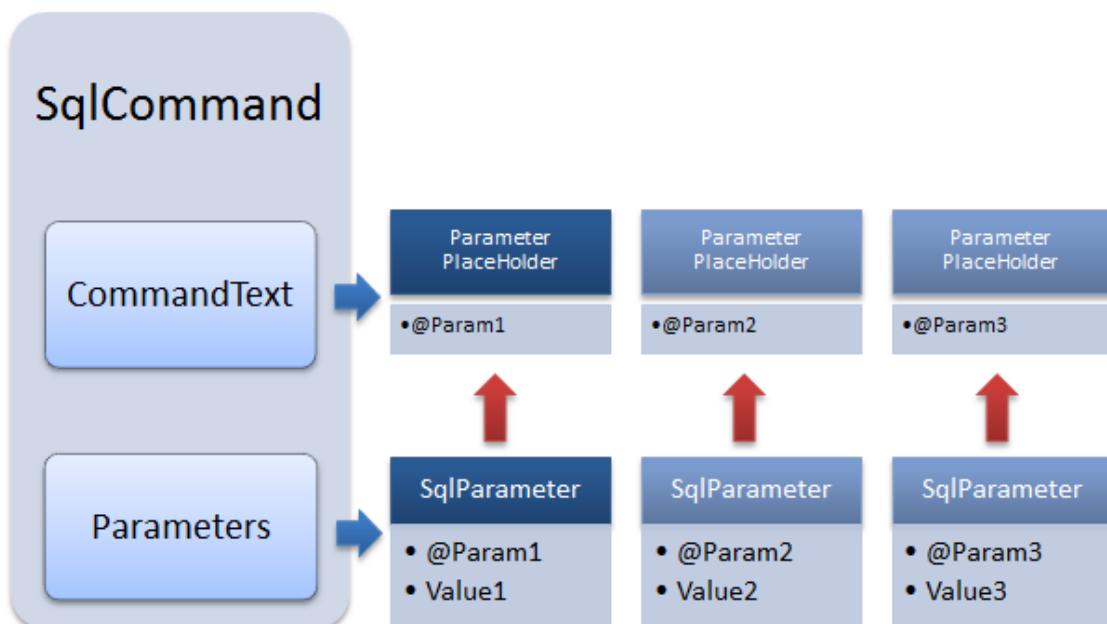
Nota. Adaptado de *DbCommand Class*, por Microsoft, s.f., (<https://docs.microsoft.com/en-us/dotnet/api/system.data.common.dbcommand?view=netframework-4.8>)

Ejecutando operaciones de actualización de datos utilizando 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 147
Diagrama del objeto SqlCommand



Nota. Tomado de *Ado.net Tutorial*, por Wordpress, 2011, wordpress.com, (<http://yinyangit.wordpress.com/2011/08/05/ado-net-tutorial-lesson-06-adding-parameters-to-sqlcommands/>)

Manejo de parámetros en actualización 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.

2.3.3. Trabajando con imágenes: uso de la clase File y HttpPostFile

Una imagen puede guardarse tanto en una base de datos como en una carpeta dedicada a guardar imágenes en el servidor. Un punto importante es que a diferencia de las aplicaciones de escritorio en donde el componente **OpenFileDialog** permitía filtrar los tipos de archivos a seleccionar con el **FileUpload** esto no es posible, el usuario podrá seleccionar el tipo de archivo que desee, así que queda en manos del programador el definir los tipos de archivos permitidos.

Si se guarda en la base de datos debe especificarse el tipo de dato como **varbinary(MAX)** y no como tipo **IMAGE** ya que según documentación de Microsoft este tipo de datos está próximo a desaparecer, igual se debe tener en cuenta el peso que supone a una base de datos el guardar imágenes, además la imagen a guardar se debe convertir en un array de bytes para almacenar el array en la base de datos.

De acuerdo con los expertos en aplicaciones web, es mejor guardar las imágenes en una carpeta que subirlas a la base de datos: Es mejor guardar en el disco duro la imagen y en la base de datos, la ruta a dicha imagen. Nuestra imagen, contenida en el objeto **HttpPostedFile** dentro de un objeto de clase **FileStream**, deberá ser convertida a un conjunto de bytes para enviarlo a la base de datos. Dicho de otro modo, convertiremos el objeto de tipo **Stream** a un arreglo de bytes.

Uso de la clase File y HttpPostFile para la actualización de datos e imágenes

Desarrollar una aplicación que nos permita subir archivos a un servidor desde un explorador Web es un proceso bastante sencillo, ya que la etiqueta <input type="file" /> hace prácticamente todo el trabajo por nosotros en lo que se refiere a la parte del "front-end".

En primer lugar, debemos crear un formulario donde hagamos uso del tag input con el type="file". Además, necesitamos especificar a nivel de formulario el atributo enctype. Este atributo indica el tipo de contenido que se va a enviar al servidor. Como nos cuentan en la página oficial de W3, cuando el método utilizado es post el valor por defecto de este atributo es "application/x-www-form-urlencoded". Sin embargo, si necesitamos hacer uso del elemento input de tipo file debemos modificar este valor por "multipart/form-data".

```
@using(Html.BeginForm(
method="post" enctype="multipart/form-data" action= "metodo"))
{}
```

La clase **HttpPostedFileBase** es una clase abstracta que contiene los mismos miembros que la clase **HttpPostedFile**. La clase **HttpPostedFileBase** le permite crear clases derivadas que son como la clase **HttpPostedFile**, pero que puede personalizar y que funcionan fuera de la tubería ASP.NET.

Cuando realiza pruebas unitarias, generalmente usa una clase derivada para implementar miembros que tienen un comportamiento personalizado que cumple con el escenario que está probando.

Tabla 10
Propiedad & método

Propiedad	Descripción
ContentLength	Obtiene el tamaño de un archivo cargado, en bytes.
ContentType	Obtiene el tipo de contenido MIME de un archivo cargado.
FileName	Obtiene el nombre completo del archivo en el cliente.
Método	Descripción
SaveAs(String)	Guarda el contenido de un archivo cargado.
ToString()	Devuelve una cadena que representa el objeto actual.

Nota. Adaptado de *HttpPostedFileBase Class*, por Microsoft, .f., (<https://docs.microsoft.com/en-us/dotnet/api/system.web.httppostedfilebase?view=netframework-4.8>)

Este es el método de acción que publicará esta vista que guarda el archivo en un directorio en la carpeta App_Data llamado "App_Data".

```
[HttpPost] public ActionResult Index (HttpPostedFileBase file)
{
    if (file.ContentLength > 0 )
    {
        var fileName = Path.GetFileName (file.FileName);
        var path = Path.Combine (Server.MapPath("~/App_Data/uploads"), fileName);
        file.SaveAs (path );
    }

    return RedirectToAction ("Index");
}
```

2.3.4. Manejo de transacciones, uso de la clase Transaction

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 error 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.

La clase Transaction

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. Además, existe una nueva clase DbTransaction disponible para la escritura de código independiente del proveedor que requiere transacciones

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.

Tabla 11*Propiedad & métodos*

Propiedad	Descripción
Connection	Obtiene el objeto SqlConnection asociado a la transacción o null si la transacción ya no es válida.
IsolationLevel	Especifica el nivel de aislamiento para esta transacción: Chaos: Los cambios pendientes de las transacciones más aisladas no se pueden sobre escribir ReadCommitted: Los bloqueos compartidos se mantienen mientras se están leyendo los datos para evitar lecturas erróneas. ReadUncommitted: Se pueden producir lecturas erróneas, lo que implica que no se emitan bloqueos compartidos y que se cumplan los bloqueos exclusivos Serializable: Se realiza un bloqueo de intervalo en DataSet, lo que impide que otros usuarios actualicen o inserten filas en el conjunto de datos hasta que la transacción haya terminado.
	Snapshot: Reduce el bloqueo almacenado una versión de los datos que una aplicación puede leer mientras otra los está modificando.
Métodos	Descripción
Commit()	Confirma la transacción de base de datos
CommitAsync()	Confirma de forma asíncrona, la transacción de la base de datos
Dispose()	Libera los recursos no administrados que DbTransaction usa.
DisposeAsync()	Desecha de forma asíncrona el objeto de transacción.
Rollback()	Revierte una transacción desde un estado pendiente
Rollback(String)	Deshace una transacción con un estado pendiente y especifica el nombre de la transacción o del punto de almacenamiento
RollbackAsync()	Revierte de forma asíncrona una transacción desde un estado pendiente

Nota. Adaptado de *HttpPostedFileBase Class*, por Microsoft, .f., (<https://docs.microsoft.com/en-us/dotnet/api/system.web.httppostedfilebase?view=netframework-4.8>)

LABORATORIO 5.1.: Actualizando datos con procedimientos almacenados

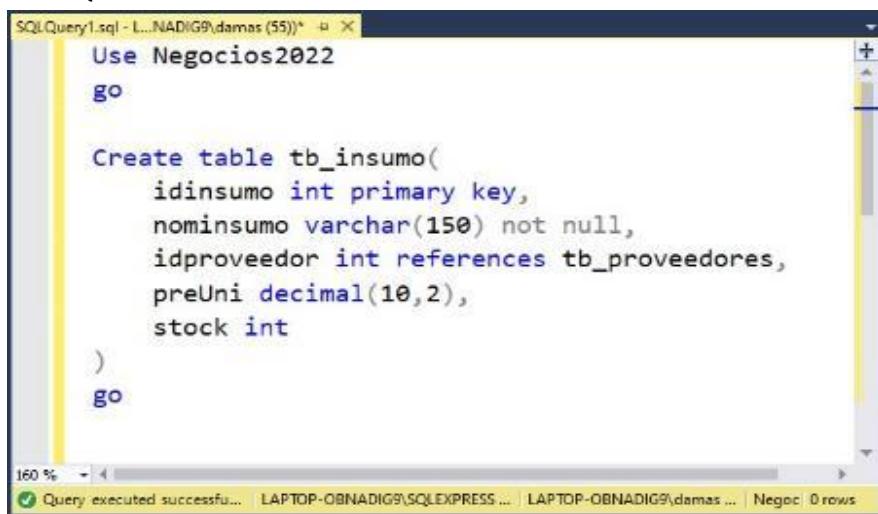
Se desea implementar un proyecto en ASP.NET MVC donde permita insertar, actualizar, eliminar y consultar los datos de la tabla tb_insumo almacenado en una base de datos en SQL Server.

Creando la tabla y procedimientos almacenados en SQL Server

Creando la tabla tb_insumo

Figura 148

Trabajando en SQL Server



```
SQLQuery1.sql - L...NADIG9\damas (55)* + X
Use Negocios2022
go

Create table tb_insumo(
    idinsumo int primary key,
    nominsumo varchar(150) not null,
    idproveedor int references tb_proveedores,
    preUni decimal(10,2),
    stock int
)
go

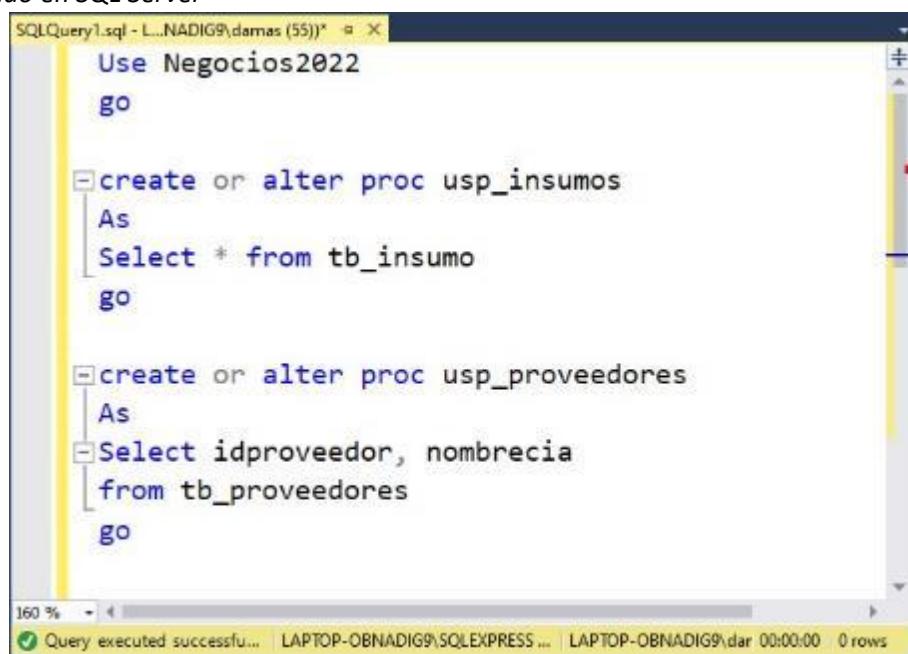
160 % < >
Query executed successfully | LAPTOP-OBNADIG9\SQLEXPRESS ... | LAPTOP-OBNADIG9\damas ... | Negocios2022 | 0 rows
```

Nota. Elaboración propia.

Creando los procedimientos donde liste los registros de tb_insumo y tb_proveedor (tabla de referencia)

Figura 149

Trabajando en SQL Server



```
SQLQuery1.sql - L...NADIG9\damas (55)* + X
Use Negocios2022
go

create or alter proc usp_insumos
As
Select * from tb_insumo
go

create or alter proc usp_proveedores
As
Select idproveedor, nombrecia
from tb_proveedores
go

160 % < >
Query executed successfully | LAPTOP-OBNADIG9\SQLEXPRESS ... | LAPTOP-OBNADIG9\damas ... | Negocios2022 | 0 rows
```

Nota. Elaboración propia.

Creando el procedure para Insertar un registro a la tabla tb_insumo

Figura 150

Trabajando en SQL Server

```
SQLQuery1.sql - L...NADIG9\damas (55)*  ↗ X
Use Negocios2022
go

create or alter proc usp_inserta_insumo
    @id int,
    @nombre varchar(150),
    @idproveedor int,
    @pre decimal(10,2),
    @stock int
As
Insert into tb_insumo
Values(@id,@nombre,@idproveedor,@pre,@stock)
go
```

Nota. Elaboración propia.

Creando el procedure para actualizar los datos por su campo idinsumo

Figura 151

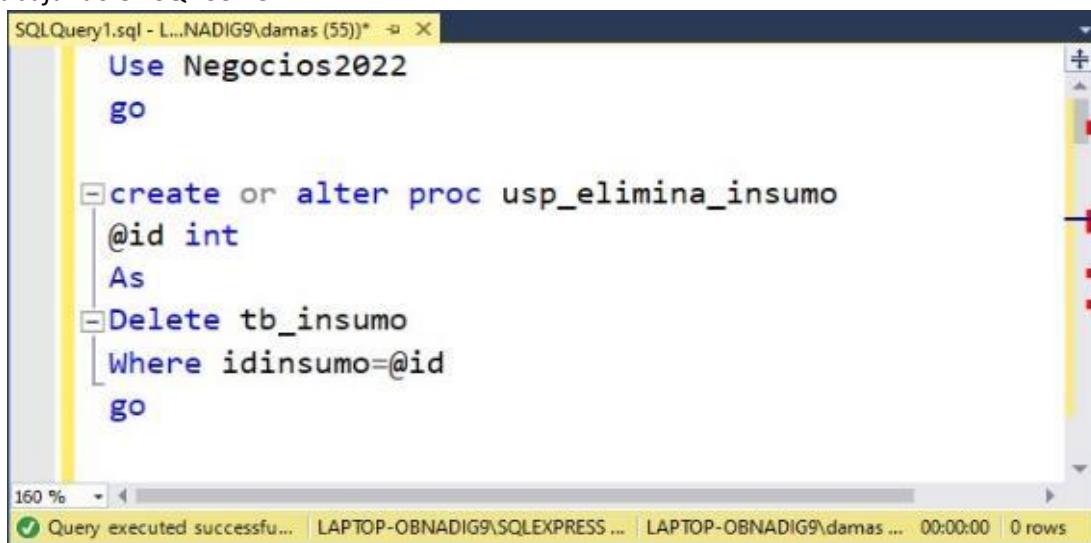
Trabajando en SQL Server

```
SQLQuery1.sql - L...NADIG9\damas (55)*  × X
Use Negocios2022
go

create or alter proc usp_actualiza_insumo
    @id int,
    @nombre varchar(150),
    @idproveedor int,
    @pre decimal(10,2),
    @stock int
As
Update tb_insumo
Set nominsumo=@nombre,idproveedor=@idproveedor,
preuni=@pre,stock=@stock
Where idinsumo=@id
go
```

Nota. Elaboración propia.

Creando el procedure para eliminar un registro por su campo idinsumo

Figura 152*Trabajando en SQL Server*


```

SQLQuery1.sql - L...NADIG9\damas (55)*  X
Use Negocios2022
go

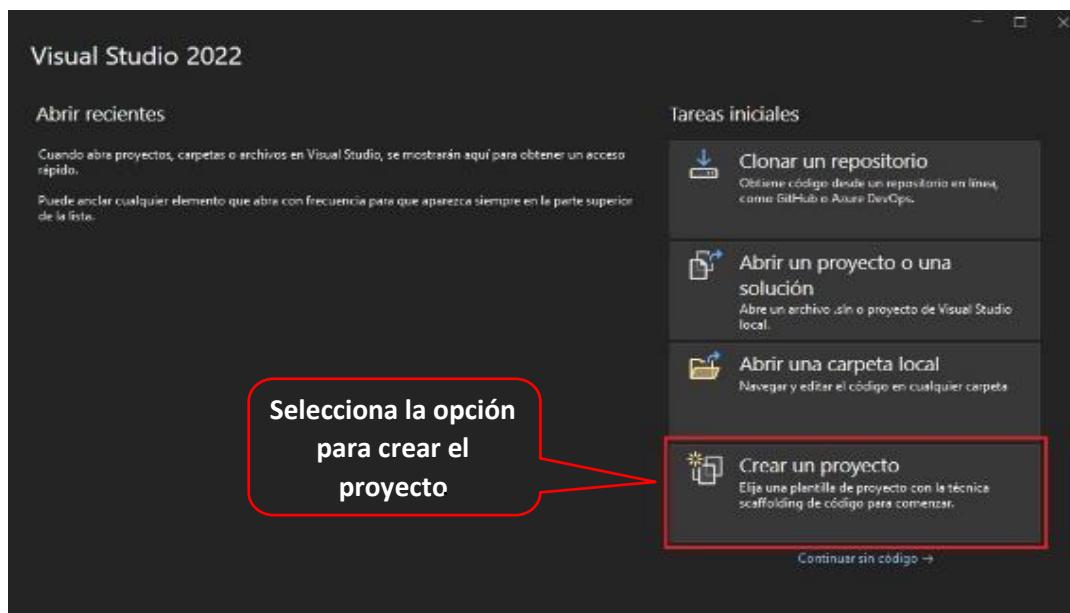
create or alter proc usp_elimina_insumo
@id int
As
Delete tb_insumo
Where idinsumo=@id
go

```

160 % Query executed successfully | LAPTOP-OBNA DIG9\SQLEXPRESS ... | LAPTOP-OBNA DIG9\damas ... 00:00:00 | 0 rows

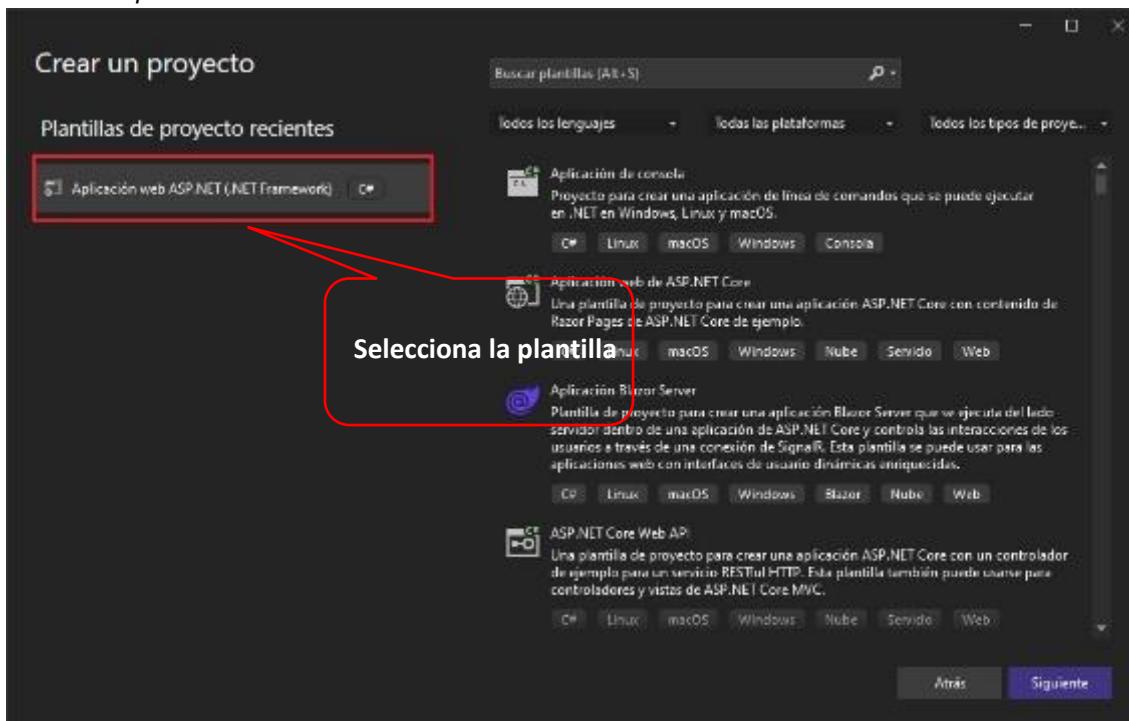
Nota. Elaboración propia.**Creando un Proyecto ASP.NET MVC****Inicio del Proyecto**

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”

Figura 153*Desarrollo práctico**Nota.* Elaboración propia.

Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción Siguiente:

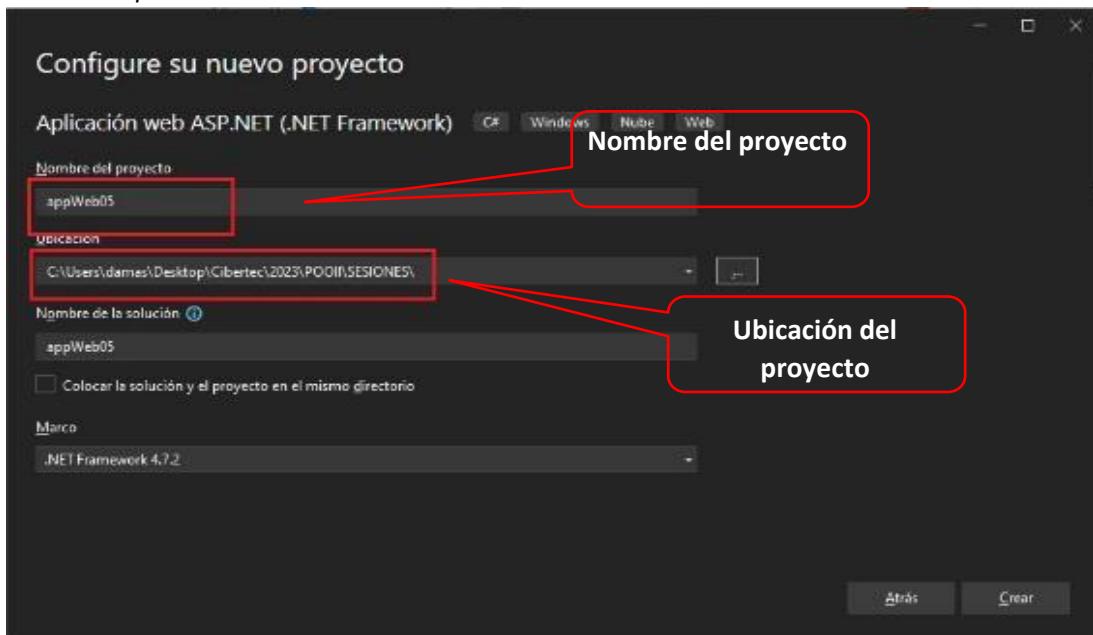
Figura 154
Desarrollo práctico



Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciona la ubicación del mismo, al terminar presiona la opción **Crear**.

Figura 155
Desarrollo práctico



Nota. Elaboración propia.

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

Figura 156
Desarrollo práctico

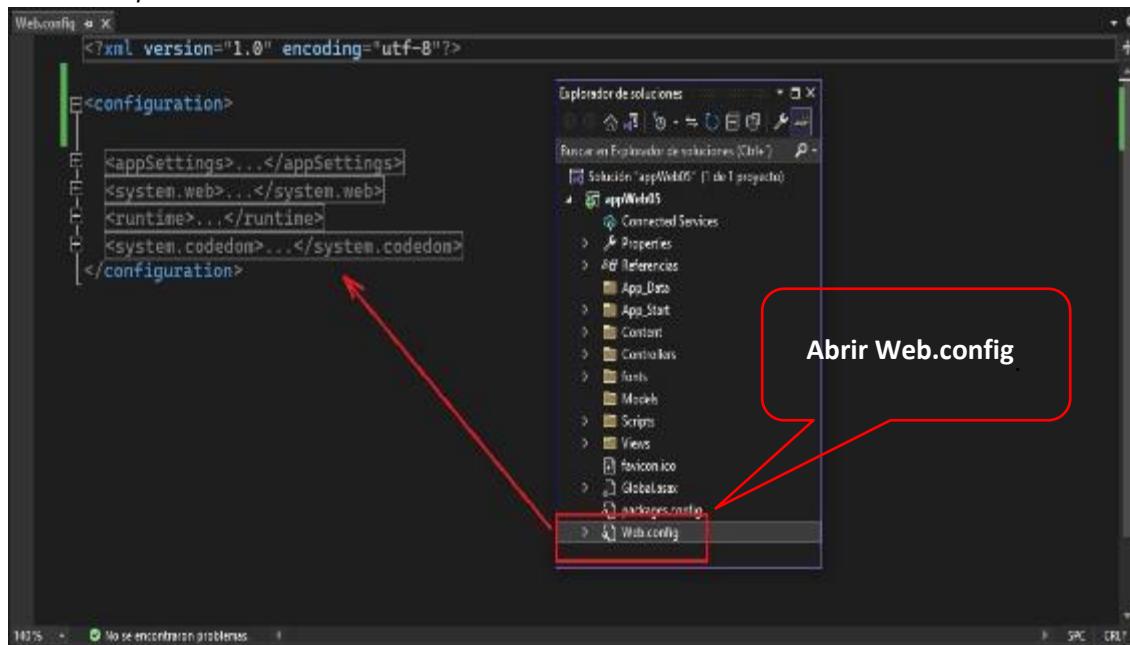


Nota. Elaboración propia.

Publicando la cadena de conexión

Desde el Explorador de proyecto, abrir el archivo Web.config, tal como se muestra.

Figura 157
Desarrollo práctico

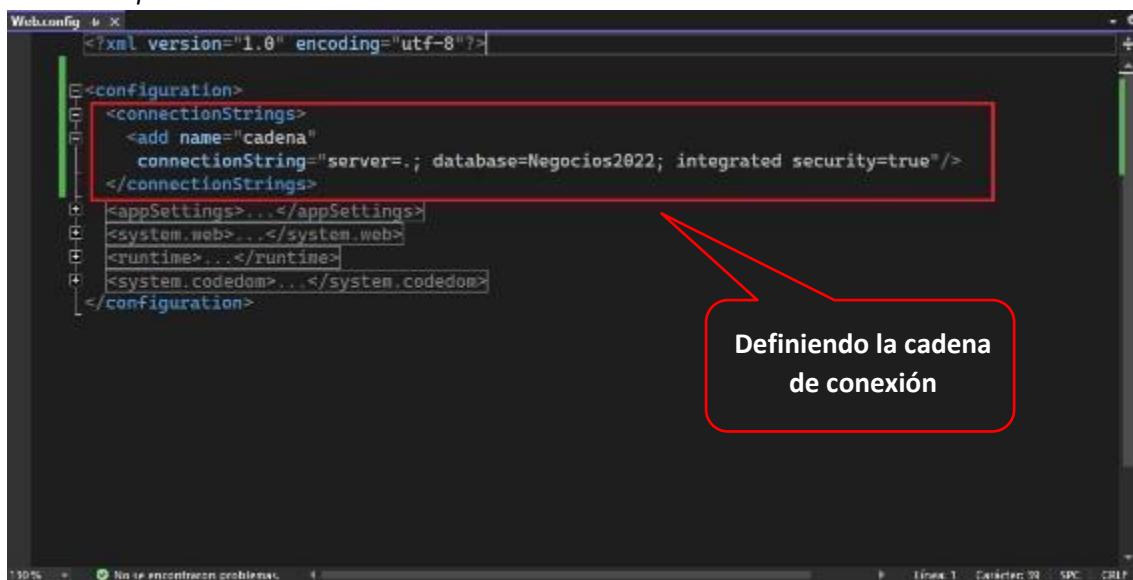


Nota. Elaboración propia.

Defina la etiqueta <connectionStrings> agregando una cadena <add> cuyo nombre es “cadena” y definiendo la cadena de conexión, tal como se muestra.

Figura 158

Desarrollo práctico



The screenshot shows the XML code for the `Web.config` file. A red box highlights the `<connectionStrings>` section, which contains an `<add name="cadena">` element with a specific connection string. A callout bubble points to this section with the text "Definiendo la cadena de conexión".

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="cadena"
      connectionString="server=.; database=Negocios2022; integrated security=true"/>
  </connectionStrings>
  <appSettings>...</appSettings>
  <system.web>...</system.web>
  <runtime>...</runtime>
  <system.codedom>...</system.codedom>
</configuration>

```

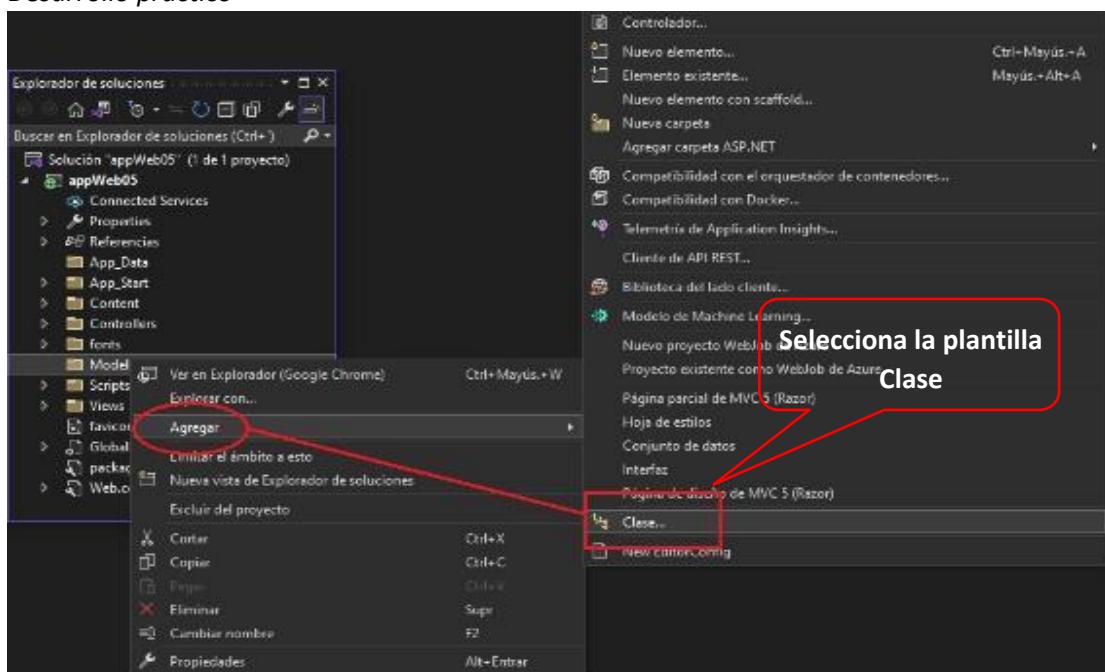
Nota. Elaboración propia.

Agregando la clase Insumo a la carpeta Models

En la carpeta Models, hacer clic derecho sobre la carpeta, seleccionar Agregar → Clase, tal como se muestra en la figura.

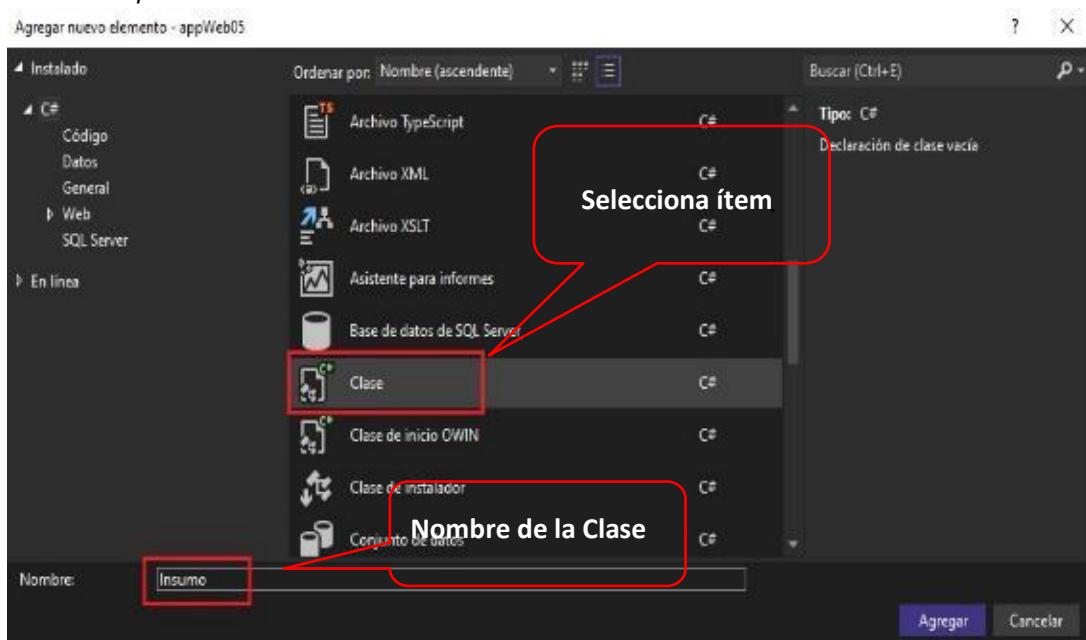
Figura 159

Desarrollo práctico



Nota. Elaboración propia.

Selecciona el elemento Clase, asigne el nombre del elemento: Insumo, tal como se muestra.

Figura 160*Desarrollo práctico**Nota.* Elaboración propia.

Agregar la librería System.ComponentModel.DataAnnotations y atributos de la clase, cada atributo será de ingreso obligatorio asignándole a cada uno la etiqueta **Required**, tal como se muestra.

Figura 161*Desarrollo práctico*

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.ComponentModel.DataAnnotations;
6  namespace appWeb05.Models
7  {
8      public class Insumo
9      {
10         [Display(Name = "Id insumo"), Required] public int idinsumo { get; set; }
11         [Display(Name = "Descripción"), Required] public string descripcion { get; set; }
12         [Display(Name = "Id Proveedor"), Required] public int idproveedor { get; set; }
13         [Display(Name = "Precio Unitario"), Required] public decimal precio { get; set; }
14         [Display(Name = "Stock"), Required] public int stock { get; set; }
15     }
16 }

```

Nota. Elaboración propia.

En la carpeta Models, agrega la clase Proveedor, defina los atributos de la clase, tal como se muestra.

Figura 162
Desarrollo práctico

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace appWeb05.Models
7  {
8      public class Proveedor
9      {
10         public int idproveedor { get; set; }
11         public string nombre { get; set; }
12     }
13 }

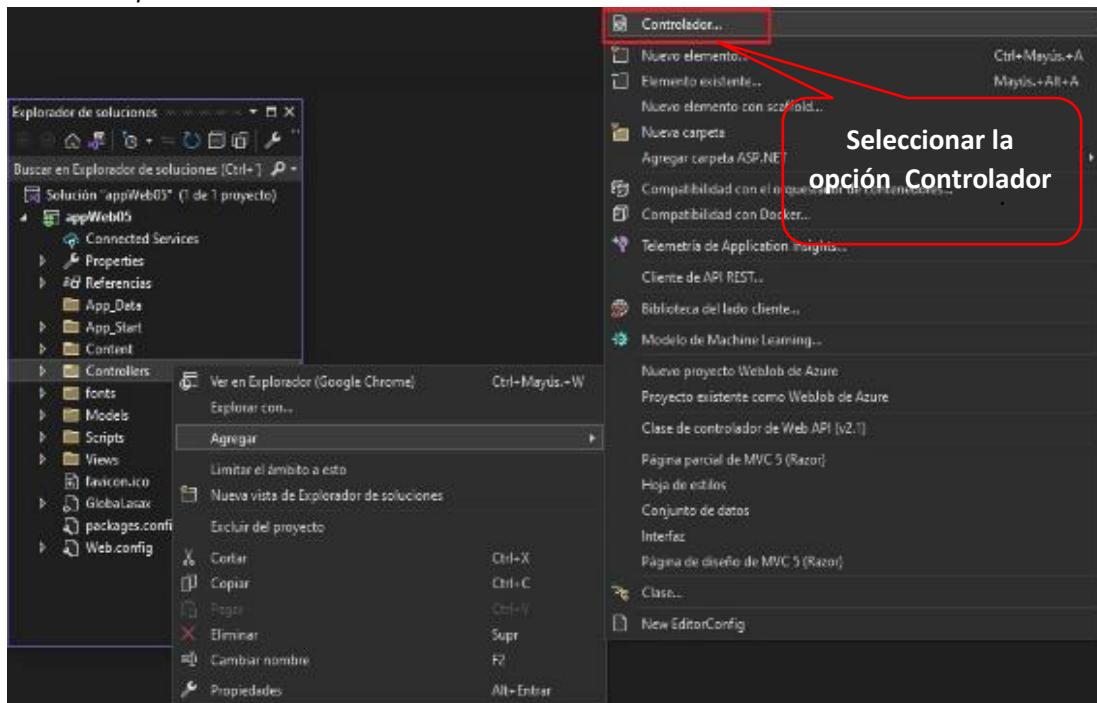
```

Nota. Elaboración propia.

Trabajando con el Controlador

A continuación, en la carpeta Controllers, agregamos un controlador, tal como se muestra:
Controllers → Agregar → Controlador

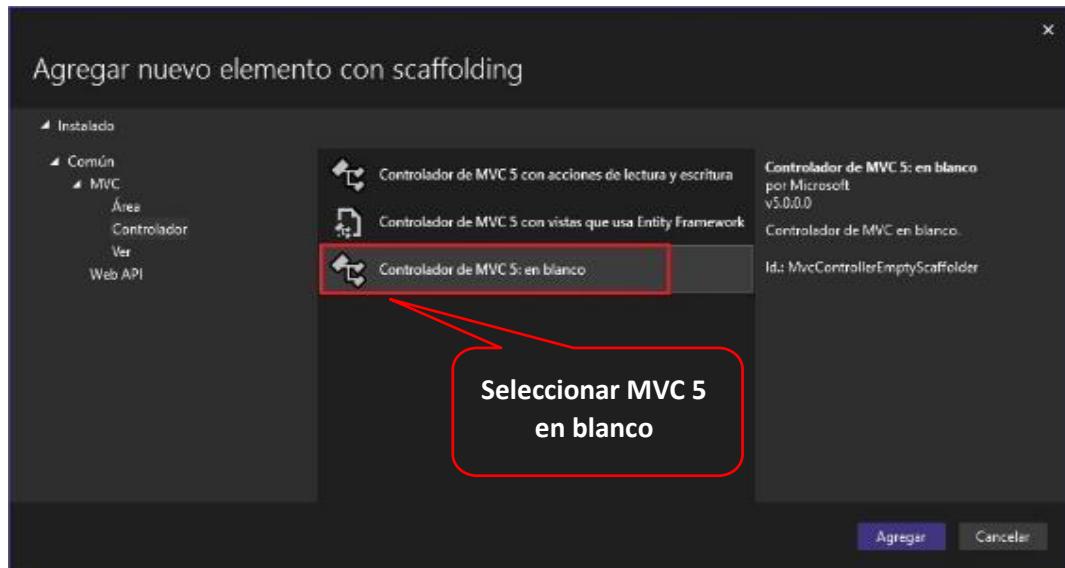
Figura 163
Desarrollo práctico



Nota. Elaboración propia.

Selecciona Controlador MVC5 en blanco, tal como se muestra.

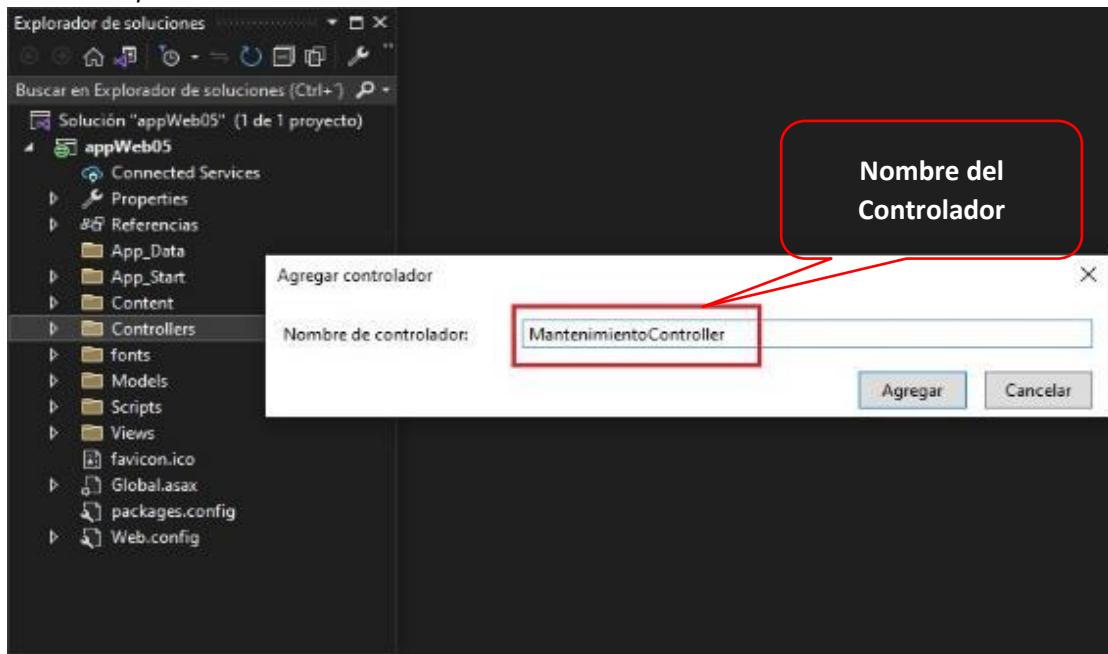
Figura 164
Desarrollo práctico



Nota. Elaboración propia.

Asigne el nombre de NegociosController, tal como se muestra. Presiona el botón Agregar

Figura 165
Desarrollo práctico



Nota. Elaboración propia.

Programando el Controlador

Importar las librerías y la carpeta Models, donde se encuentra almacenado la clase.

Figura 166*Desarrollo práctico*

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using System.Configuration;
7  using System.Data;
8  using System.Data.SqlClient;
9  using appWeb05.Models;
10 namespace appWeb05.Controllers
11 {
12     public class MantenimientoController : Controller
13     {
14     }
15 }

```

Nota. Elaboración propia.

En el Controlador, defina y codifique el método `insumos()` donde retorna la lista numerada de los registros de `tb_insumo`, tal como se muestra.

Figura 167*Desarrollo práctico*

```

14     IEnumarable<Insumo> insumos()
15     {
16         List<Insumo> temporal= new List<Insumo>();
17         using(SqlConnection cn=new SqlConnection(
18             ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
19         {
20             cn.Open();
21             SqlCommand cmd = new SqlCommand("exec usp_insumos", cn);
22
23             SqlDataReader dr=cmd.ExecuteReader();
24             while(dr.Read())
25             {
26                 temporal.Add(new Insumo()
27                 {
28                     idinsumo = dr.GetInt32(0),
29                     descripcion= dr.GetString(1),
30                     idproveedor=dr.GetInt32(2),
31                     precio=dr.GetDecimal(3),
32                     stock=dr.GetInt32(4)
33                 });
34             }
35             dr.Close();
36         }
37         return temporal;
38     }

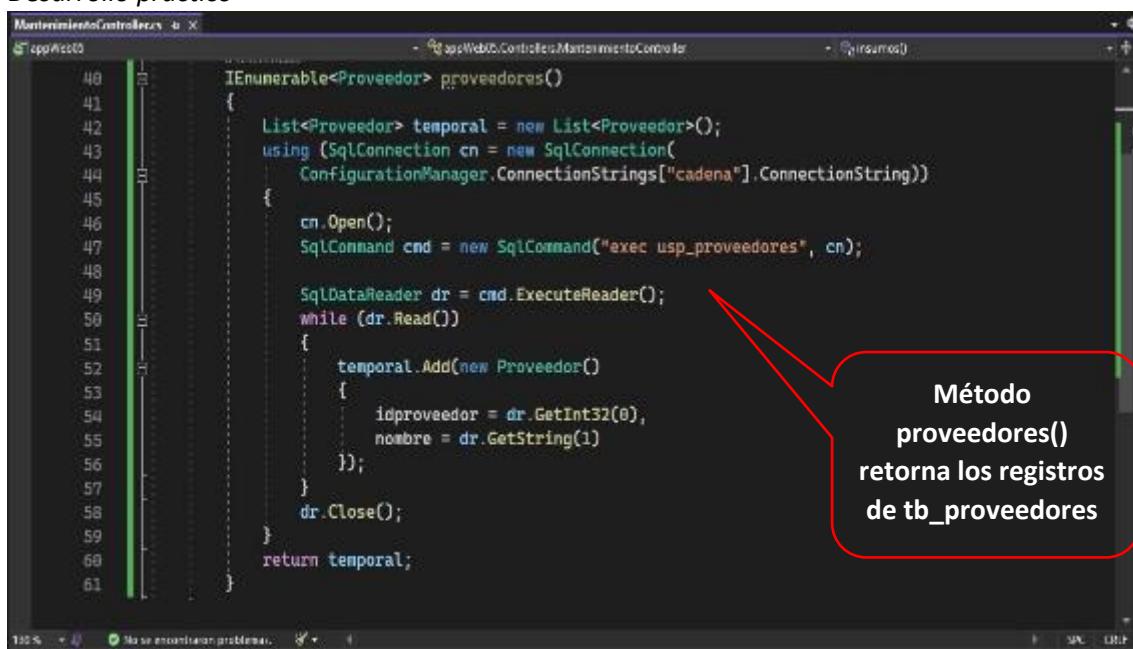
```

Nota. Elaboración propia.

En el Controlador, defina y codifique el método proveedores() donde retorna la lista numerada de los registros de tb_proveedores, tal como se muestra.

Figura 168

Desarrollo práctico



```

MantenimientoController.cs + X
appWeb05 - %\appWeb05\Controllers\MantenimientoController.cs - %\insumos()

40    IEnumarable<Proveedor> proveedores()
41    {
42        List<Proveedor> temporal = new List<Proveedor>();
43        using (SqlConnection cn = new SqlConnection(
44            ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
45        {
46            cn.Open();
47            SqlCommand cmd = new SqlCommand("exec usp_proveedores", cn);
48
49            SqlDataReader dr = cmd.ExecuteReader();
50            while (dr.Read())
51            {
52                temporal.Add(new Proveedor()
53                {
54                    idproveedor = dr.GetInt32(0),
55                    nombre = dr.GetString(1)
56                });
57            }
58            dr.Close();
59        }
60        return temporal;
61    }

```

Método proveedores()
retorna los registros de tb_proveedores

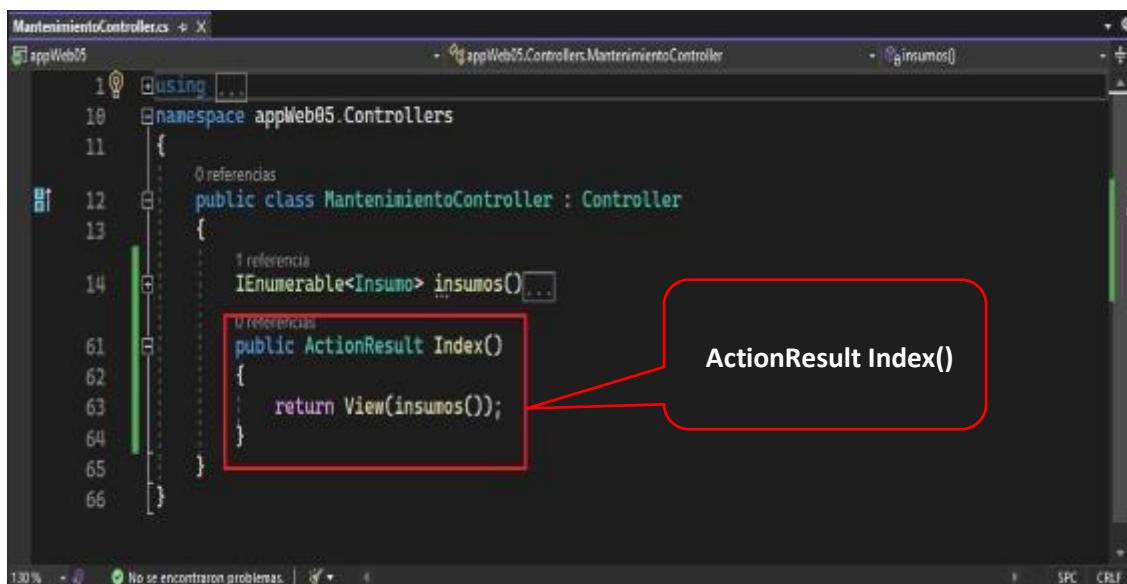
Nota. Elaboración propia.

Trabajando con el ActionResult Index

Defina el ActionResult Index(y), el cual enviará a la Vista el resultado del método insumos() tal como se muestra.

Figura 169

Desarrollo práctico



```

MantenimientoController.cs + X
appWeb05 - %\appWeb05\Controllers\MantenimientoController.cs - %\insumos()

1  using ...
2  namespace appWeb05.Controllers
3  {
4      public class MantenimientoController : Controller
5      {
6          0 referencias
7          public ActionResult Index()
8          {
9              1 referencia
10             IEnumarable<Insumo> insumos()...
11
12             0 referencias
13             public ActionResult Index()
14             {
15                 1 referencia
16                 return View(insumos());
17             }
18         }
19     }
20 }

```

ActionResult Index()

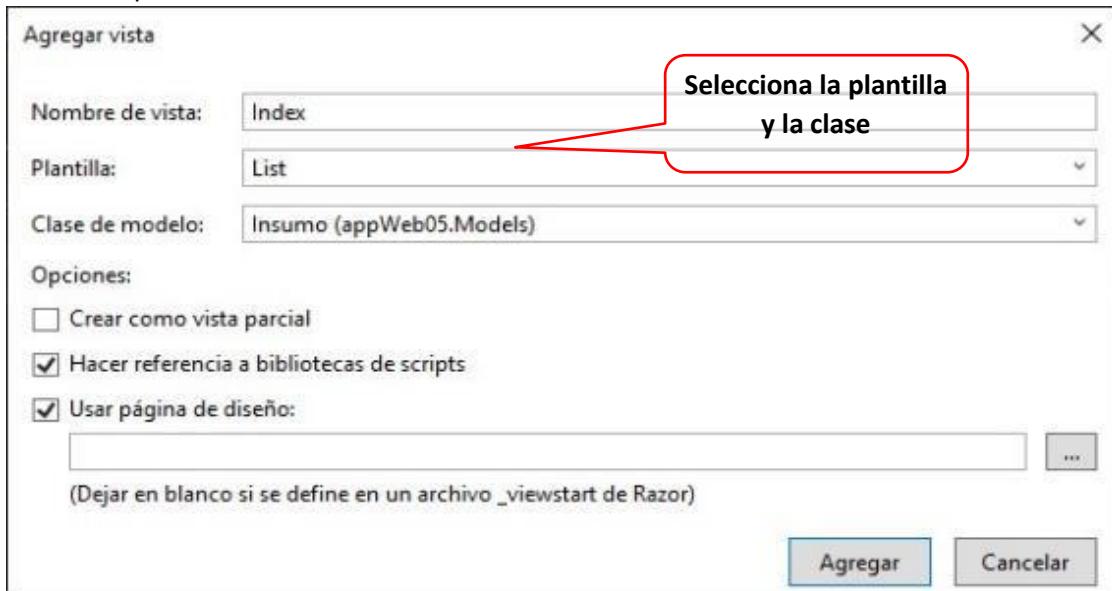
Nota. Elaboración propia.

Trabajando con la Vista Index

- En el ActionResult, hacer clic derecho y selecciona, Agregar vista.
- En dicha ventana, seleccione la plantilla, la cual será List; y la clase de modelo la cual es Insumo, tal como se muestra.

Figura 170

Desarrollo práctico



Nota. Elaboración propia.

Vista generada por la plantilla List, modifique los ActionLink del Create, diseño, y los ActionLink Edit, Details y Delete definiendo el parámetro id con el valor de idinsumo, tal como se muestra.

Figura 171

Desarrollo práctico

```

@model IEnumerable<appWeb05.Models.Insumo>
@{
    ViewBag.Title = "Index";
}


## Index



@Html.ActionLink("Nuevo Insumo", "Create", null, new { @class = "btn btn-primary" })



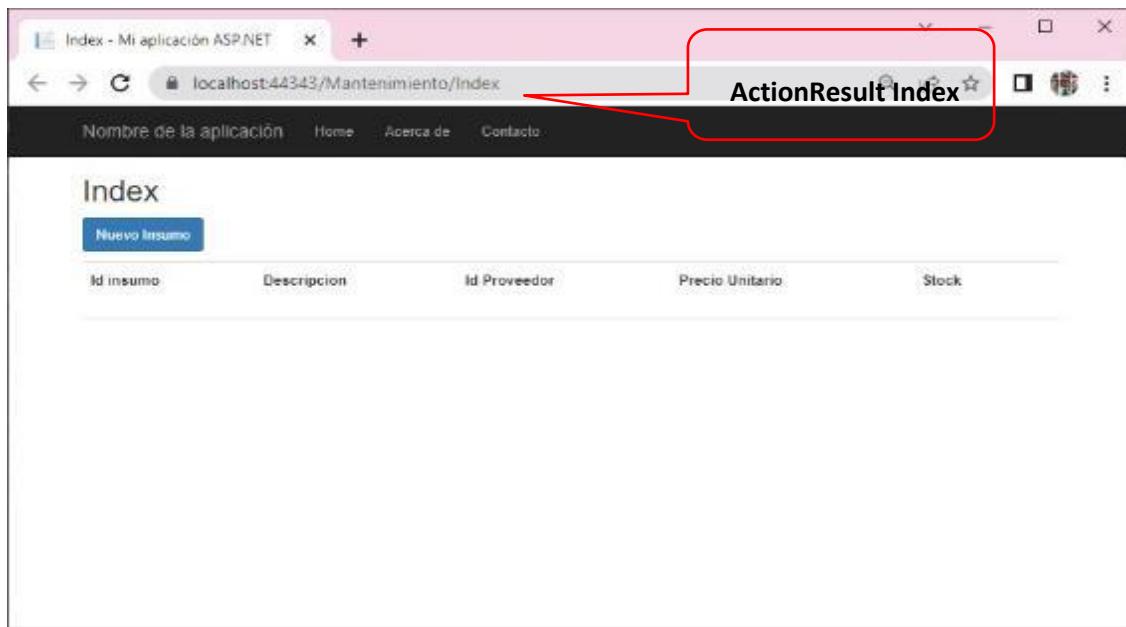
|  | idinsumo | descripcion | idproveedor | precio | stock |
|--|----------|-------------|-------------|--------|-------|
|--|----------|-------------|-------------|--------|-------|


```

Nota. Elaboración propia.

Presiona la tecla F5 para ejecutar el ActionResult Index y visualizar la Vista.

Figura 172
Desarrollo práctico



Nota. Elaboración propia.

Programando el ActionResult Create

En el Controlador, defina el método agregar el cual ejecuta el procedimiento almacenado usp_inserta_insumo, donde retorna el mensaje del proceso.

Figura 173
Desarrollo práctico

```

14     string agregar(Insuno reg)
15     {
16         string mensaje = "";
17         using (SqlConnection cn = new SqlConnection(
18             ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
19         {
20             cn.Open();
21             SqlCommand cmd = new SqlCommand("exec usp_inserta_insumo @id,@nombre,@idproveedor,@pre,@stock", cn);
22             cmd.Parameters.AddWithValue("@id", reg.idinsumo);
23             cmd.Parameters.AddWithValue("@nombre", reg.descripcion);
24             cmd.Parameters.AddWithValue("@idproveedor", reg.idproveedor);
25             cmd.Parameters.AddWithValue("@precio", reg.precio);
26             cmd.Parameters.AddWithValue("@stock", reg.stock);
27
28             int i = cmd.ExecuteNonQuery();
29             mensaje = $"Se ha registrado {i} insumo";
30         }
31         return mensaje;
32     }

```

Nota. Elaboración propia.

Defina el ActionResult Create (GET) donde envía la lista de los proveedores a la vista y un nuevo Insumo, tal como se muestra. En el POST ejecuta el método Agregar almacenando el resultado en un ViewBag, refrescando la vista con los datos del registro ingresado.

Figura 174
Desarrollo práctico

```

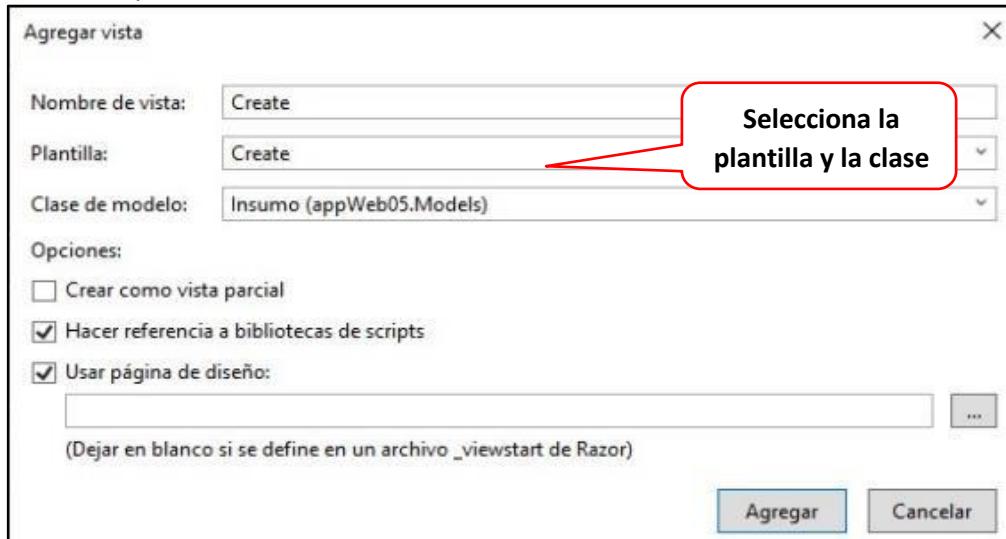
public class MantenimientoController : Controller
{
    string agregar(Insumo reg);
    IEnumerable<Proveedor> proveedores();
    public ActionResult Create()
    {
        ViewBag.proveedores=new SelectList(proveedores(),"idproveedor","nombrecia");
        return View(new Insumo());
    }
    [HttpPost]
    public ActionResult Create[HttpPost](Insumo reg)
    {
        ViewBag.mensaje = agregar(reg);
        ViewBag.proveedores = new SelectList(proveedores(), "idproveedor", "nombrecia", reg.idproveedor);
        return View(reg);
    }
}

```

Nota. Elaboración propia.

A continuación, agregar la vista al ActionResult Create, tal como se muestra

Figura 175
Desarrollo práctico



Nota. Elaboración propia.

En la vista Create, realizar los ajustes correspondientes:

- En el campo idproveedor, cambiar al helper DropDownList.
- Agregar 2 ActionLink: Nuevo, Retornar.
- Agregar el ViewBag.mensaje.

Figura 176
Desarrollo práctico

```

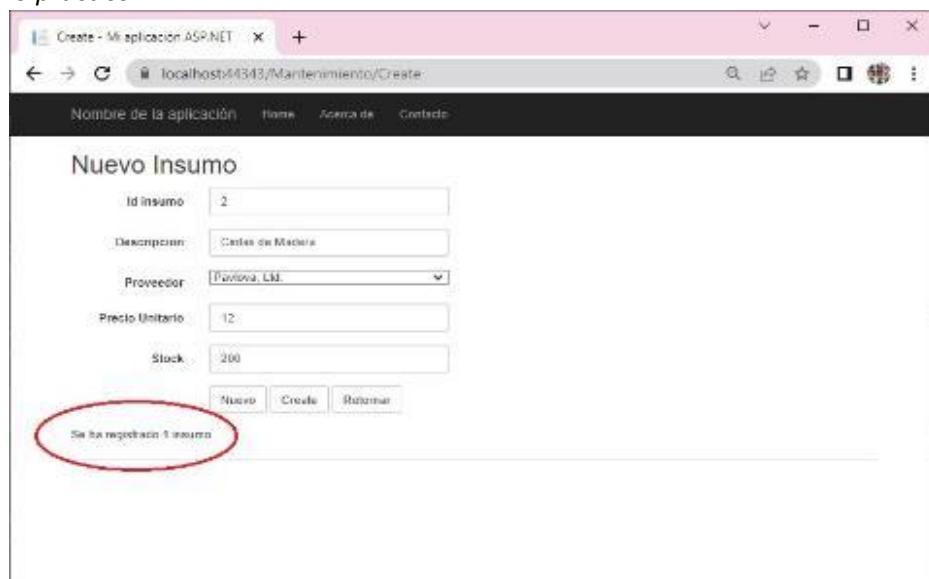
@using (Html.BeginForm())
{
    <div class="form-horizontal">
        <div class="form-group">
            <div class="form-group">
                <div class="col-md-2">
                    @Html.Label("Proveedor", htmlAttributes: new { @class = "control-label col-md-2" })
                </div>
                <div class="col-md-10">
                    @Html.DropDownList("idproveedor", ViewBag.proveedores as SelectList, new { @class = "select-control" })
                </div>
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <Html.ActionLink("Nuevo", "Create", null, new { @class = "btn btn-default" })>
                <input type="submit" value="Create" class="btn btn-default" />
                <Html.ActionLink("Retornar", "Index", null, new { @class = "btn btn-default" })>
            </div>
        </div>
    </div>
    <div>
        @ViewBag.mensaje
    </div>
}

```

Nota. Elaboración propia.

Ejecute la Vista, presiona la tecla F5, ingrese los datos, al presionar el botón Create, se ejecuta el proceso visualizando un mensaje: "se ha insertado 1 registro".

Figura 177
Desarrollo práctico



Nota. Elaboración propia.

Programando el ActionResult Edit

En el controlador Negocios, defina el método actualizar el cual ejecuta el procedimiento almacenado de actualice un registro por el campo idinsumo; y el método buscar donde retorna el registro de tb_insumo por su campo idinsumo.

Figura 178

Desarrollo práctico

```

NuevoministroController.cs
@{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Web;
    using System.Data.SqlClient;
    using System.Configuration;
}

namespace Negocios
{
    public class NegociosController : Controller
    {
        // Método que ejecuta el
        // procedimiento de actualización
        public string actualizar(Insumo reg)
        {
            string mensaje = "";
            using (SqlConnection cn = new SqlConnection(
                ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
            {
                cn.Open();
                SqlCommand cmd = new SqlCommand("exec usp_actualiza_insumo @id, @nombre, @idproveedor, @pre, @stock", cn);
                cmd.Parameters.AddWithValue("@id", reg.idinsumo);
                cmd.Parameters.AddWithValue("@nombre", reg.descripcion);
                cmd.Parameters.AddWithValue("@idproveedor", reg.idproveedor);
                cmd.Parameters.AddWithValue("@pre", reg.precio);
                cmd.Parameters.AddWithValue("@stock", reg.stock);

                int i = cmd.ExecuteNonQuery();
                mensaje = $"Se ha actualizado {i} insumo";
            }
            return mensaje;
        }

        // Método que retorna el
        // insumo por su código
        public Insumo buscar(int id)
        {
            return insumos().FirstOrDefault(x => x.idinsumo == id);
        }
    }
}

```

Nota. Elaboración propia.

El ActionResult Edit de tipo GET, el cual retorna el registro del Insumo por su campo idinsumo, el POST ejecuta el método actualizar. En ambos casos enviar la lista de proveedores.

Figura 179

Desarrollo práctico

The screenshot shows the code editor with the file `MantenimientoController.cs` open. The code implements a controller for managing inventories. It includes methods for displaying a record for editing and updating it via POST. A red callout points to the `Edit(int id)` method with the text "Envía el registro de Insumo por su código". Another red callout points to the `[HttpPost]public ActionResult Edit(Insumo reg)` method with the text "Ejecuta el método para actualizar el registro".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using SppWeb06.Clases;
using SppWeb06.Clases.Mantenimientos;
using SppWeb06.Clases.Proveedores;

namespace SppWeb06.Controllers
{
    public class MantenimientoController : Controller
    {
        // GET: Mantenimiento
        public ActionResult Index()
        {
            return View();
        }

        // GET: Mantenimiento/Edit/5
        public ActionResult Edit(int id)
        {
            Insumo reg = buscar(id);

            ViewBag.proveedores = new SelectList(proveedores(), "idproveedor", "nombre", reg.idproveedor);
            return View(reg);
        }

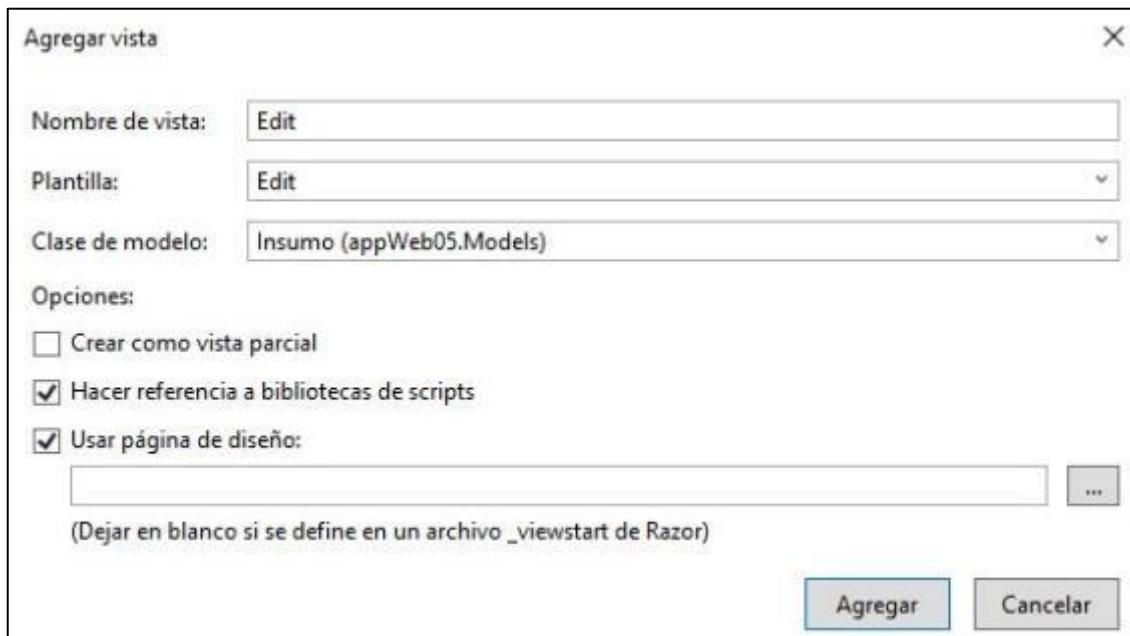
        [HttpPost]
        public ActionResult Edit(Insumo reg)
        {
            ViewBag.mensaje = actualizar(reg);
            ViewBag.proveedores = new SelectList(proveedores(), "idproveedor", "nombre", reg.idproveedor);
            return View(reg);
        }
    }
}
```

Nota. Elaboración propia.

A continuación, agregar la vista al ActionResult Edit, tal como se muestra:

Figura 180

Desarrollo práctico



Nota. Elaboración propia.

En la vista Edit, realizar los ajustes correspondientes: En el campo idproveedor, cambiar a DropDownList; agregar 1 ActionLink: Retornar y agregar el ViewBag.mensaje.

Figura 181

Desarrollo práctico

```
<div class="form-horizontal">
    <div class="form-group"></div>
    <div class="form-group"></div>

    <div class="form-group">
        <@Html.Label("Proveedor", htmlAttributes: new { @class = "control-label col-md-2" })>
        <div class="col-md-10">
            <@Html.DropDownList("idproveedor", ViewBag.proveedores as SelectList, new { @class = "select-control" })>
        </div>
    </div>

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

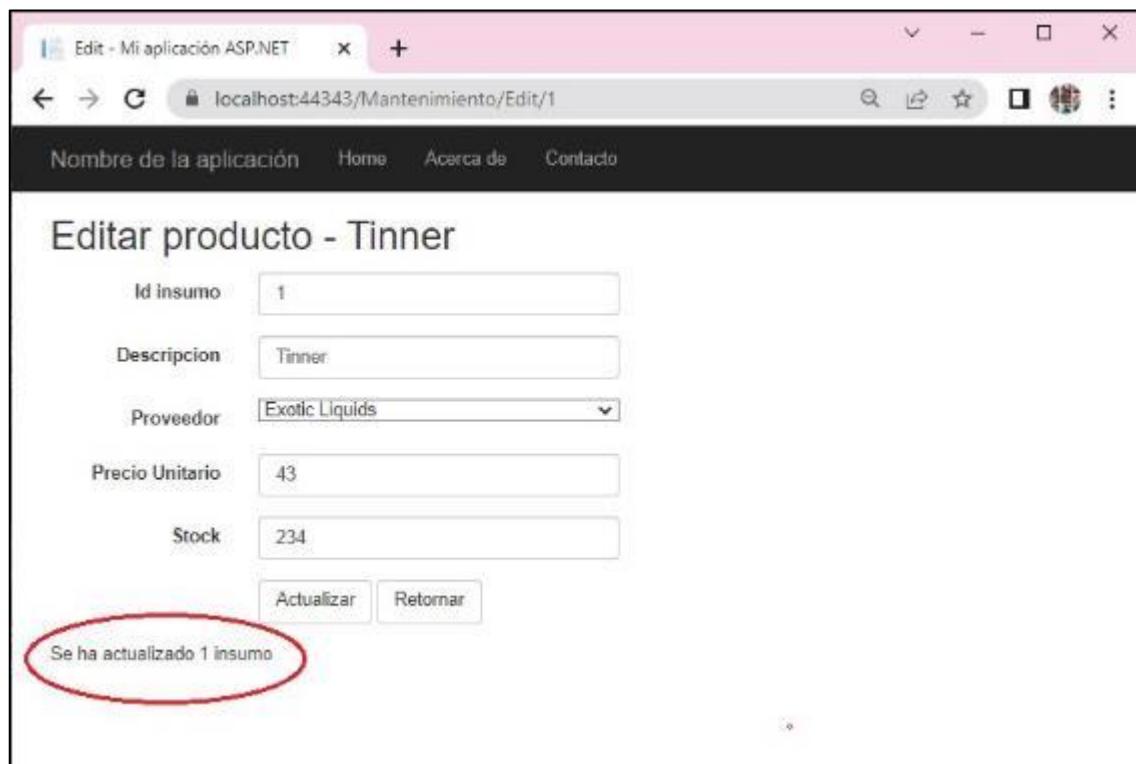
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Actualizar" class="btn btn-default" />
            <@Html.ActionLink("Retornar", "Index", null, new { @class = "btn btn-default" })>
        </div>
    </div>
</div>
</div>
```

<div> @ViewBag.mensaje </div>

Nota. Elaboración propia.

Ejecute la Vista Index, presiona la tecla F5, selecciona un insumo para visualizar los datos del registro, modifique los valores, al presionar el botón Actualizar, se ejecuta el proceso visualizando un mensaje: “se ha actualizado 1 insumo”.

Figura 182
Desarrollo práctico



Nota. Elaboración propia.

Programando el ActionResult Details

Defina el ActionResult Details (GET) donde busca el registro de tb_insumo por su campo idinsumo. Defina un parámetro opcional llamado id cuyo valor inicial es null.

Figura 183
Desarrollo práctico

```

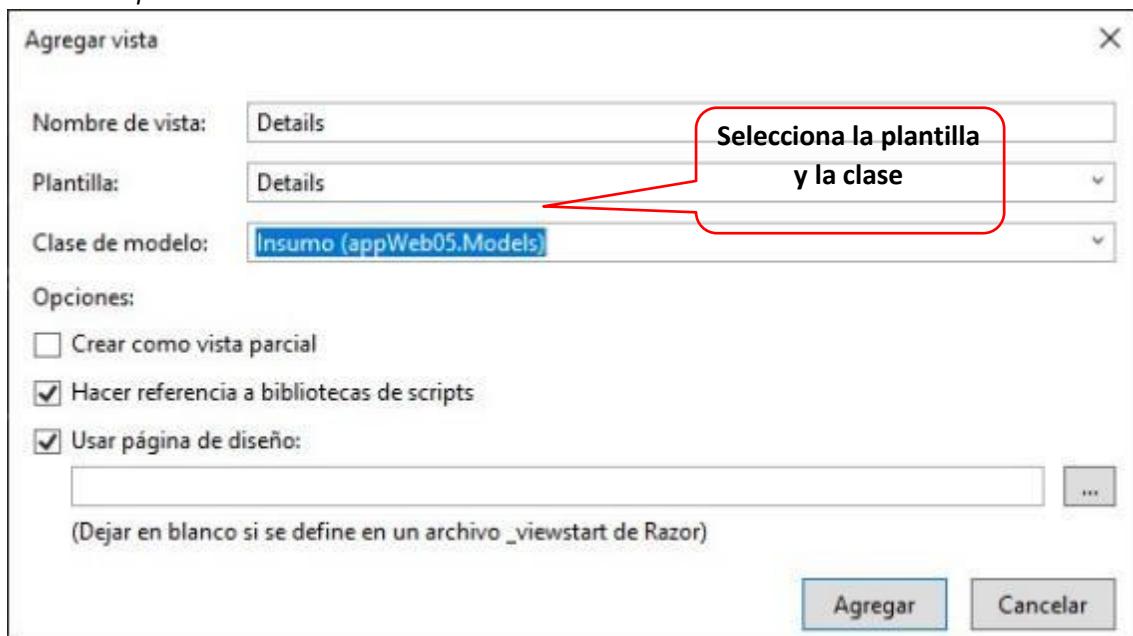
14 0 referencias
15 public class MantenimientoController : Controller
16 {
17     0 referencias
18     public ActionResult Details(int ? id = null)
19     {
20         if (id == null)
21         {
22             return RedirectToAction("Index");
23         }
24
25         Insumo reg=insumos().FirstOrDefault(x => x.idinsumo== id);
26         return View(reg);
27     }
28 }

```

Nota. Elaboración propia.

A continuación, agregar la vista al ActionResult Details, tal como se muestra.

Figura 184
Desarrollo práctico



Nota. Elaboración propia.

En la vista Details, agregar en el título la descripción del insumo.

Figura 185
Desarrollo práctico

```

@model appWeb05.Models.Insumo

@{ ViewBag.Title = "Details"; }

<h2>Detalle del Producto - @Model.descripcion</h2>

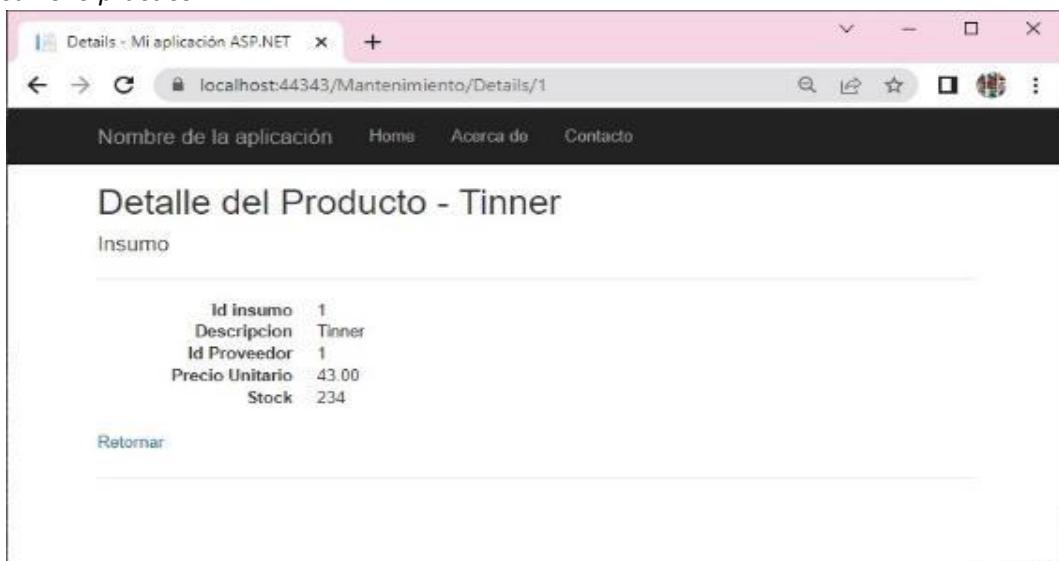
<div>
    <h4>Insumo</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>@Html.DisplayNameFor(model => model.idinsumo)</dt>
        <dd>@Html.DisplayFor(model => model.idinsumo)</dd>
        <dt>@Html.DisplayNameFor(model => model.descripcion)</dt>
        <dd>@Html.DisplayFor(model => model.descripcion)</dd>
        <dt>@Html.DisplayNameFor(model => model.idproveedor)</dt>
        <dd>@Html.DisplayFor(model => model.idproveedor)</dd>
        <dt>@Html.DisplayNameFor(model => model.precio)</dt>
        <dd>@Html.DisplayFor(model => model.precio)</dd>
        <dt>@Html.DisplayNameFor(model => model.stock)</dt>
        <dd>@Html.DisplayFor(model => model.stock)</dd>
    </dl>
</div>
<p>
    <a href="#">@Html.ActionLink("Retornar", "Index")</a>
</p>

```

Nota. Elaboración propia.

Ejecute la Vista, presiona la tecla F5, seleccione un producto desde la Vista Index, donde se visualiza los datos del insumo seleccionado.

Figura 186
Desarrollo práctico



Nota. Elaboración propia.

Programando el ActionResult Delete

En el controlador Negocios, defina el método eliminar el cual ejecuta el procedimiento almacenado donde elimina un registro por el campo idinsumo.

Figura 187
Desarrollo práctico

```

public class MantenimientoController : Controller
{
    public ActionResult Eliminar(int id)
    {
        string mensaje = "";
        using (SqlConnection cn = new SqlConnection(
            ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
        {
            cn.Open();
            SqlCommand cmd = new SqlCommand("exec usp_elimina_insumo @id", cn);
            cmd.Parameters.AddWithValue("@id", id);

            int i = cmd.ExecuteNonQuery();
            mensaje = $"Se ha eliminado {i} insumo";
        }
        return mensaje;
    }
}

```

Nota. Elaboración propia.

El ActionResult Delete el cual elimina el registro del Insumo por su campo idinsumo. Al finalizar redirecciona al Action Index.

Figura 188
Desarrollo práctico

```
14 public class MantenimientoController : Controller
15 {
16     [metodos]
17     0 referencias
18     1 referencia
19     string Eliminar(int id){...}
20     0 referencias
21     public ActionResult Delete(int ? id = null)
22     {
23         if(id==null)
24             return RedirectToAction("Index");
25
26         ViewBag.mensaje = Eliminar(id.Value);
27         return RedirectToAction("Index");
28     }
29 }
```

Nota. Elaboración propia.

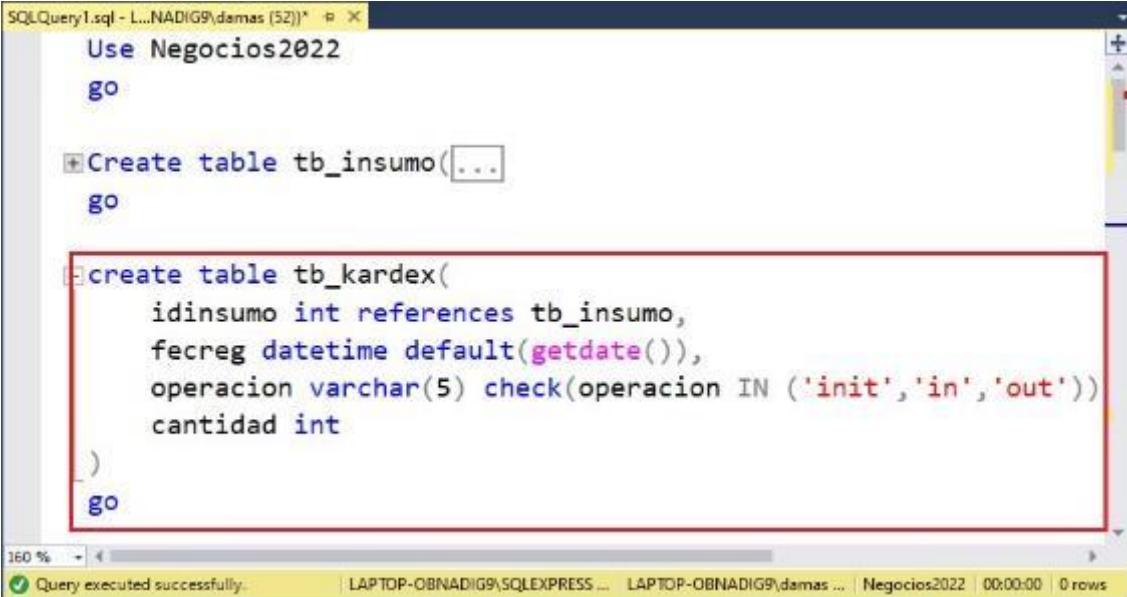
Ejecute la Vista Index, presiona la tecla F5, selecciona un insumo para eliminar el registro.

LABORATORIO 5.2.: Manejo de Transacciones

Se desea implementar un proyecto en ASP.NET MVC donde permita insertar y consultas los datos de la tabla tb_insumo y tb_kardex almacenado en una base de datos en SQL Server. Implemente una transacción en el proceso de registrar un nuevo Insumo registrando un registro a la tabla tb_kardex.

Creando la tabla y procedimientos almacenados en SQL Server

Creando la tabla tb_kardex la cual hace referencia a la tabla tb_insumo.

Figura 189*Trabajando en SQL Server*


```

SQLQuery1.sql - L...NADIG9\damas (52)*  X
Use Negocios2022
go

+Create table tb_insumo([...]
go

+create table tb_kardex(
    idinsumo int references tb_insumo,
    fecreg datetime default(getdate()),
    operacion varchar(5) check(operacion IN ('init','in','out'))
    cantidad int
)
go

```

The screenshot shows a SQL query window in SSMS. The code is for creating a table named tb_kardex. The entire section from the first 'create table' statement to the closing 'go' is highlighted with a red rectangle.

Nota. Elaboración propia.

Creando el procedure para Insertar un registro a la tabla tb_kardex.

Figura 190*Trabajando en SQL Server*


```

SQLQuery1.sql - L...NADIG9\damas (52)*  X
Use Negocios2022
go

+create table tb_kardex([...]
go

+create or alter proc usp_inserta_kardex
    @id int,
    @opera varchar(5),
    @cantidad int
    As
+Insert into tb_kardex(idinsumo,operacion,cantidad)
    Values(@id,@opera,@cantidad)
    go

```

The screenshot shows a SQL query window in SSMS. The code is for creating a stored procedure named usp_inserta_kardex. The entire section from the 'create or alter proc' statement to the final 'go' is highlighted with a red rectangle.

Nota. Elaboración propia.

Creando el procedure para listar los insumos donde incluya los datos de la tabla tb_kardex y tb_proveedores.

Figura 191*Trabajando en SQL Server*

```

SQLQuery1.sql - L..NADIG9\damas (52)  X
Use Negocios2022
go

create or alter proc usp_inventario
As
Select i.idinsumo,nominsumo, NombreCia,operacion, preUni,cantidad
from tb_insumo i join tb_kardex k on i.idinsumo=k.idinsumo
join tb_proveedores p on i.idproveedor=p.IdProveedor
go

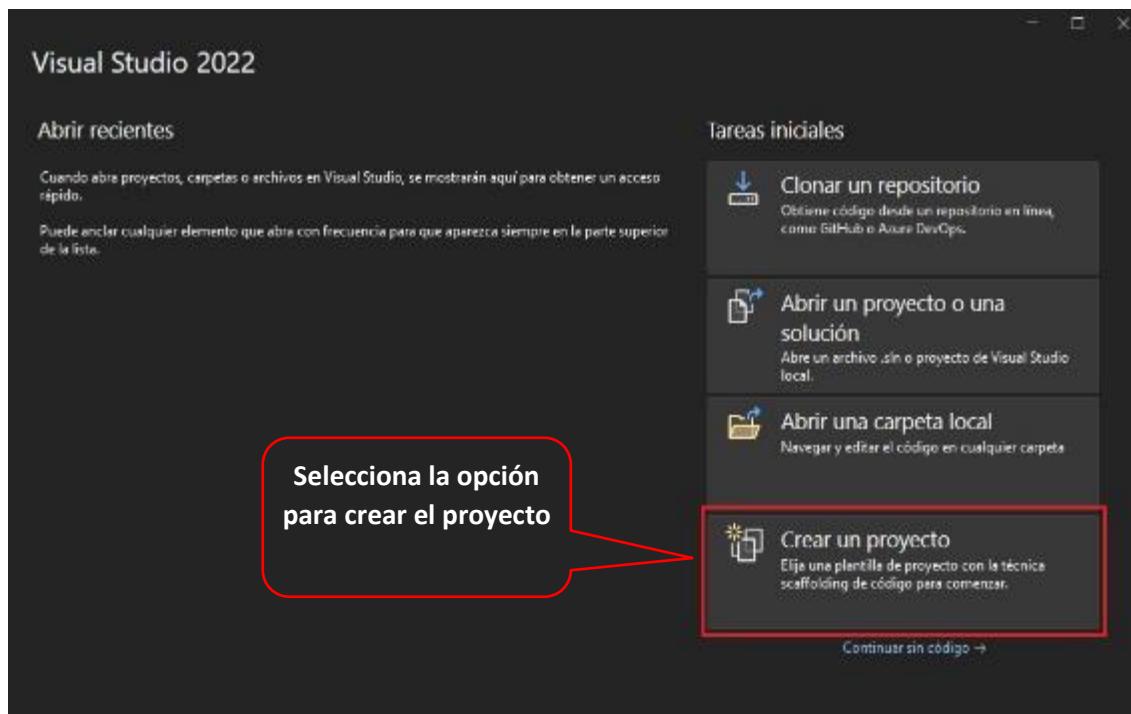
```

160 % 4

Query executed successfully. LAPTOP-OBNAJDIG9\damas ... Negocios2022 00:00:00 0 rows

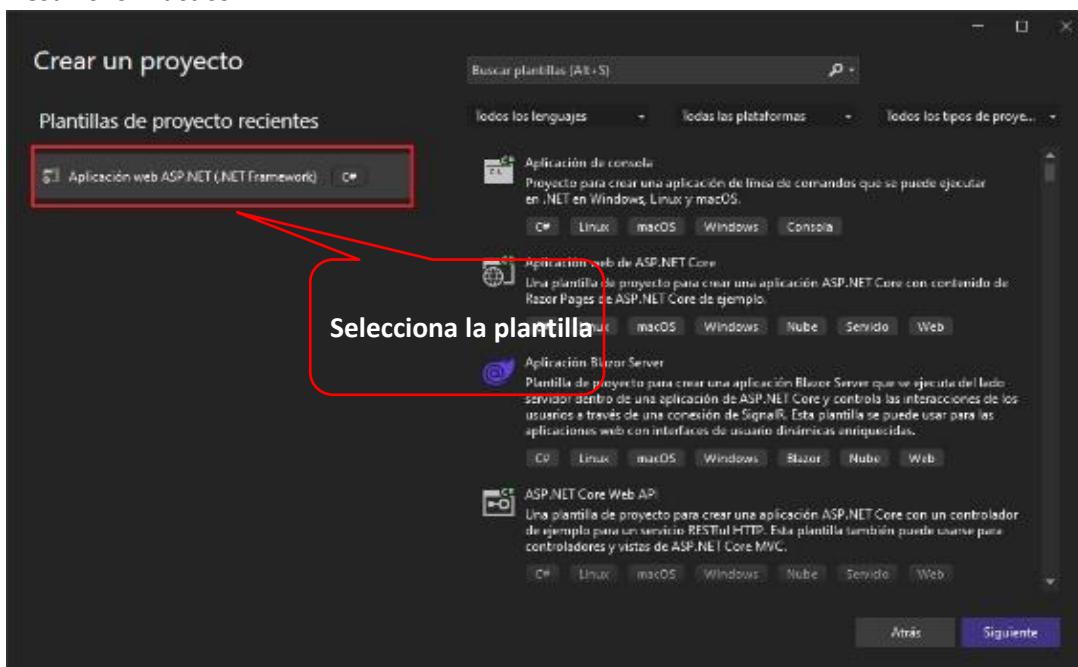
*Nota. Elaboración propia.***Creando un Proyecto ASP.NET MVC****Inicio del Proyecto**

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

Figura 192*Desarrollo Práctico**Nota. Elaboración propia.*

Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción Siguiente.

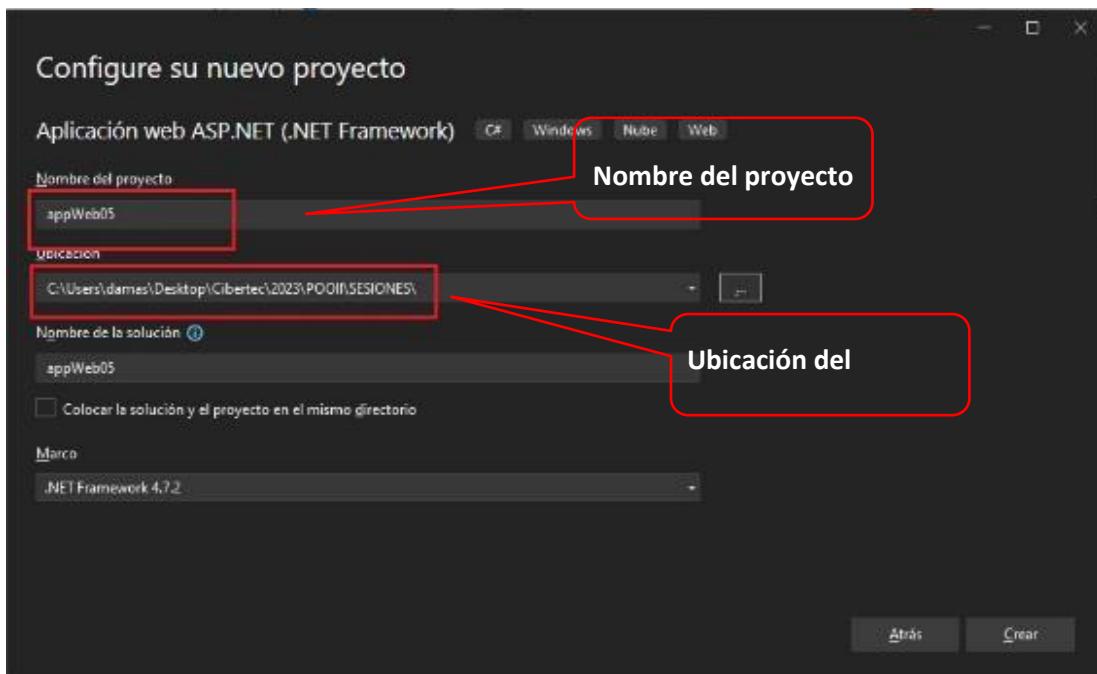
Figura 193
Desarrollo Práctico



Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciona la ubicación del mismo, al terminar presiona la opción **Crear**.

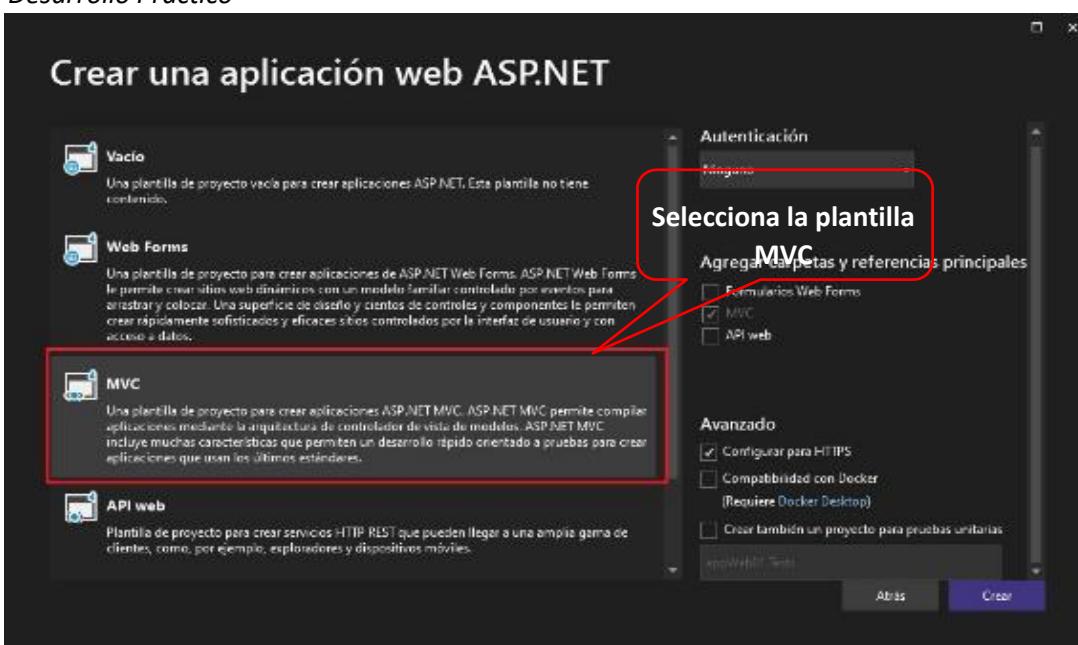
Figura 194
Desarrollo Práctico



Nota. Elaboración propia.

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

Figura 195
Desarrollo Práctico

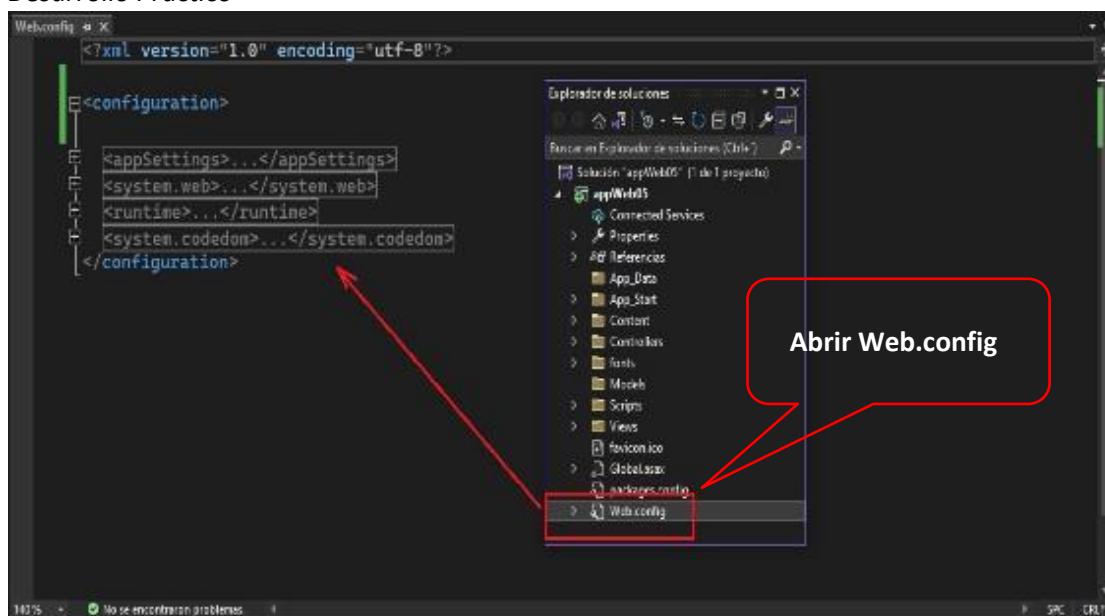


Nota. Elaboración propia.

Publicando la cadena de conexión

Desde el Explorador de proyecto, abrir el archivo Web.config, tal como se muestra.

Figura 196
Desarrollo Práctico



Nota. Elaboración propia.

Defina la etiqueta `<connectionStrings>` agregando una cadena `<add>` cuyo nombre es "cadena" y definiendo la cadena de conexión, tal como se muestra.

Figura 197
Desarrollo Práctico

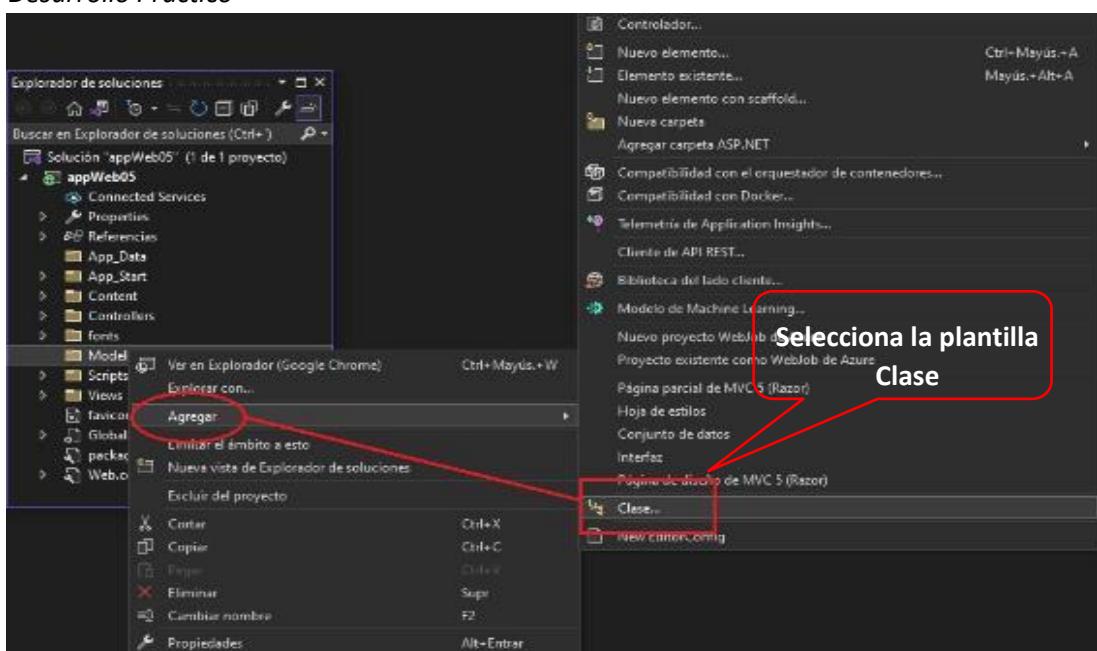
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="cadena"
      connectionString="server=.; database=Negocios2022; integrated security=true"/>
  </connectionStrings>
  <appSettings>...</appSettings>
  <system.web>...</system.web>
  <runtime>...</runtime>
  <system.codedom>...</system.codedom>
</configuration>
```

Nota. Elaboración propia.

Agregando la clase Insumo a la carpeta Models

En la carpeta Models, hacer clic derecho sobre la carpeta, seleccionar Agregar → Clase, tal como se muestra en la figura.

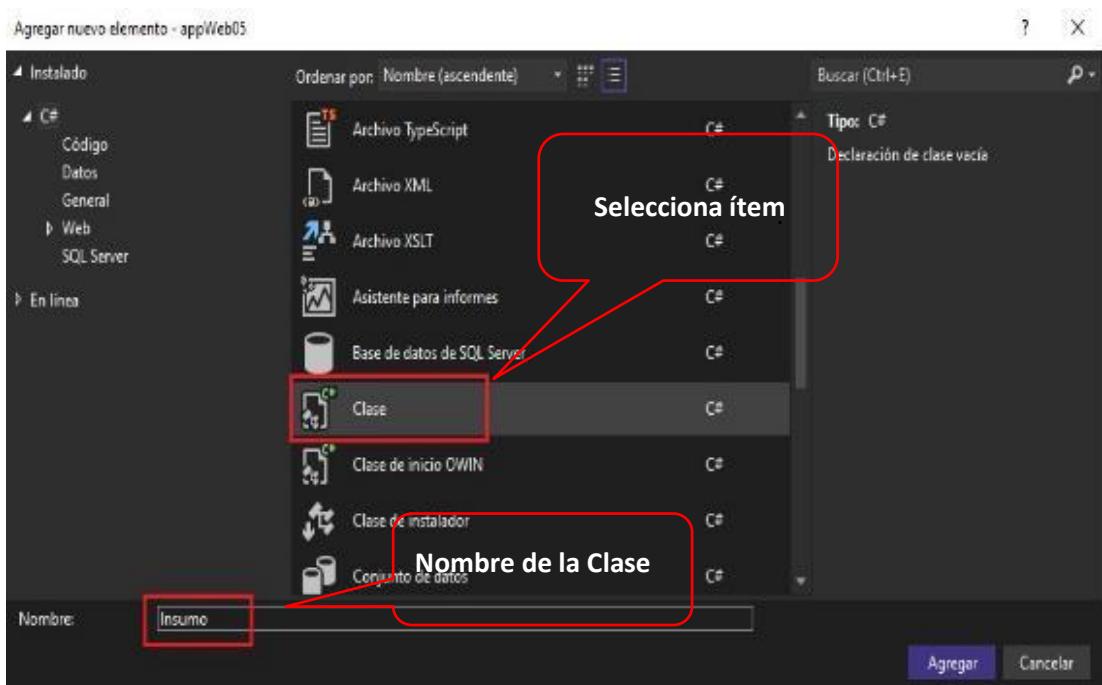
Figura 198
Desarrollo Práctico



Nota. Elaboración propia.

Selecciona el elemento Clase, asigne el nombre del elemento: Insumo, tal como se muestra.

Figura 199
Desarrollo Práctico



Nota. Elaboración propia.

Agregar la librería System.ComponentModel.DataAnnotations y atributos de la clase, cada atributo será de ingreso obligatorio asignándole a cada uno la etiqueta **Required**, tal como se muestra.

Figura 200
Desarrollo Práctico

```

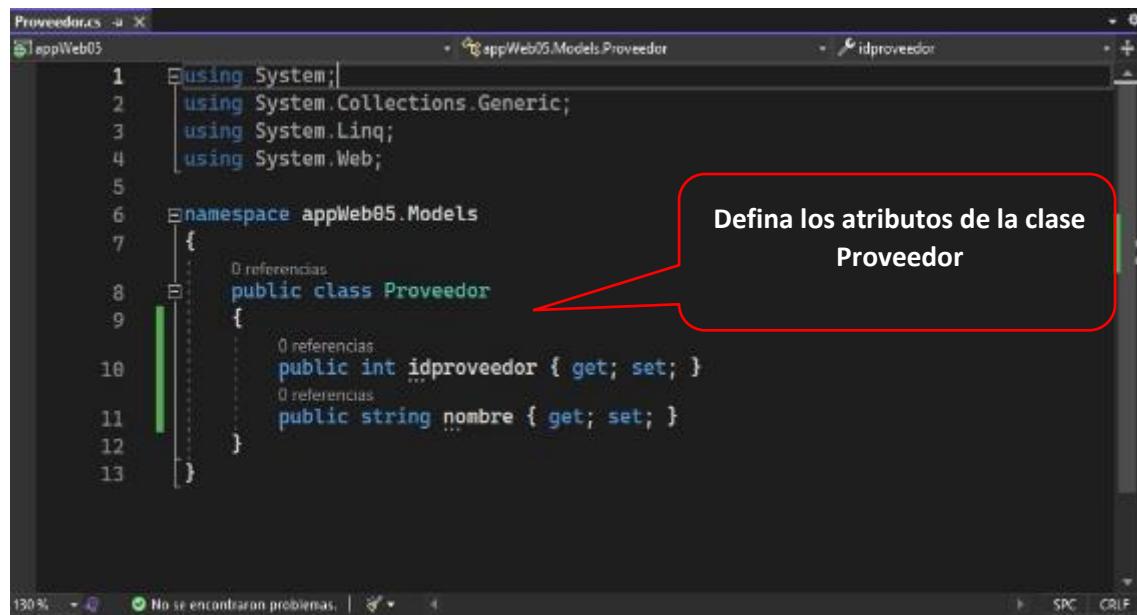
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.ComponentModel.DataAnnotations;
6  namespace appWeb05.Models
7  {
8      public class Insumo
9      {
10         [Display(Name = "Id insumo"), Required] public int idinsumo { get; set; }
11         [Display(Name = "Descripción"), Required] public string descripcion { get; set; }
12         [Display(Name = "Id Proveedor"), Required] public int idproveedor { get; set; }
13         [Display(Name = "Precio Unitario"), Required] public decimal precio { get; set; }
14         [Display(Name = "Stock"), Required] public int stock { get; set; }
15     }
16 }

```

Nota. Elaboración propia.

En la carpeta Models, agrega la clase Proveedor, defina los atributos de la clase, tal como se muestra.

Figura 201
Desarrollo Práctico

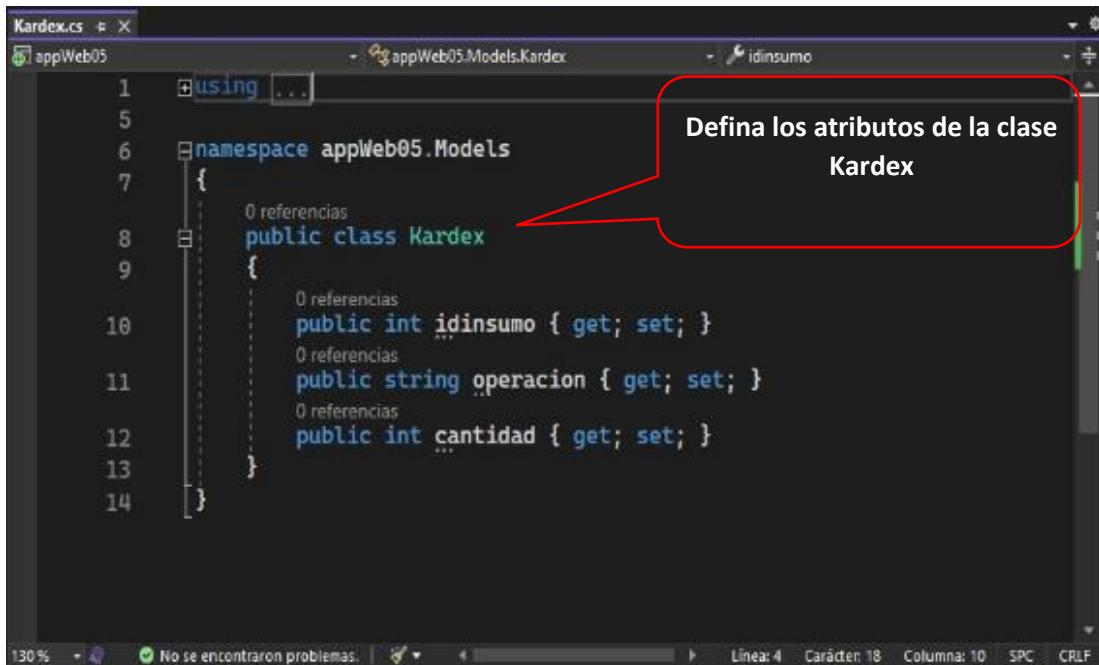


```
Proveedor.cs - x
appWeb05          appWeb05.Models.Proveedor      idproveedor
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace appWeb05.Models
7  {
8      public class Proveedor
9      {
10         public int idproveedor { get; set; }
11         public string nombre { get; set; }
12     }
13 }
```

Nota. Elaboración propia.

En la carpeta Models, agrega la clase Kardex, defina los atributos de la clase, tal como se muestra.

Figura 202
Desarrollo Práctico



```
Kardex.cs - x
appWeb05          appWeb05.Models.Kardex      idinsumo
1  using ...
5
6  namespace appWeb05.Models
7  {
8      public class Kardex
9      {
10         public int idinsumo { get; set; }
11         public string operacion { get; set; }
12         public int cantidad { get; set; }
13     }
14 }
```

Nota. Elaboración propia.

En la carpeta Models, agrega la clase Almacén, defina los atributos de la clase, tal como se muestra.

Figura 203
Desarrollo Práctico

```

Almacen.cs  x
appWeb05      appWeb05.Models.Almacen
1  using ...
5
6  namespace appWeb05.Models
7  {
8      public class Almacen
9      {
10         public int idinsumo { get; set; }
11         public string descripcion { get; set; }
12         public string proveedor { get; set; }
13         public string operacion { get; set; }
14         public decimal preuni { get; set; }
15         public int cantidad { get; set; }
16     }
17 }

```

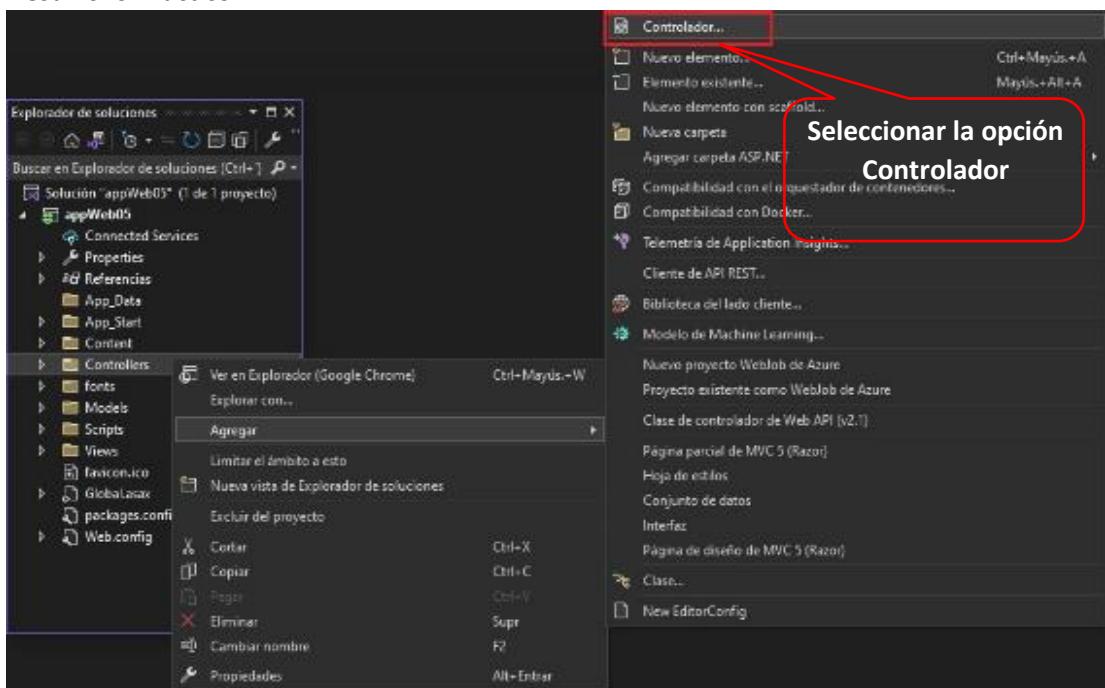
130% No se encontraron problemas. Línea: 4 Carácter: 18 Columna: 10 SPC CRLF

Nota. Elaboración propia.

Trabajando con el Controlador

En la carpeta Controllers, agregamos un controlador, tal como se muestra.

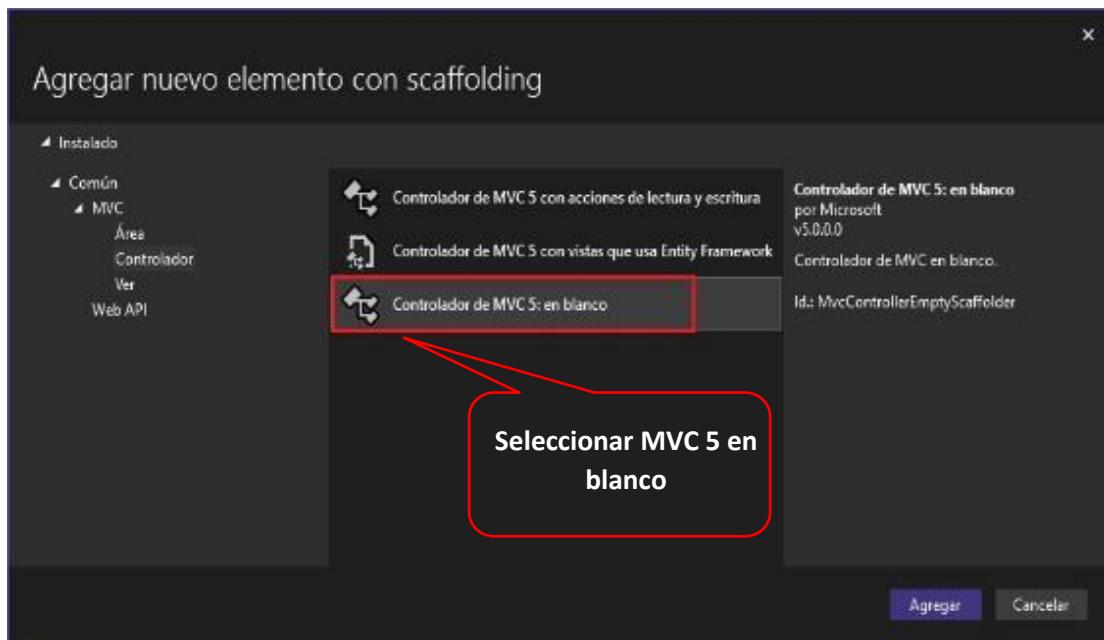
Figura 204
Desarrollo Práctico



Nota. Elaboración propia.

Selecciona Controlador MVC5 en blanco, tal como se muestra.

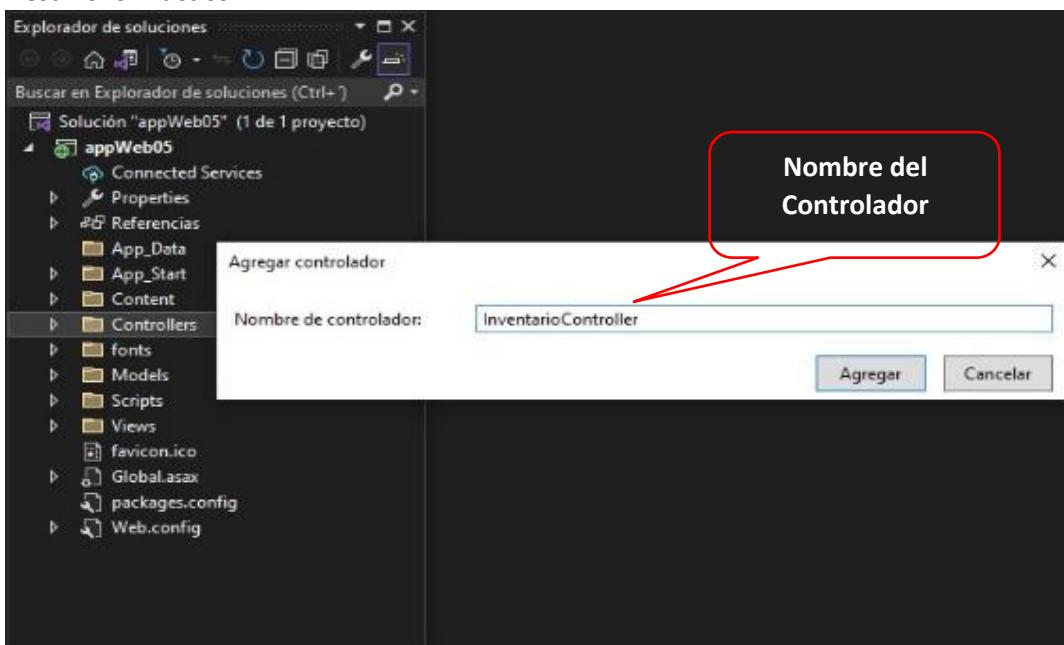
Figura 205
Desarrollo Práctico



Nota. Elaboración propia.

Asigne el nombre de NegociosController, tal como se muestra. Presiona el botón Agregar.

Figura 206
Desarrollo Práctico



Nota. Elaboración propia.

Programando el Controlador

Importar las librerías y la carpeta Models, donde se encuentra almacenado la clase.

Figura 207*Desarrollo Práctico*

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using System.Configuration;
7  using System.Data;
8  using System.Data.SqlClient;
9  using appWeb05.Models;
10 namespace appWeb05.Controllers
11 {
12     public class InventarioController : Controller
13     {
14     }
15 }

```

Nota. Elaboración propia.

En el Controlador, defina y codifique el método proveedores() donde retorna la lista numerada de los registros de tb_proveedores, tal como se muestra.

Figura 208*Desarrollo Práctico*

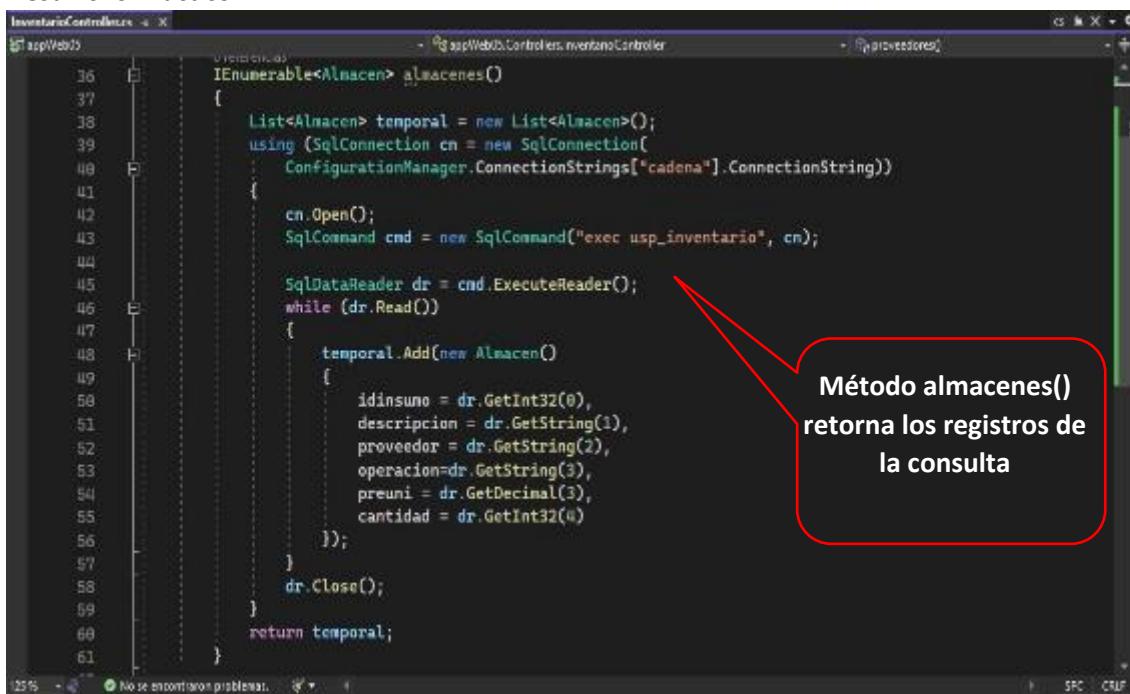
```

40 Ienumerable<Proveedor> proveedores()
41 {
42     List<Proveedor> temporal = new List<Proveedor>();
43     using (SqlConnection cn = new SqlConnection(
44         ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
45     {
46         cn.Open();
47         SqlCommand cmd = new SqlCommand("exec usp_proveedores", cn);
48
49         SqlDataReader dr = cmd.ExecuteReader();
50         while (dr.Read())
51         {
52             temporal.Add(new Proveedor()
53             {
54                 idproveedor = dr.GetInt32(0),
55                 nombre = dr.GetString(1)
56             });
57         }
58         dr.Close();
59     }
60     return temporal;
61 }

```

Nota. Elaboración propia.

Defina y codifique el método almacenes() donde retorna la lista numerada de los registros del procedimiento almacenado usp_inventario, tal como se muestra.

Figura 209*Desarrollo Práctico*


```

InventarioController.cs < X
aspWeb5                                     - & gspWeb5.Controllers.InventarioController
  36  | IEnumarable<Almacen> almacenes()
  37  | {
  38  |     List<Almacen> temporal = new List<Almacen>();
  39  |     using (SqlConnection cn = new SqlConnection(
  40  |         ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
  41  |     {
  42  |         cn.Open();
  43  |         SqlCommand cmd = new SqlCommand("exec usp_inventario", cn);
  44  |
  45  |         SqlDataReader dr = cmd.ExecuteReader();
  46  |         while (dr.Read())
  47  |         {
  48  |             temporal.Add(new Almacen()
  49  |             {
  50  |                 idinsuno = dr.GetInt32(0),
  51  |                 descripcion = dr.GetString(1),
  52  |                 proveedor = dr.GetString(2),
  53  |                 operacion=dr.GetString(3),
  54  |                 preuni = dr.GetDecimal(3),
  55  |                 cantidad = dr.GetInt32(4)
  56  |             });
  57  |         }
  58  |         dr.Close();
  59  |     }
  60  |     return temporal;
  61  | }

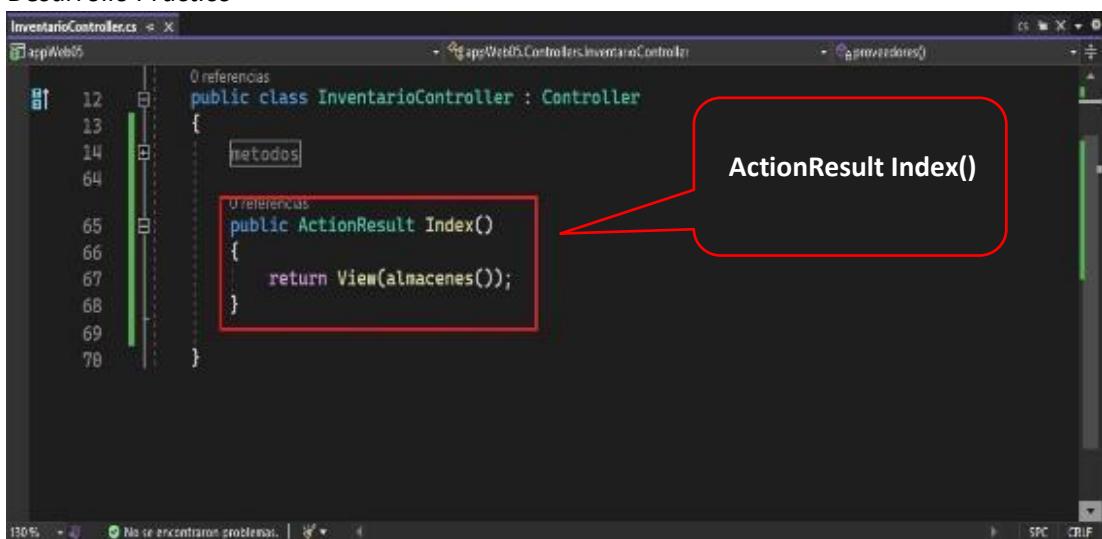
```

Nota. Elaboración propia.

Método almacenes()
retorna los registros de
la consulta

Trabajando con el ActionResult Index

Defina el ActionResult Index(y), el cual enviará a la Vista el resultado del método almacenes() tal como se muestra.

Figura 210*Desarrollo Práctico*


```

InventarioController.cs < X
aspWeb5                                     - & gspWeb5.Controllers.InventarioController
  12  | 0 referencias
  13  | public class InventarioController : Controller
  14  | {
  15  |     metodos
  16  |
  17  |     0 referencias
  18  |     public ActionResult Index()
  19  |     {
  20  |         return View(almacenes());
  21  |     }
  22  | }

```

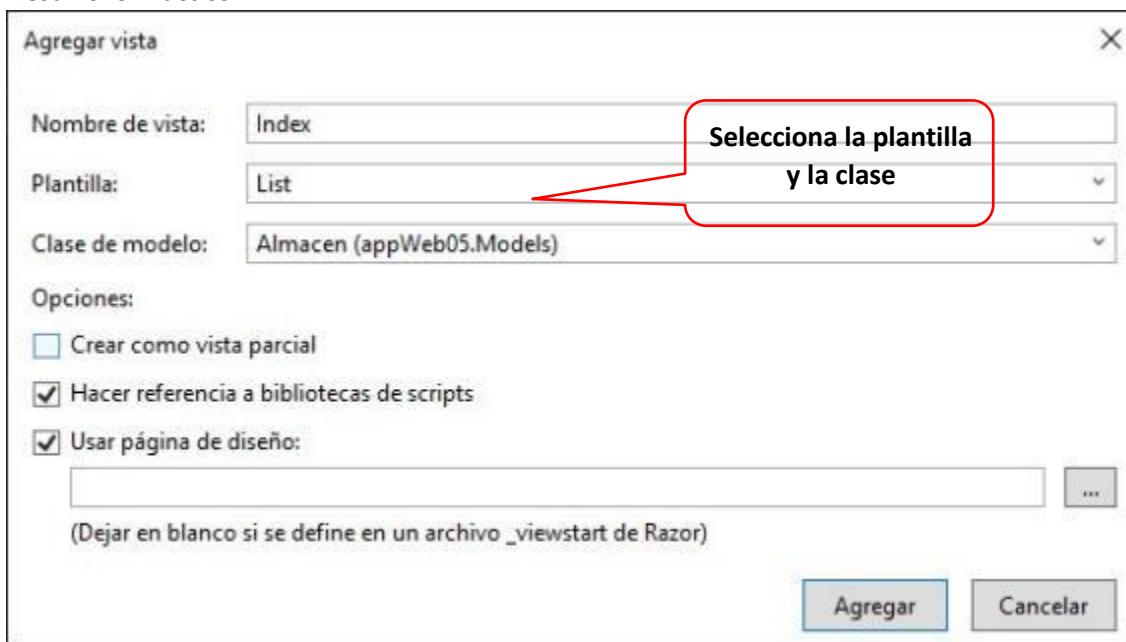
Nota. Elaboración propia.

ActionResult Index()

Trabajando con la Vista Index

En el ActionResult, hacer clic derecho y selecciona, Agregar vista. Seleccione la plantilla, la cual será List; y la clase de modelo la cual es Almacén, tal como se muestra.

Figura 211
Desarrollo Práctico



Nota. Elaboración propia.

Vista generada por la plantilla List, modifique el ActionLink del Create, tal como se muestra.

Figura 212
Desarrollo Práctico

```

Index.cshtml  X  Inventario.Controllers
@model IEnumerable<appWeb05.Models.Almacen>
@{
    ViewBag.Title = "Index";
}
<h2>Inventario de Insumos</h2>

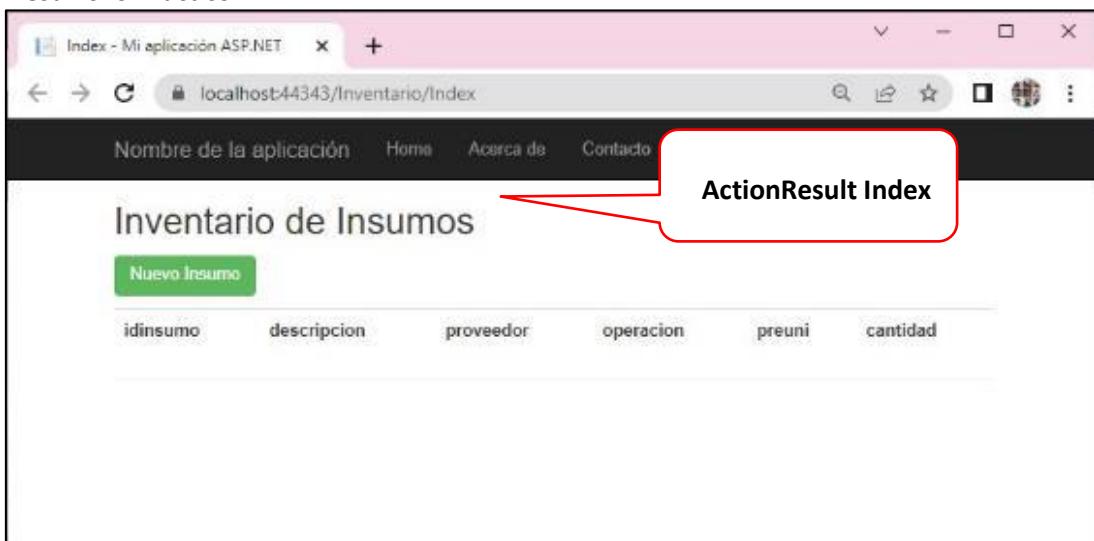
<p>
    @Html.ActionLink("Nuevo Insumo", "Create", null, new { @class = "btn btn-success" })
</p>
<table class="table">
    <thead>
        <tr>
            <th>@Html.DisplayNameFor(model => model.idinsumo)</th>
            <th>@Html.DisplayNameFor(model => model.descripcion)</th>
            <th>@Html.DisplayNameFor(model => model.proveedor)</th>
            <th>@Html.DisplayNameFor(model => model.operacion)</th>
            <th>@Html.DisplayNameFor(model => model.preuni)</th>
            <th>@Html.DisplayNameFor(model => model.cantidad)</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model) {
            <tr>
                <td>@Html.DisplayFor(modelItem => item.idinsumo)</td>
                <td>@Html.DisplayFor(modelItem => item.descripcion)</td>
                <td>@Html.DisplayFor(modelItem => item.proveedor)</td>
                <td>@Html.DisplayFor(modelItem => item.operacion)</td>
                <td>@Html.DisplayFor(modelItem => item.preuni)</td>
                <td>@Html.DisplayFor(modelItem => item.cantidad)</td>
            </tr>
        }
    </tbody>
</table>

```

Nota. Elaboración propia.

Presiona la tecla F5 para ejecutar el ActionResult Index y visualizar la Vista.

Figura 213
Desarrollo Práctico



Nota. Elaboración propia.

Programando el ActionResult Create

En el Controlador, defina el método agregar el cual ejecuta dos procedimientos almacenados controlándolos por una transacción, donde retorna el mensaje del proceso.

Figura 214
Desarrollo Práctico

The screenshot shows a code editor with a red callout box highlighting the text 'Trabajando con transacción'. The code is written in C# and defines a method 'Agregar' in a controller named 'InventarioController'. The method uses a transaction to execute two stored procedures: 'usp_inserta_insumo' and 'usp_inserta_kardex'. It handles exceptions and returns a message indicating successful registration.

```

64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
  string Agregar(Insurno reg)
  {
    string mensaje = "";
    SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["cadena"].ConnectionString);
    cn.Open();
    SqlTransaction tr = cn.BeginTransaction(IsolationLevel.Serializable);
    try
    {
      SqlCommand cmd = new SqlCommand("exec usp_inserta_insumo @id,@nombre,@idproveedor,@pre,@stock", cn,tr);
      cmd.Parameters.AddWithValue("@id", reg.idinsumo);
      cmd.Parameters.AddWithValue("@nombre", reg.descripcion);
      cmd.Parameters.AddWithValue("@idproveedor", reg.idproveedor);
      cmd.Parameters.AddWithValue("@pre", reg.precio);
      cmd.Parameters.AddWithValue("@stock", reg.stock);
      cmd.ExecuteNonQuery();

      cmd = new SqlCommand("exec usp_inserta_kardex @id,@opera,@cantidad", cn, tr);
      cmd.Parameters.AddWithValue("@id", reg.idinsumo);
      cmd.Parameters.AddWithValue("@opera", "init");
      cmd.Parameters.AddWithValue("@cantidad", reg.stock);
      cmd.ExecuteNonQuery();

      tr.Commit();
      mensaje = "Se ha registrado el Insumo y su tarjeta";
    }
    catch(SqlException ex){
      mensaje = ex.Message;
      tr.Rollback();
    }
    finally{ cn.Close(); }
    return mensaje;
  }

```

Nota. Elaboración propia.

Defina el ActionResult Create (GET) donde envía la lista de los proveedores a la vista y un nuevo Insumo, tal como se muestra. En el POST ejecuta el método Agregar almacenando el resultado en un ViewBag, refrescando la vista con los datos del registro ingresado.

Figura 215
Desarrollo Práctico

```

InventoryController.cs  E
aspWeb05
  ↗aspWeb05\Controllers\InventarioController.cs

12  E
13
14
101
102  E
103
104  ViewBag.proveedores = new SelectList(proveedores(), "idproveedor", "nombre");
105  return View(new Insumo());
106
107  [HttpPost]
108  E
109
110  public ActionResult Create(Insumo reg)
111  {
112    ViewBag.mensaje = Agregar(reg);
113    ViewBag.proveedores = new SelectList(proveedores(), "idproveedor", "nombre", reg.idproveedor);
114  }
  
```

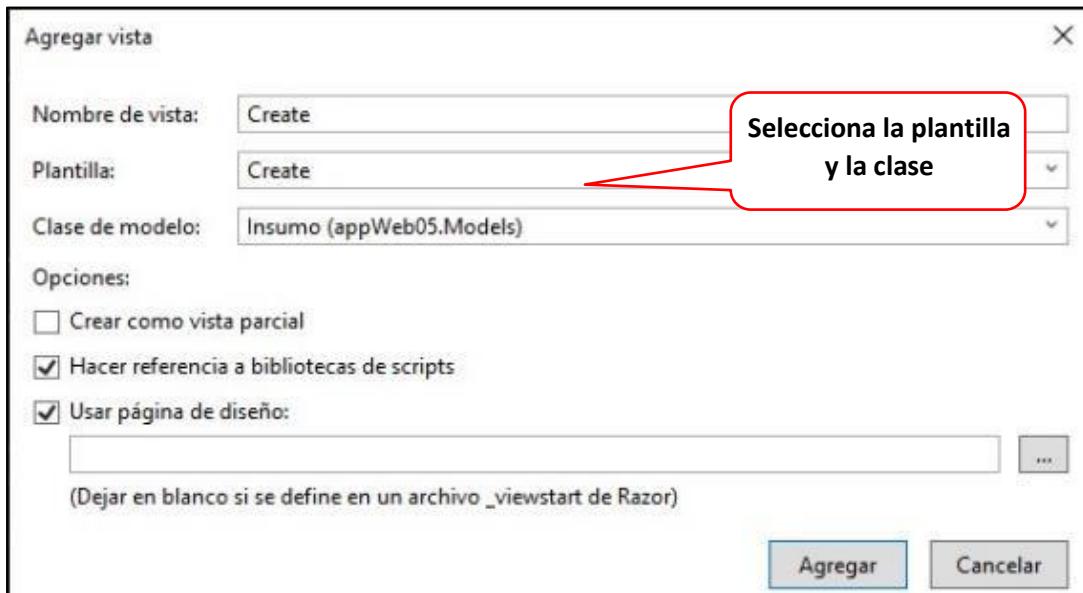
Método GET envía un nuevo Insumo()

Método POST ejecuta el método agregar()

Nota. Elaboración propia.

A continuación, agregar la vista al ActionResult Create, tal como se muestra.

Figura 216
Desarrollo Práctico



Nota. Elaboración propia.

En la vista Create, realizar los ajustes correspondientes: En el campo idproveedor, cambiar al helper DropDownList, agregar 2 ActionLink: Nuevo, Retorna y agregar el ViewBag.mensaje.

Figura 217*Desarrollo Práctico*

```

@using (Html.BeginForm())
{
    <div class="form-horizontal">
        <div class="form-group"></div>
        <div class="form-group"></div>

        <div class="form-group">
            <div class="col-md-18">
                @Html.DropDownList("idproveedor", ViewBag.proveedores as SelectList, new { @class = "select-control" })
            </div>
        </div>

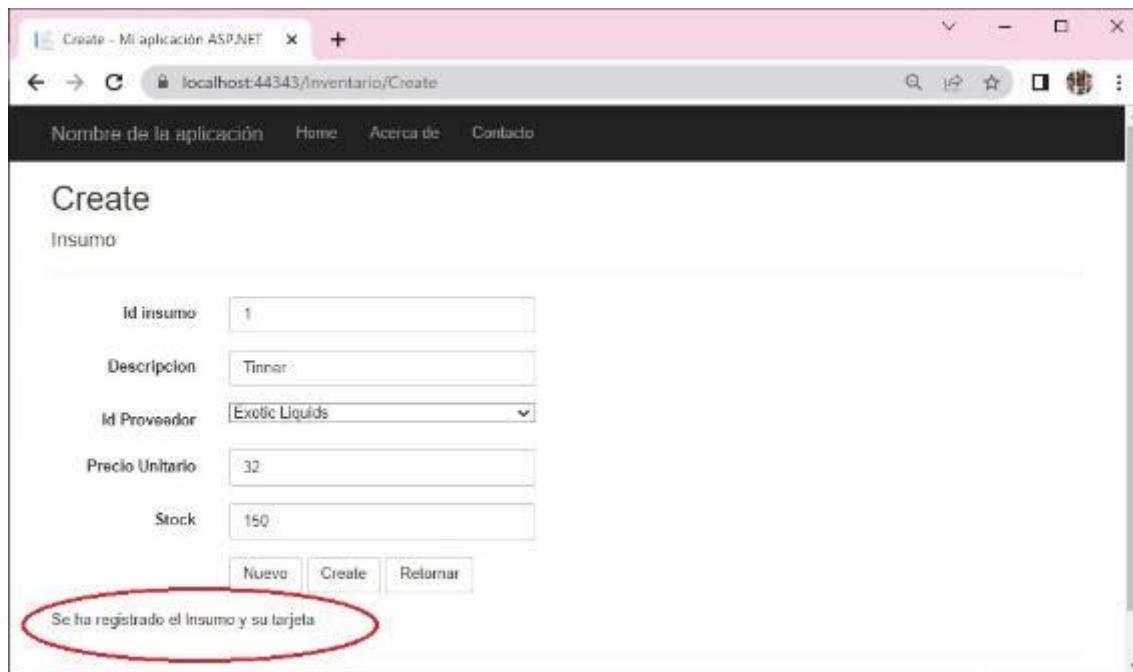
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
                @Html.ActionLink("Nuevo", "Create", null, new { @class = "btn btn-default" })
                @Html.ActionLink("Retornar", "Index", null, new { @class = "btn btn-default" })
            </div>
        </div>
    </div>
}

<div>
    @ViewBag.mensaje
</div>

```

Nota. Elaboración propia.

Ejecute la Vista, presiona la tecla F5, ingrese los datos, al presionar el botón Create, se ejecuta el proceso visualizando un mensaje: “se ha insertado el insumo y su tarjeta”.

Figura 218*Desarrollo Práctico**Nota.* Elaboración propia.

Al presionar el botón Retornar, visualizamos la información del registro tb_insumo, el proveedor seleccionado y la operación que es “init”.

Figura 219

Desarrollo Práctico

IdInsumo	descripción	proveedor	operación	preuni	cantidad
1	Tinner	Exotic Liquids	init	32.00	150

Nota. Elaboración propia.

Resumen

1. Las instrucciones SQL INSERT, UPDATE o DELETE no devuelven ninguna fila. Para ejecutar comandos que no devuelvan filas, cree un objeto Command con el comando SQL adecuado y una Connection, incluidos los Parameters necesarios.
2. El espacio de nombres System.ComponentModel.DataAnnotations proporciona clases, atributos y métodos para realizar validación dentro de nuestros programas en .NET.
3. Para crear un SqlCommand, 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.
4. El método ExecuteReader permite ejecutar una consulta que retorna uno o más conjunto de resultados. Este método puede pasar un parámetro al método el cual representa la enumeración CommandType, que permite controlar al Command como se ejecutado.
5. 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.
6. Desarrollar una aplicación que nos permita subir archivos a un servidor desde un explorador Web es un proceso bastante sencillo, ya que la etiqueta `<input type="file" />` hace prácticamente todo el trabajo por nosotros en lo que se refiere a la parte del "front-end".
7. La clase HttpPostedFileBase le permite crear clases derivadas que son como la clase HttpPostedFile, pero que puede personalizar y que funcionan fuera de la tubería ASP.NET.
8. Una transacción consiste en un comando único o en un grupo de comandos que se ejecutan como un paquete.
9. 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.
10. En ADO, utilice el método BeginTrans en un objeto Connection para iniciar una transacción implícita. Para finalizar la transacción, llame a los métodos CommitTrans o RollbackTrans del objeto Connection.
11. En el proveedor administrado de SqlConnection de ADO.NET, utilice el método BeginTransaction en un objeto SqlConnection para iniciar una transacción. Para finalizar la transacción, llame a los métodos Commit() o Rollback() del objeto SqlTransaction.

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- https://www-csharp-com.translate.goog/article/execute-a-stored-procedureprogrammatically/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=sc
- <https://www.cjorellana.net/2011/12/cargar-y-guardar-una-imagen-en-la-base.html>
- <http://pabletoreto.blogspot.com/2015/12/guardar-mostrar-imagenes-en-aspnet.html>

- <https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/local-transactions>
- <https://learn.microsoft.com/es-es/dotnet/api/system.data.isolationlevel?view=net-6.0&viewFallbackFrom=dotnet-plat-ext-6.0>
- <https://learn.microsoft.com/es-es/dotnet/api/system.data.sqlclient.sqltransaction?view=net-9.0-pp&viewFallbackFrom=dotnet-plat-ext-%206.0>
- <https://learn.microsoft.com/es-es/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver16&viewFallbackFrom=sql-serverver15>

2.4. ARQUITECTURA DE CAPAS CON ACCESO A DATOS

2.4.1. Introducción

En una aplicación donde el diseño está orientado al dominio (Domain design Driven o DDD), término 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 está 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.

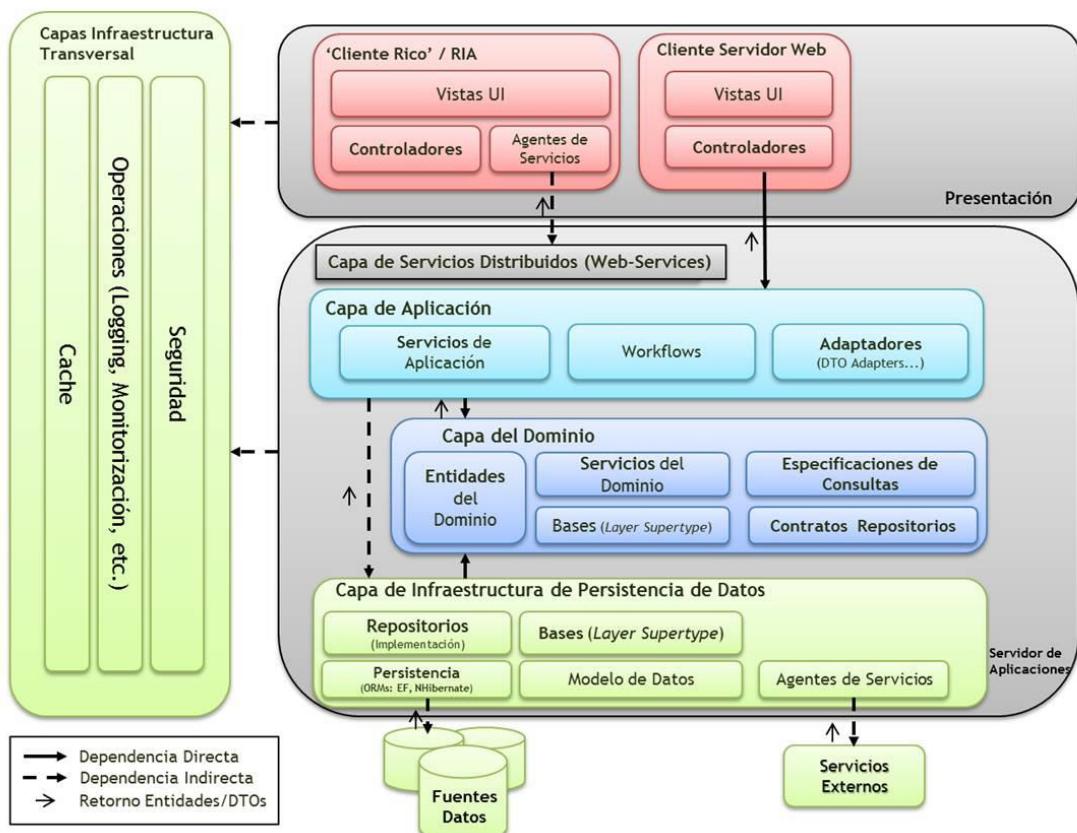
En un diseño orientado al dominio lo dependiente de la tecnología reside en el exterior como si capas de una cebolla fueran, donde puede sustituir una capa por otra utilizando otra tecnología y la funcionalidad de la aplicación no se ve comprometida.

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 220

Arquitectura de capas orientadas al dominio

Arquitectura N-Capas con Orientación al Dominio

Nota. Tomado de ASP: .NET MVC, por stackoverflow.com, 2023, (<http://es.stackoverflow.com/questions/41889/asp-net-mvc-arquitectura-ddd-domain-driven-design>)

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

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

Esta arquitectura permite estructurar de una forma limpia y clara la complejidad de una aplicación empresarial basada en las diferentes capas de la arquitectura, siguiendo el patrón NLayered 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

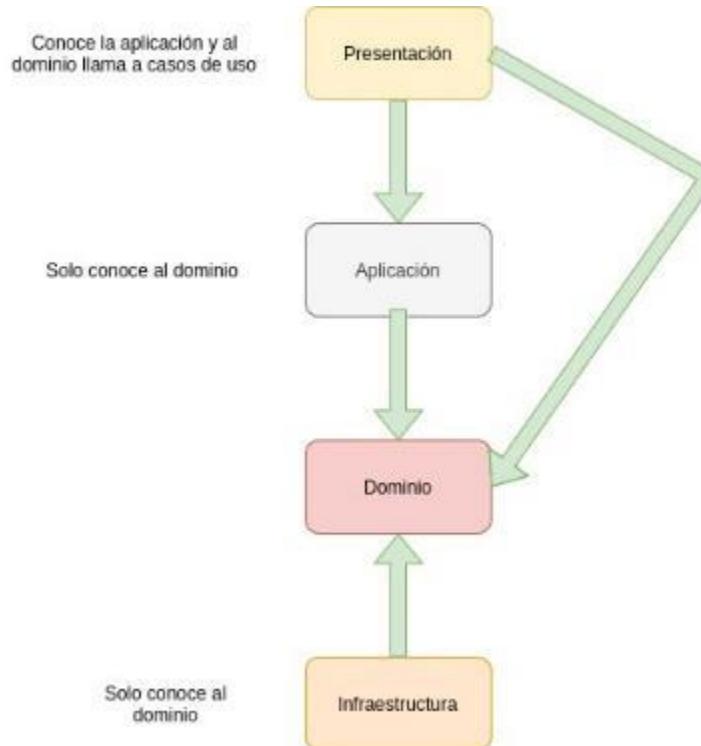
que sirva como punto base a ser modificada o adaptada por arquitectos según sus necesidades y requisitos.

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

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

Interacción en la Arquitectura DDD

Figura 221
Modelo de Capas



Nota. Tomado de *Introducción domain drive design*, por Refactorizando, 2021, refactorizando.com, (<https://refactorizando.com/introduccion-domain-drive-design/>)

Modelo de capas en DDD

Una arquitectura que podría perfectamente encajar con esa solución y se acopla muy bien al DDD, es la arquitectura hexagonal o de puertos y adaptadores.

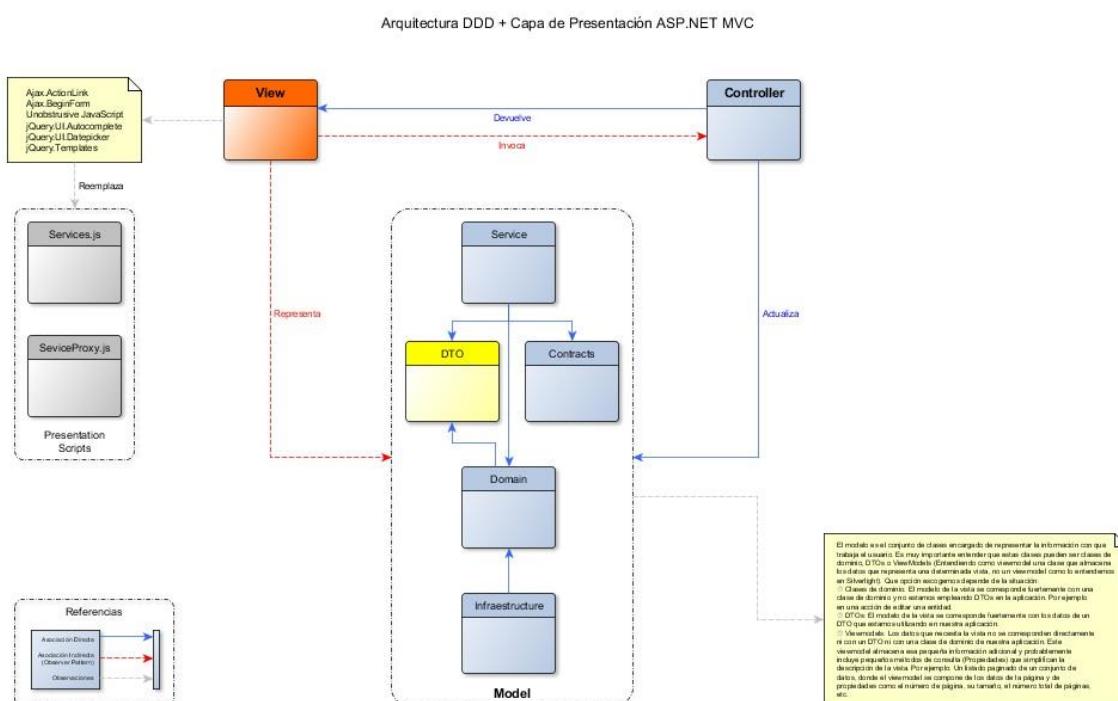
1. Capa de presentación: muestra la información al usuario.
2. Aplicación: esta capa define los casos de uso que son trasladados al software. Se intentará mantener tan simple como sea posible y se encargará de delegar las tareas de los objetos de dominio a la siguiente capa.
3. Dominio: la capa de dominio será la responsable de representar los conceptos del negocio, así como reglas y situaciones particulares. Hay que tener en cuenta que esta capa es la principal del negocio.
4. Infraestructura: esta capa permite la interacción entre las 4 capas a través de algún framework. Es en donde reside la parte técnica de la aplicación de coordinación (Servicios) de la Capa de Aplicación, puesto que es la parte que los orquesta.

2.4.2. Implementando una arquitectura de capas en un proceso de actualización de datos

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

Figura 222

Implementando Capas + aplicación MVC

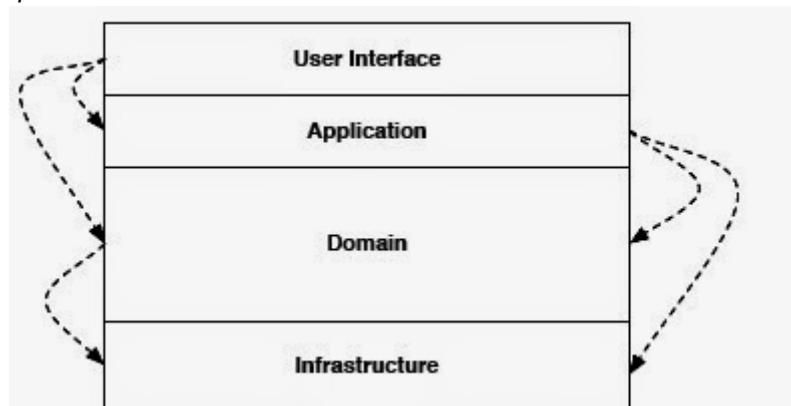


Nota. Tomado de Arquitectura DDD, por Wordpress, 2012, wordpress.com, (<https://jestruch.wordpress.com/2012/02/21/arquitectura-ddd-domain-driven-design-asp-net-mvc/>)

Capas de la arquitectura DDD para actualización de datos

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 punto de unión del 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 último 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 puede 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 se utilizan simplemente como clases de datos estaremos cayendo en el antipatrón de modelo de datos anémico.
 - b. **Domain:** en este proyecto, se tiene 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 se abordará la colección de métodos que consumirán desde la capa application. En los repositories, se va a instanciar las entidades del 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 223
Capas en una aplicación MVC



Nota. Tomado de *Arquitectura DDD y sus capas*, por Nfanjul, 2014, nfanjul.blogspot.pe (<http://nfanjul.blogspot.pe/2014/09/arquitectura-ddd-y-sus-capas.html>)

LABORATORIO 6.1.: Actualizando datos en una Arquitectura de Capas DDD

Implemente un proyecto ASP.NET MVC, donde permita INSERTAR, CONSULTAR y ACTUALIZAR los datos de la tabla tb_clientes, la cual se encuentra almacenado en la base de datos Negocios2022.

1. Trabajando con SQL Server

Active la base de datos Negocios2022, y creamos los procedimientos almacenados de listado para tb_clientes y tb_paises.

Figura 224
Manejador del SQL Server

```

Use Negocios2022
go
create or alter proc usp_paises
As
Select * from tb_paises
go

create or alter proc usp_clientes
As
Select idcliente,nombrecia,direccion,nombrepais,telefono
from tb_clientes c join tb_paises p on p.Idpais=c.idpais
go
  
```

En la barra de status se indica: Query executed successfully.

Nota. Elaboración propia.

Creamos el procedimiento almacenado `usp_buscar_cliente` el cual retorna el registro del cliente filtrando por su campo `idcliente`.

Figura 225

Manejador del SQL Server

```

SQLQuery1.sql - L...NADIG9\damas (74)* + X
Use Negocios2022
go

create or alter proc usp_buscar_cliente
@idcliente varchar(5)
As
Select Top 1 *
from tb_clientes
Where IdCliente=@idcliente
go

```

160 % ← →

Query executed successfully. LAPTOP-OBNADIG9\SQL EXPRESS ... LAPTOP-OBNADIG9\damas ... Negocios2022 00:00:00 0 rows

Nota. Elaboración propia.

Creamos el procedimiento almacenado `usp_Merge_Cliente`, el cual inserta o actualiza un registro a la tabla `tb_clientes`

Figura 226

Manejador del SQL Server

```

SQLQuery1.sql - L...NADIG9\damas (74)* + X
Use Negocios2022
go

create or alter proc usp_Merge_Cliente
@cod varchar(5),
@nombre varchar(60),
@dir varchar(80),
@idpais char(3),
@fono varchar(24)
As
Merge tb_clientes as target
Using(Select @cod,@nombre,@dir,@idpais,@fono) as source(cod,nom,dir,idpais,fono)
on target.idcliente = source.cod
When MATCHED Then
    Update Set target.nombrecia=source.nom, target.direccion=source.dir,
    target.idpais=source.idpais, target.Telefono=source.fono
When NOT MATCHED Then
    Insert Values(source.cod,source.nom,source.dir,source.idpais,source.fono);
go

```

150 % ← →

Query executed successfully. LAPTOP-OBNADIG9\SQL EXPRESS ... LAPTOP-OBNADIG9\damas ... Negocios2022 00:00:00 0 rows

Nota. Elaboración propia.

Creando un Proyecto ASP.NET MVC

Inicio del Proyecto

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

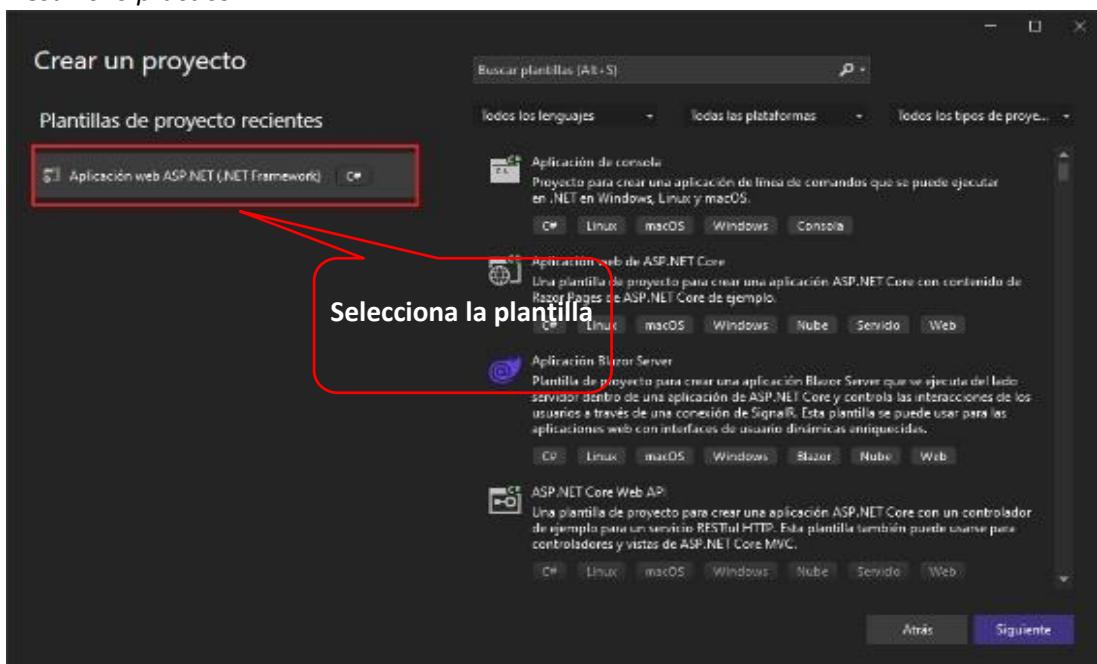
Figura 227
Desarrollo práctico



Nota. Elaboración propia.

Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción siguiente.

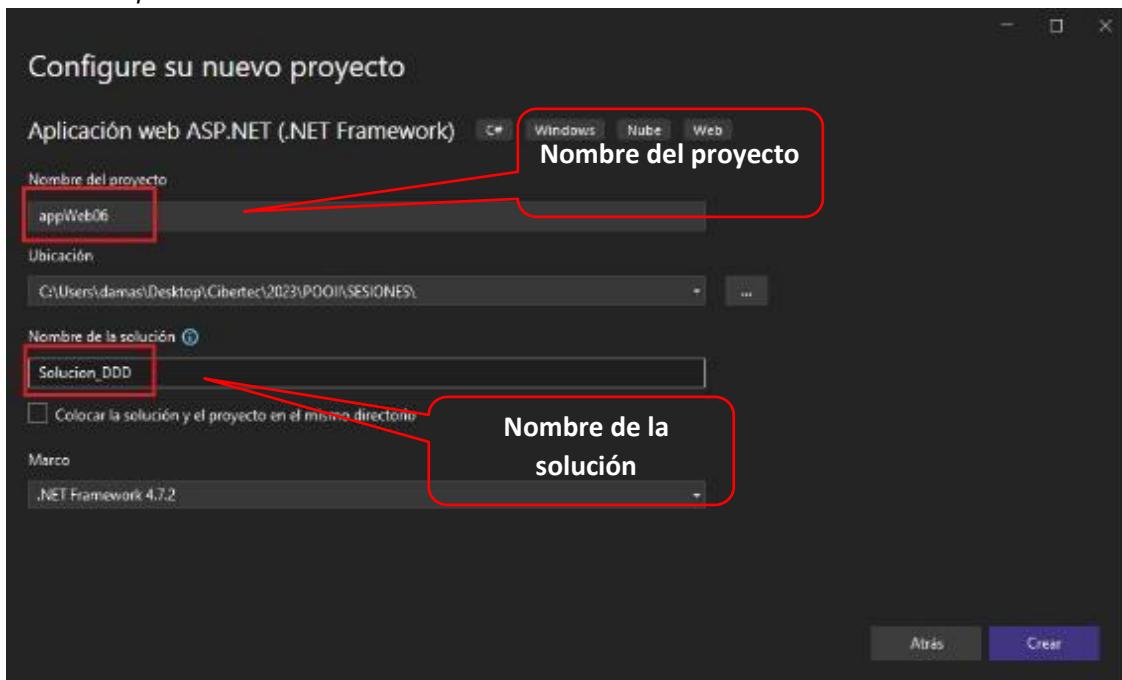
Figura 228
Desarrollo práctico



Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciona la ubicación del mismo, y el nombre de la solución, al terminar presiona la opción **Crear**.

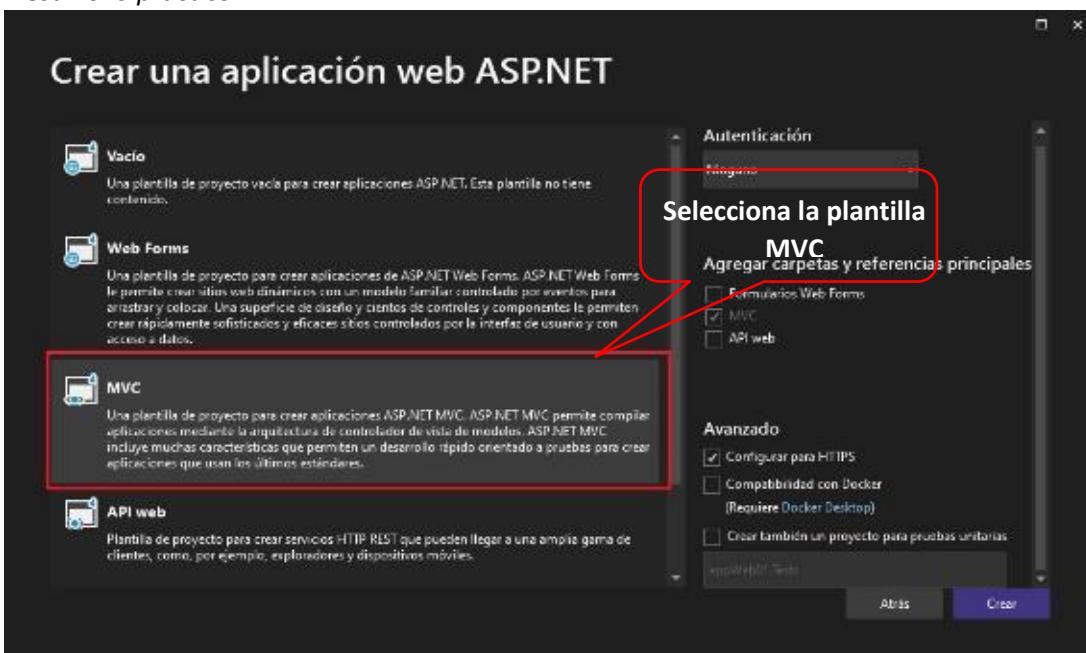
Figura 229
Desarrollo práctico



Nota. Elaboración propia.

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

Figura 230
Desarrollo práctico



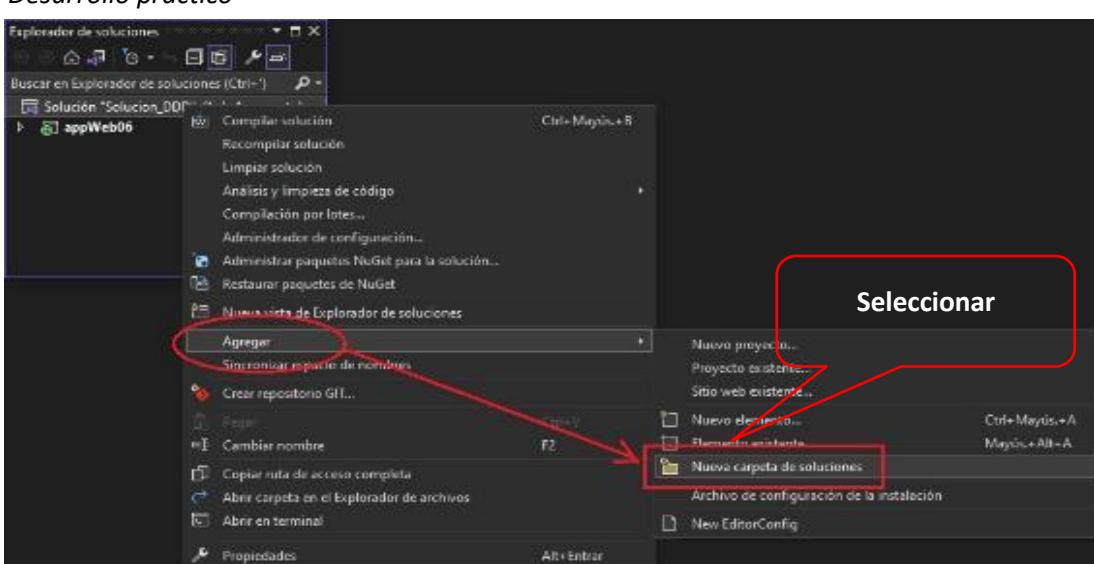
Nota. Elaboración propia.

Creando la estructura de la aplicación de Capas

Una vez creado el proyecto base se procede a armar la estructura de la aplicación, la cual será básicamente carpetas en donde irán los proyectos:

Clic derecho al archivo de solución → Agregue → Nueva carpeta de soluciones (se hará esta operación varias veces).

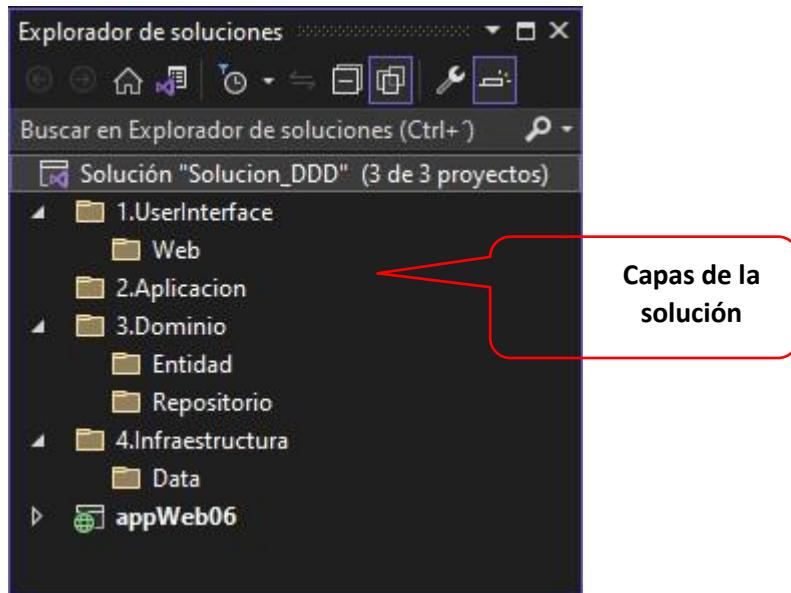
Figura 231
Desarrollo práctico



Nota. Elaboración propia.

Se crearán las siguientes carpetas con la siguiente jerarquía, tal como se muestra en la figura:

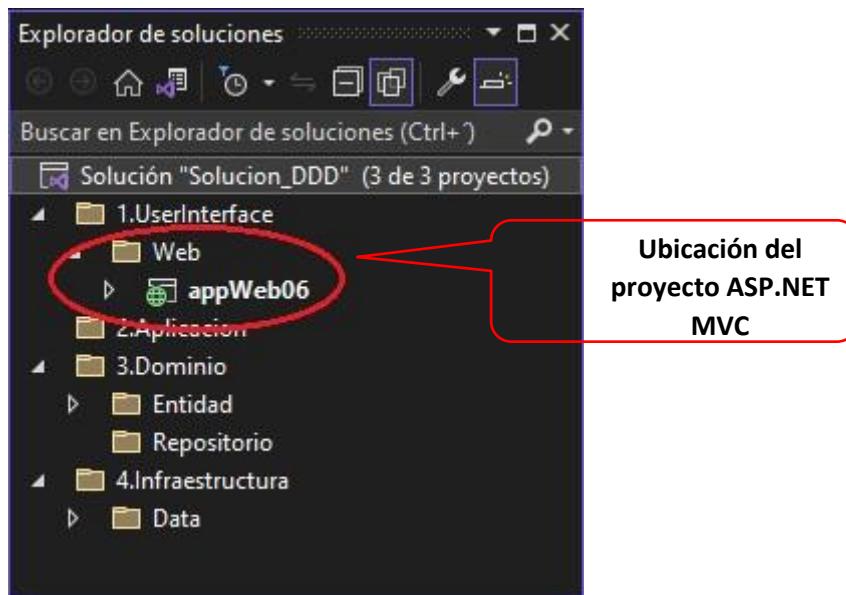
Figura 232
Desarrollo práctico



Nota. Elaboración propia.

A continuación, mueva el proyecto ASP.NET MVC appWeb06 a la carpeta UserInterface subcarpeta Web, tal como se muestra en la figura.

Figura 233
Desarrollo práctico

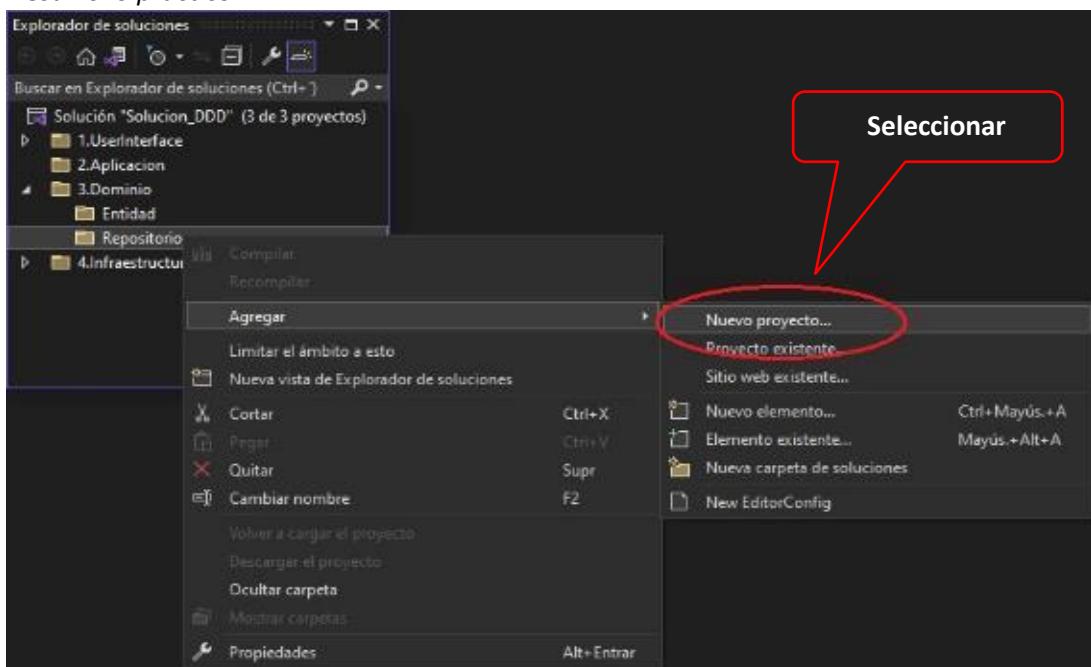


Nota. Elaboración propia.

Creando el proyecto Dominio.Repositorio

En la carpeta Repositorio, agregar un Nuevo Proyecto: Clic derecho a la carpeta → Agregar → Nuevo Proyecto, tal como se muestra.

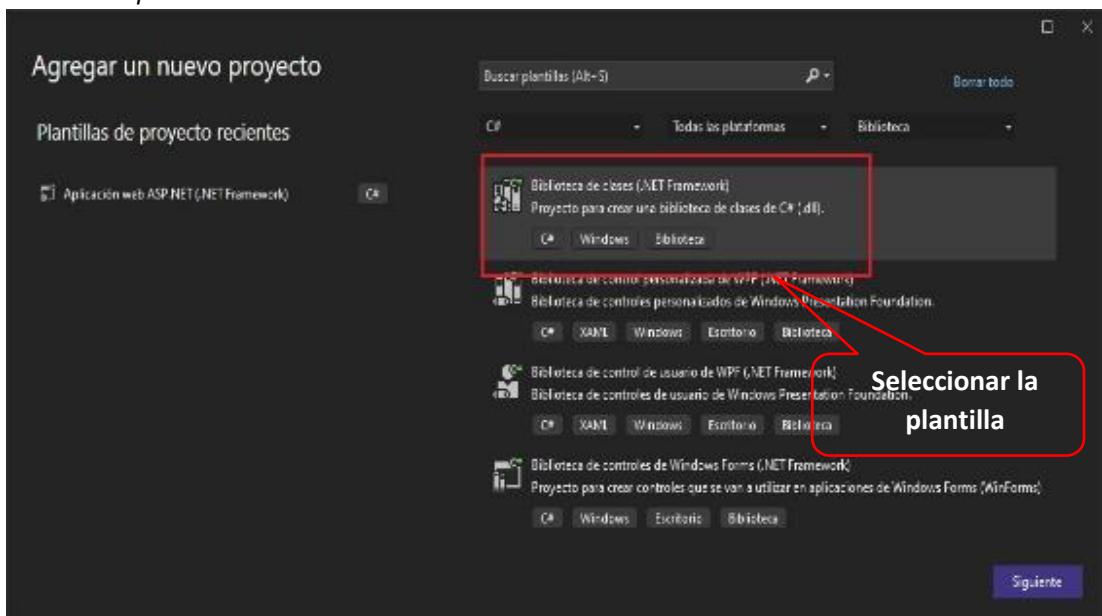
Figura 234
Desarrollo práctico



Nota. Elaboración propia.

Seleccionar la plantilla de Proyecto Biblioteca de Clases (.NET Framework), presionar el botón Siguiente.

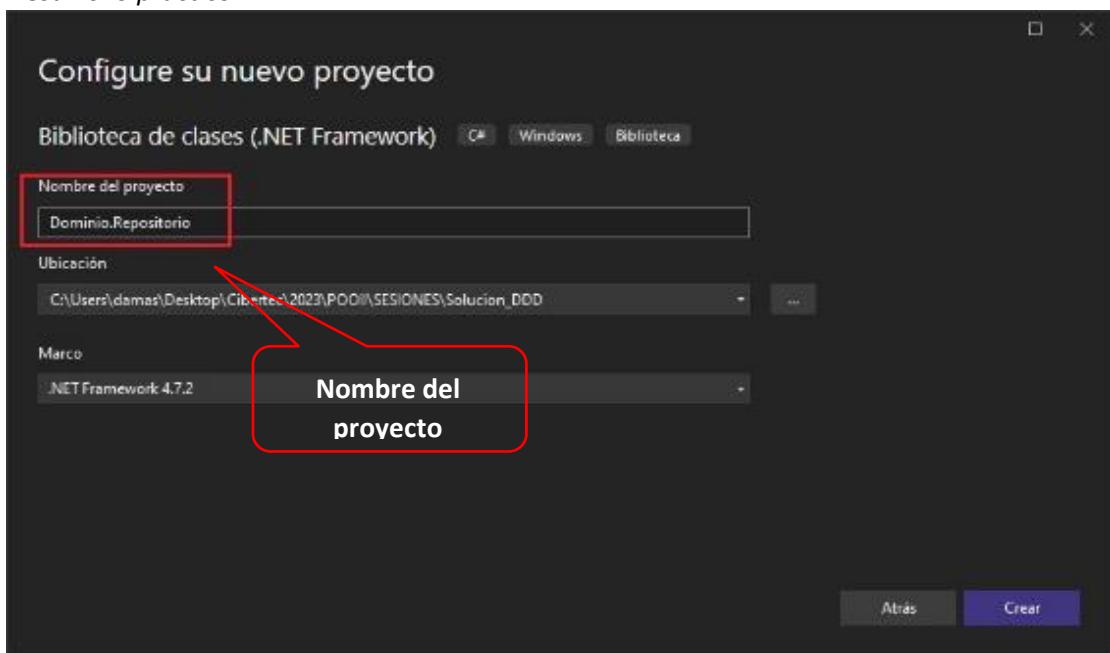
Figura 235
Desarrollo práctico



Nota. Elaboración propia.

En la ventana Configure su nuevo proyecto, asigne el nombre del proyecto: Dominio.Repositorio, presione el botón Crear.

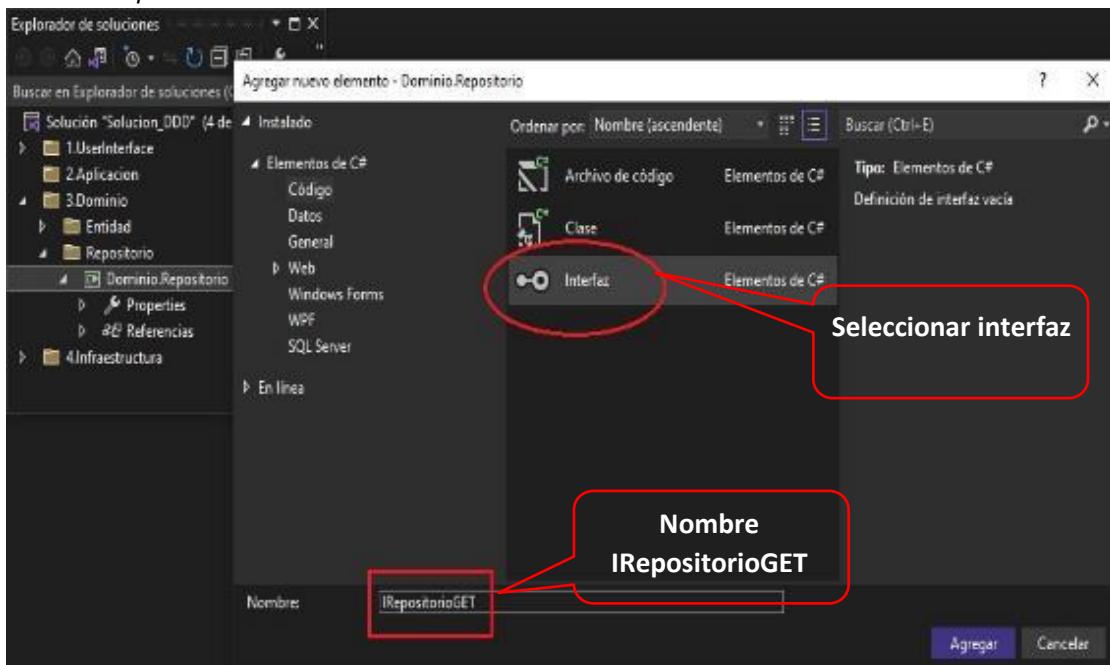
Figura 236
Desarrollo práctico



Nota. Elaboración propia.

En el proyecto creado agregar nuevo elemento de tipo Interfaz llamado **IRepositoryGET**.

Figura 237
Desarrollo práctico



Nota. Elaboración propia.

Defina la interfaz **IRepositoryGET** de tipo **T**, que especifica que debe ser una clase. La cláusula **where** se usa para especificar el tipo de estructura que debe tener **T**. En esta interfaz defina los métodos de tipo **IEnumerable** donde retornan los datos de un determinado tipo.

Figura 238*Desarrollo práctico*

```

1  using ...
2
3  namespace Dominio.Repositorio
4  {
5      public interface IRepositorioGET<T> where T:class
6      {
7          0 referencias
8          IEnumerable<T> GetAll();
9          0 referencias
10         IEnumerable<T> GetByName(string nombre);
11     }
12 }
13
14

```

Interfaz donde retorna listas numeradas

Nota. Elaboración propia.

Defina la interfaz IRepositorioCRUD de tipo **T**, que especifica que debe ser una clase. La cláusula **where** se usa para especificar el tipo de estructura que debe tener **T**. En esta interfaz defina los métodos para ejecutar las operaciones de inserción, actualización, eliminación y búsqueda.

Figura 239*Desarrollo práctico*

```

1  using ...
2
3  namespace Dominio.Repositorio
4  {
5      0 referencias
6      public interface IRepositorioCRUD<T> where T : class
7      {
8          0 referencias
9          string Add(T registro);
10         0 referencias
11         string Update(T registro);
12         0 referencias
13         bool Delete(T registro);
14         0 referencias
15         T Find(string id);
16     }
17 }
18

```

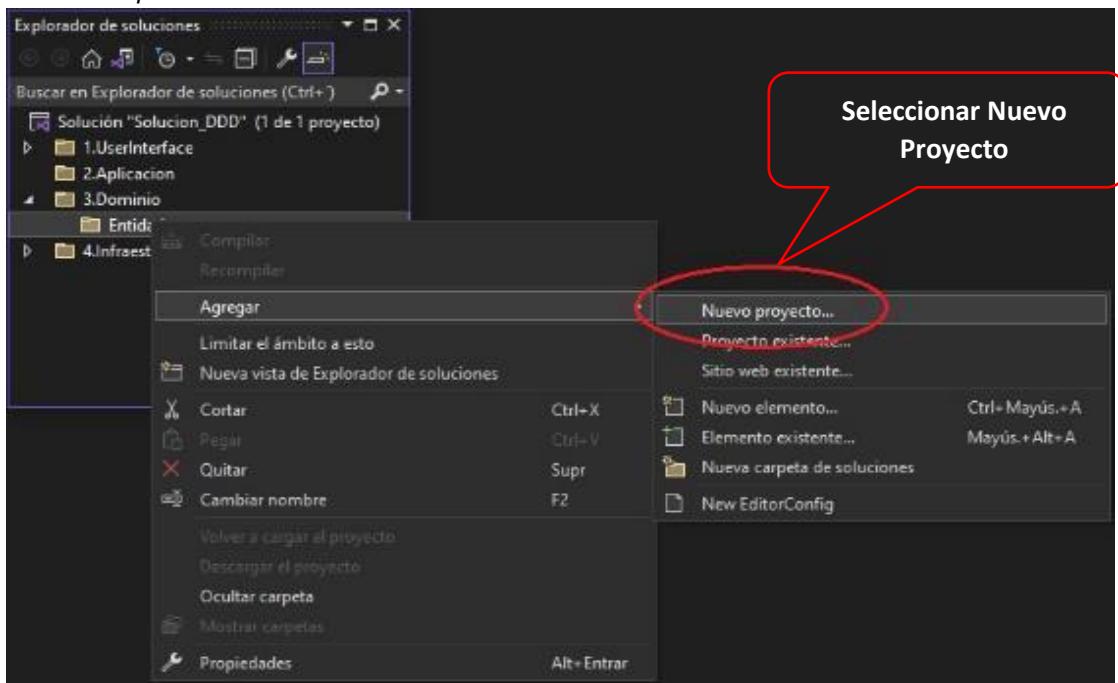
Interfaz donde define los métodos CRUD

Nota. Elaboración propia.

Creando el proyecto Entidades

A continuación, en la carpeta **Entidad** agregar un nuevo Proyecto:
Clic derecho a Entidad → Agregar → Nuevo Proyecto

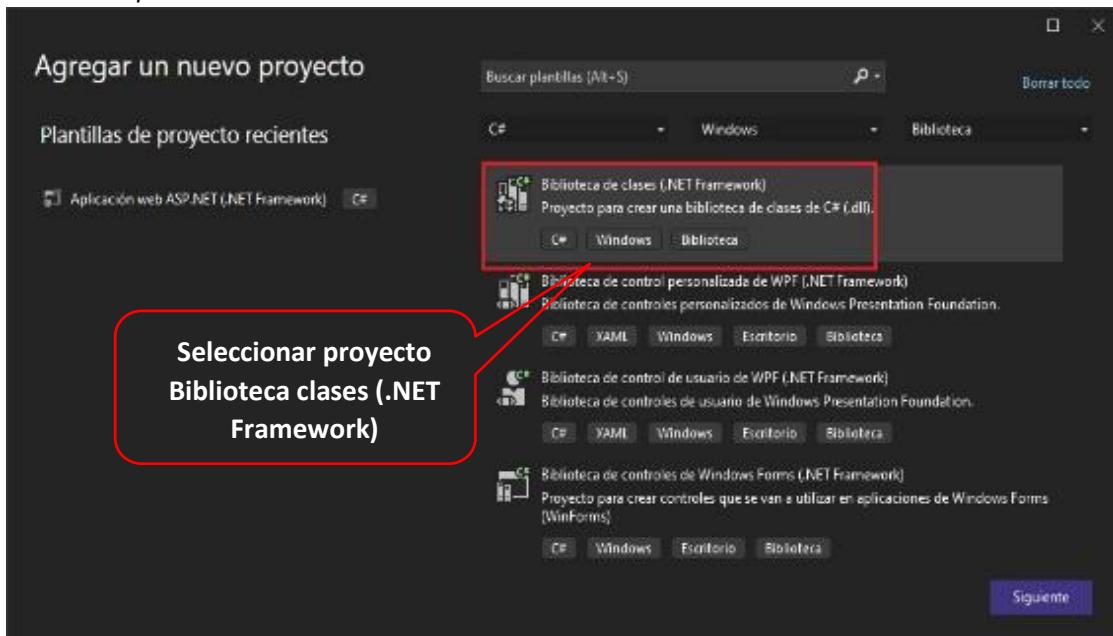
Figura 240
Desarrollo práctico



Nota. Elaboración propia.

Seleccionar el proyecto Biblioteca de clases (.NET Framework).

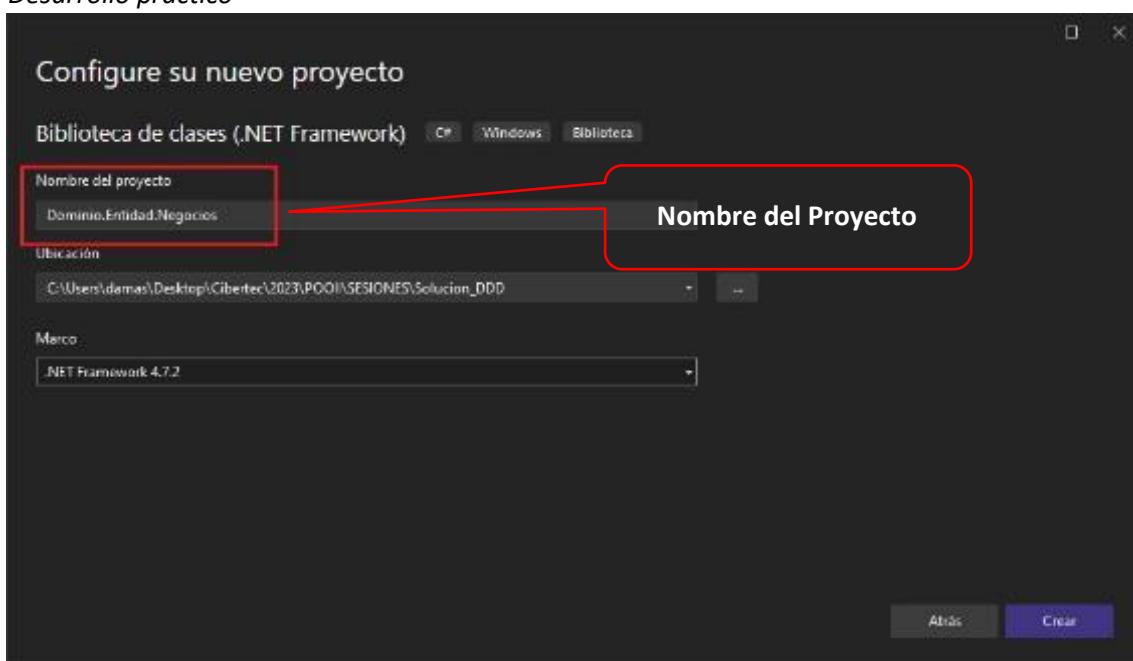
Figura 241
Desarrollo práctico



Nota. Elaboración propia.

En la ventana **Configura su nuevo proyecto**, asigne el nombre del proyecto: Dominio.Entidad.Negocio, tal como se muestra. Presiona el botón **Crear**.

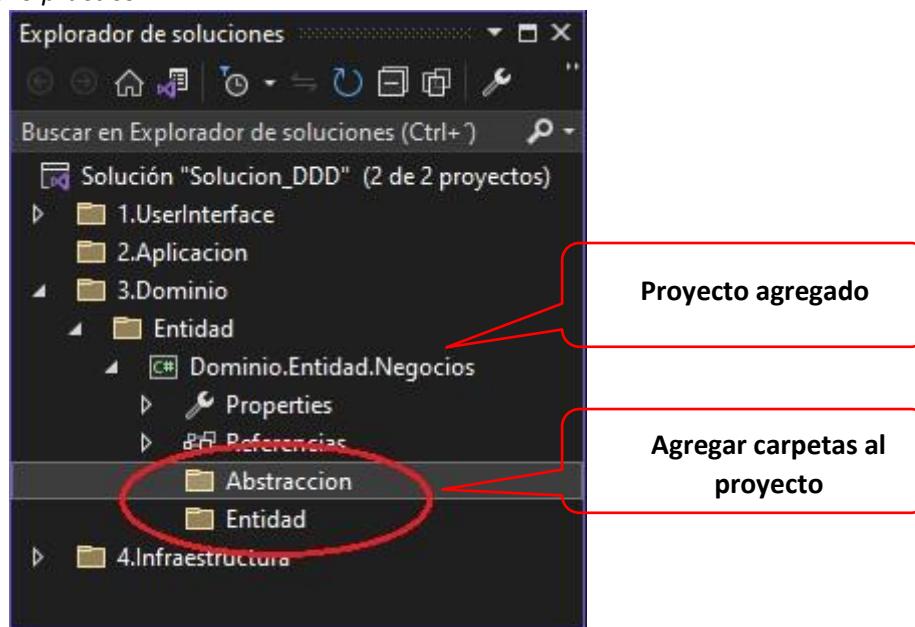
Figura 242
Desarrollo práctico



Nota. Elaboración propia.

De esta manera se agrega el proyecto a la carpeta Entidad, tal como se muestra. Agregar, dentro del proyecto dos carpetas: Abstracciones y Entidades.

Figura 243
Desarrollo práctico

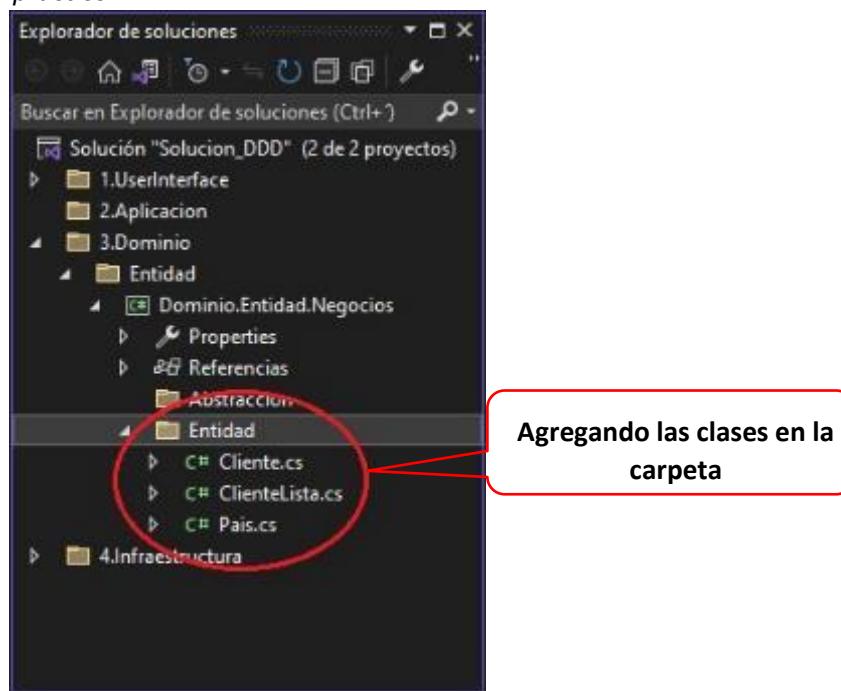


Nota. Elaboración propia.

Definiendo las Entidades del proyecto

En la carpeta Entidad agregar la clase Cliente.cs, ClienteLista.cs y País.cs, tal como se muestra.

Figura 244
Desarrollo práctico



Nota. Elaboración propia.

Abrir el archivo Pais.cs; a continuación, defina los atributos de la clase, tal como se muestra en la figura.

Figura 245
Desarrollo práctico

```

1  using ...
2
3  namespace Dominio.Entidad.Negocios
4  {
5      public class País
6      {
7          public string idpais { get; set; }
8          public string nombrepais { get; set; }
9      }
10 }
11
12
13
14
15

```

The screenshot shows the code editor with the file 'Dominio.Entidad.Negocios.Pais.cs' open. The code defines a class 'Pais' with two properties: 'idpais' and 'nombrepais', both with get and set methods. A red callout bubble points to the class definition with the text 'Atributos de la clase'.

Nota. Elaboración propia.

Abrir el archivo Cliente.cs, a continuación, agregar la referencia a la librería DataAnnotations y defina los atributos de la clase, tal como se muestra en la figura:

Figura 246*Desarrollo práctico*

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.ComponentModel.DataAnnotations;
7  namespace Dominio.Entidad.Negocios.Entidad
8  {
9      public class Cliente
10     {
11         [Display(Name = "Id Cliente"), Required] public string idcliente { get; set; }
12         [Display(Name = "Nombre Cliente"), Required] public string nombre { get; set; }
13         [Display(Name = "Direccion Cliente"), Required] public string direccion { get; set; }
14         [Display(Name = "Id Pais"), Required] public string idpais { get; set; }
15         [Display(Name = "Telefono"), Required] public string fono { get; set; }
16     }
17 }
18

```

Nota. Elaboración propia.

Abrir el archivo ClienteLista.cs, a continuación, agregar la referencia a la librería DataAnnotations y defina los atributos de la clase, tal como se muestra en la figura.

Figura 247*Desarrollo práctico*

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.ComponentModel.DataAnnotations;
7  namespace Dominio.Entidad.Negocios.Entidad
8  {
9      public class ClienteLista
10     {
11         [Display(Name = "Id Cliente"), Required] public string idcliente { get; set; }
12         [Display(Name = "Nombre Cliente"), Required] public string nombre { get; set; }
13         [Display(Name = "Direccion Cliente"), Required] public string direccion { get; set; }
14         [Display(Name = "Pais"), Required] public string nombrepais { get; set; }
15         [Display(Name = "Telefono"), Required] public string fono { get; set; }
16     }
17 }
18

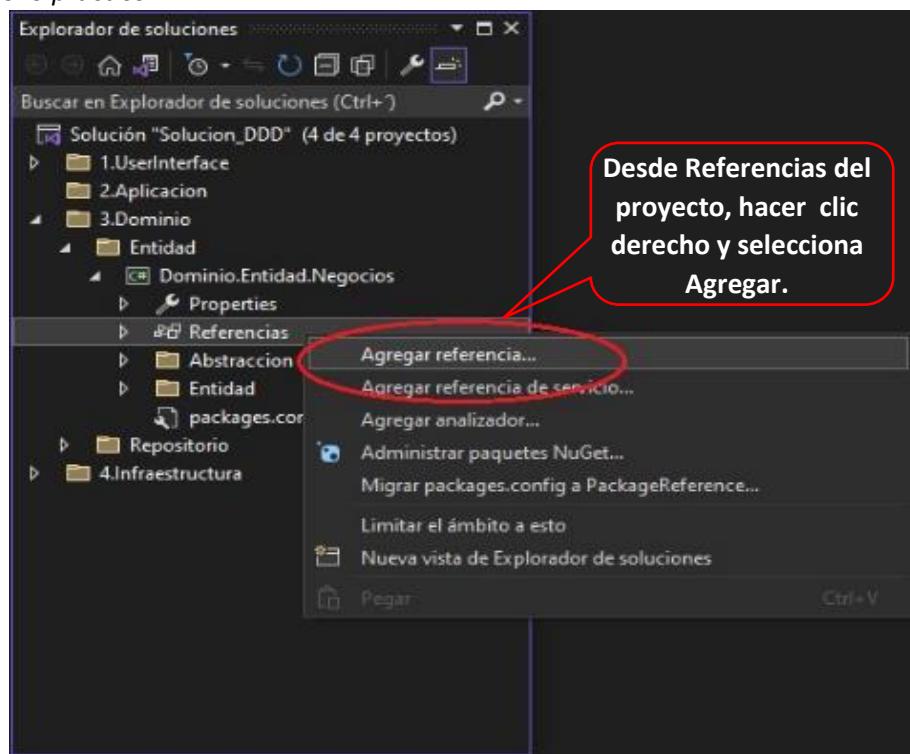
```

Nota. Elaboración propia.

Agregando Referencia al proyecto Dominio.Repository

Para utilizar las interfaces definidas en Dominio.Repository, en el proyecto Dominio.Entidad.Negocios, agregamos una referencia, tal como se muestra.

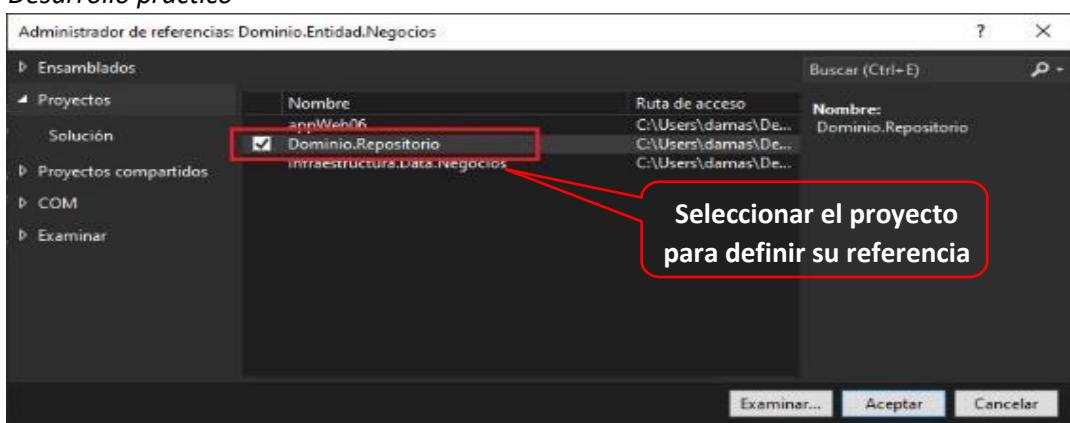
Figura 248
Desarrollo práctico



Nota. Elaboración propia.

Desde el Administrador de referencias, seleccionar la opción **Proyectos**, donde listará todos los proyectos de la solución, **marcar el proyecto Dominio.Repositorio**, tal como se muestra.

Figura 249
Desarrollo práctico

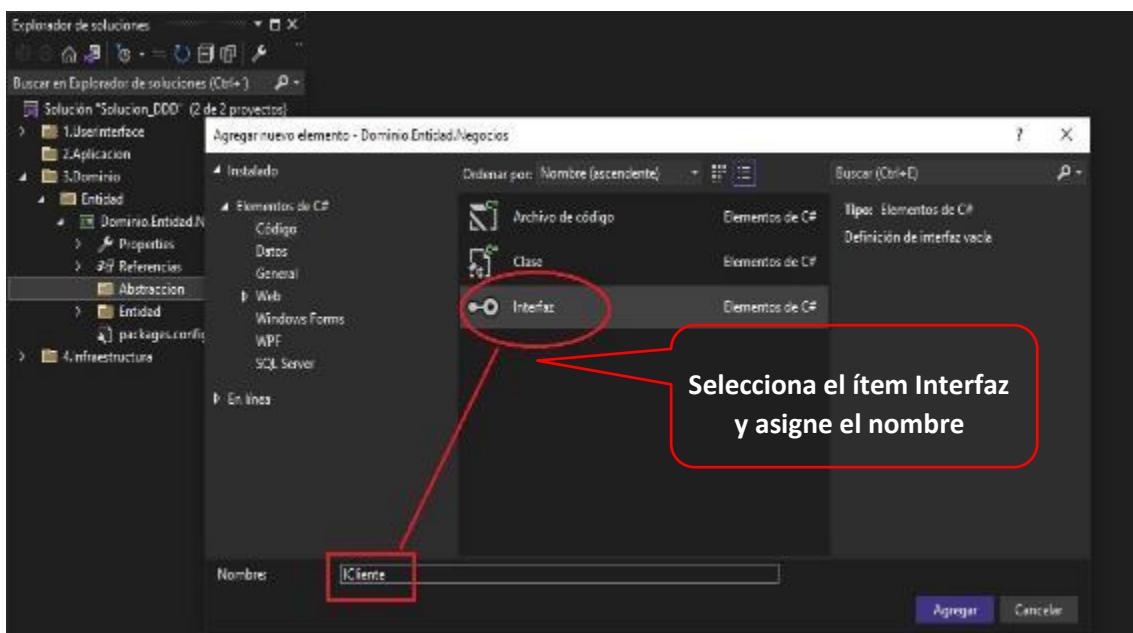


Nota. Elaboración propia.

Definiendo las Interfaces del Proyecto

Como parte del Dominio, defina las Interfaces para cada una de las Entidades. En la carpeta Abstracción, clic derecho Agregar Nuevo Elemento, seleccionar el elemento Interfaz y asigne su nombre ICliente, lo mismo para IPais.

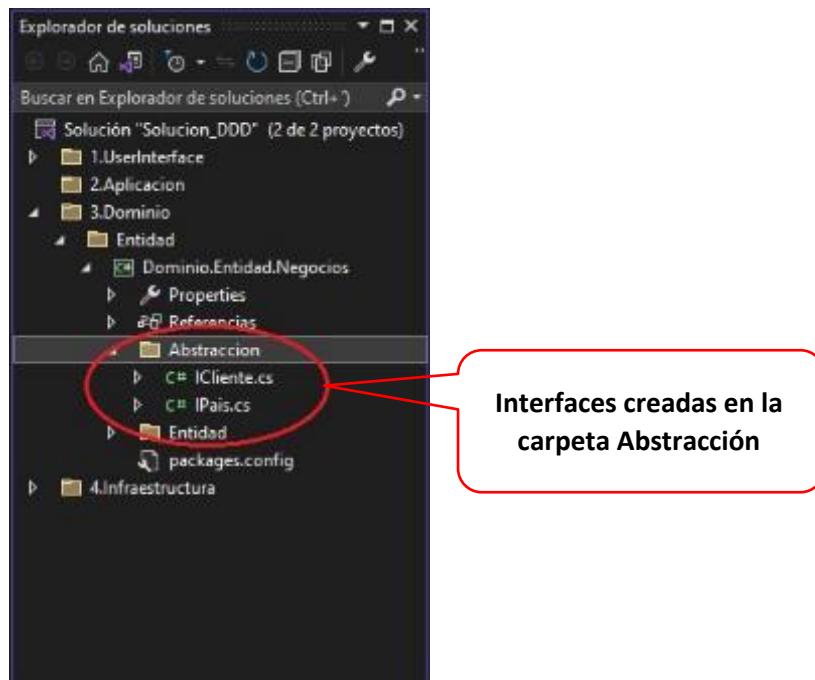
Figura 250
Desarrollo práctico



Nota. Elaboración propia.

Interfaces creadas en la carpeta Abstracciones. A continuación, defina los métodos a cada una de las Interfaces.

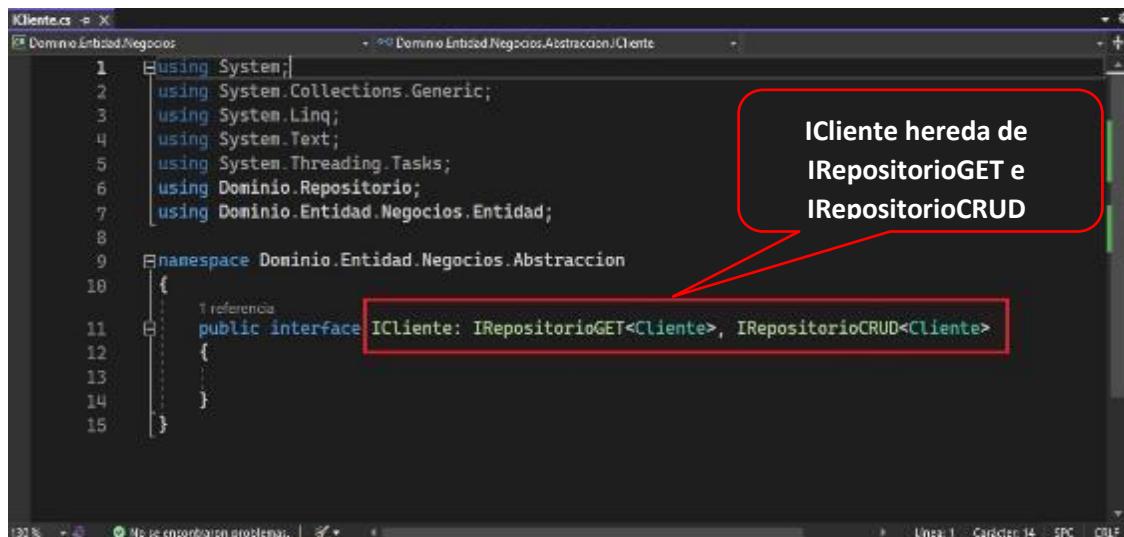
Figura 251
Desarrollo práctico



Nota. Elaboración propia.

En la **interfaz** **ICliente**, importar la carpeta Entidades y el proyecto Dominio.Repositorio; a continuación, definir a **ICliente** como interfaz que hereda los métodos de **IRepositoryGET** e **IRepositoryCRUD** de tipo **Cliente**, tal como se muestra.

Figura 252
Desarrollo práctico



```

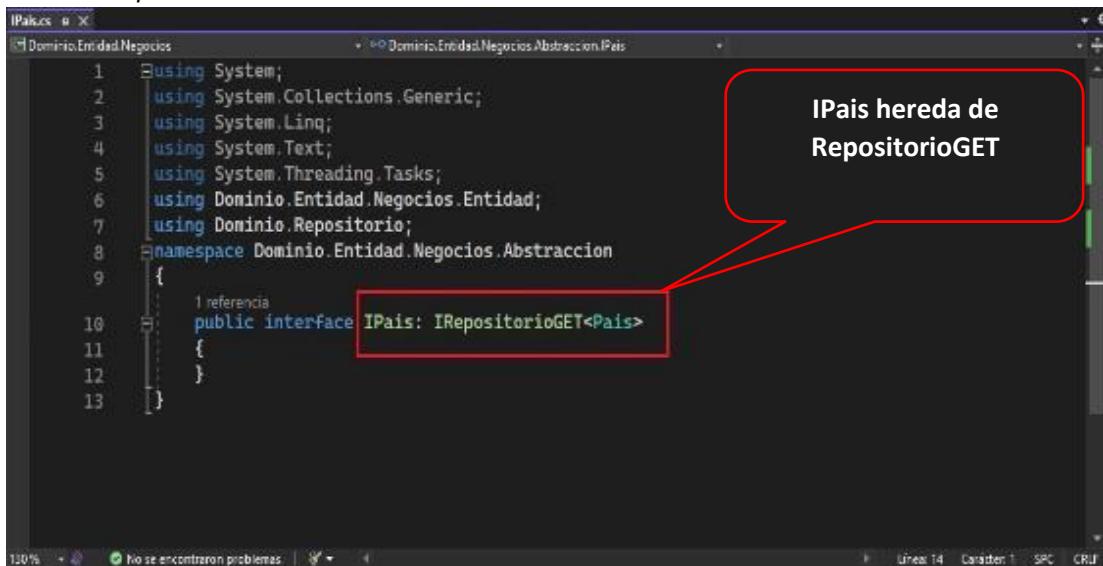
1  Using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using Dominio.Repositorio;
7  using Dominio.Entidad.Negocios.Entidad;
8
9  namespace Dominio.Entidad.Negocios.Abstraccion
10 {
11     public interface ICliente: IRepositoryGET<Cliente>, IRepositoryCRUD<Cliente>
12     {
13     }
14 }
15

```

Nota. Elaboración propia.

En la **interfaz** **IPais**, importar la carpeta Entidades y el proyecto Dominio.Repositorio; a continuación, definir a **IPais** como interfaz que hereda los métodos de **IRepositoryGET** de tipo **País**, tal como se muestra.

Figura 253
Desarrollo práctico



```

1  Using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using Dominio.Entidad.Negocios.Entidad;
7  using Dominio.Repositorio;
8
9  namespace Dominio.Entidad.Negocios.Abstraccion
10 {
11     public interface IPais: IRepositoryGET<País>
12     {
13     }
14 }
15

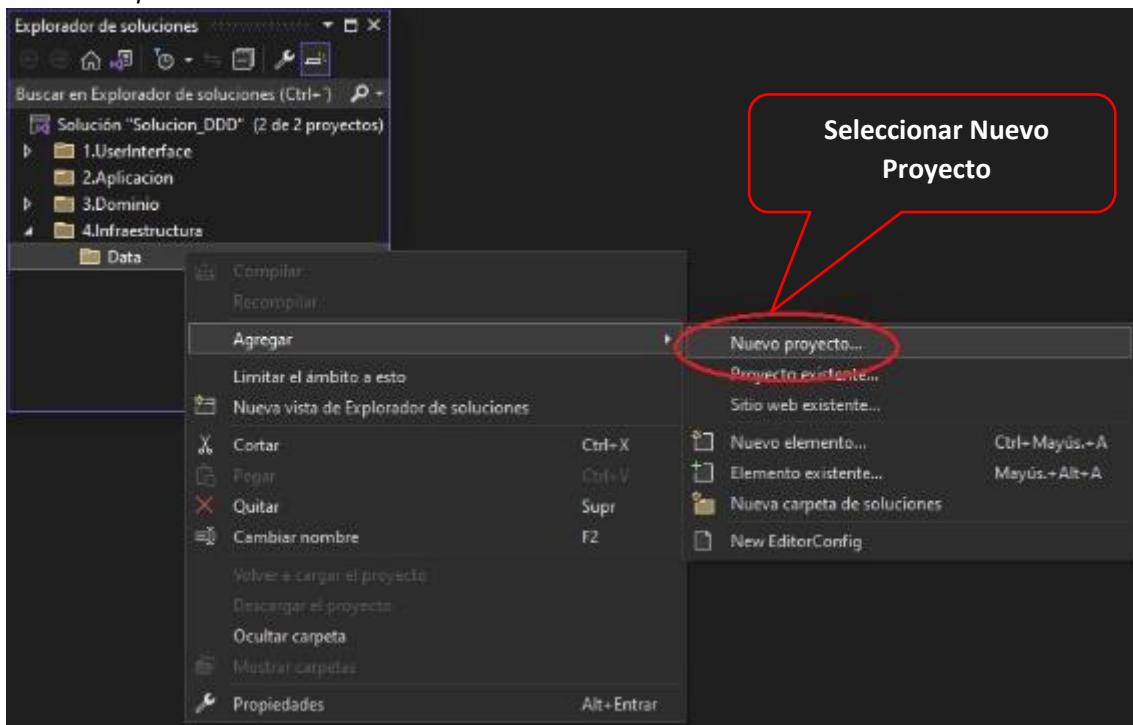
```

Nota. Elaboración propia.

Creando el proyecto Data.Negocios

A continuación, agregue en la carpeta Data dentro de la carpeta Infraestructura:
Clic derecho a Data → Agregar → Nuevo Proyecto

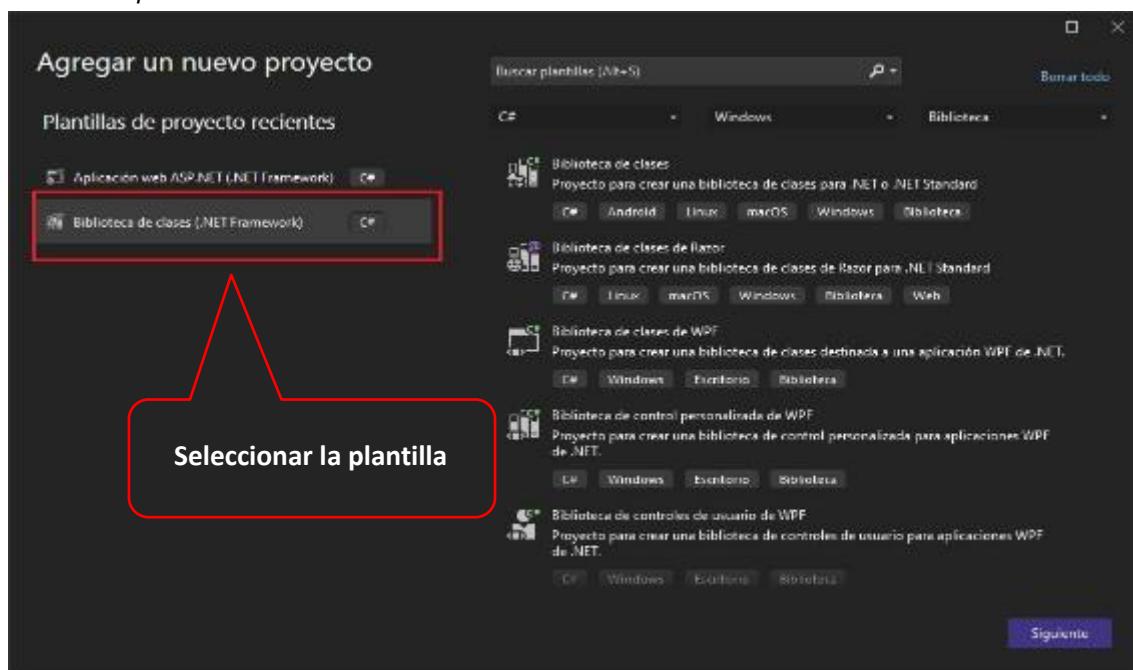
Figura 254
Desarrollo práctico



Nota. Elaboración propia.

En la ventana Agregar Nuevo Proyecto, seleccionar la plantilla Biblioteca de clases (.NET Framework).

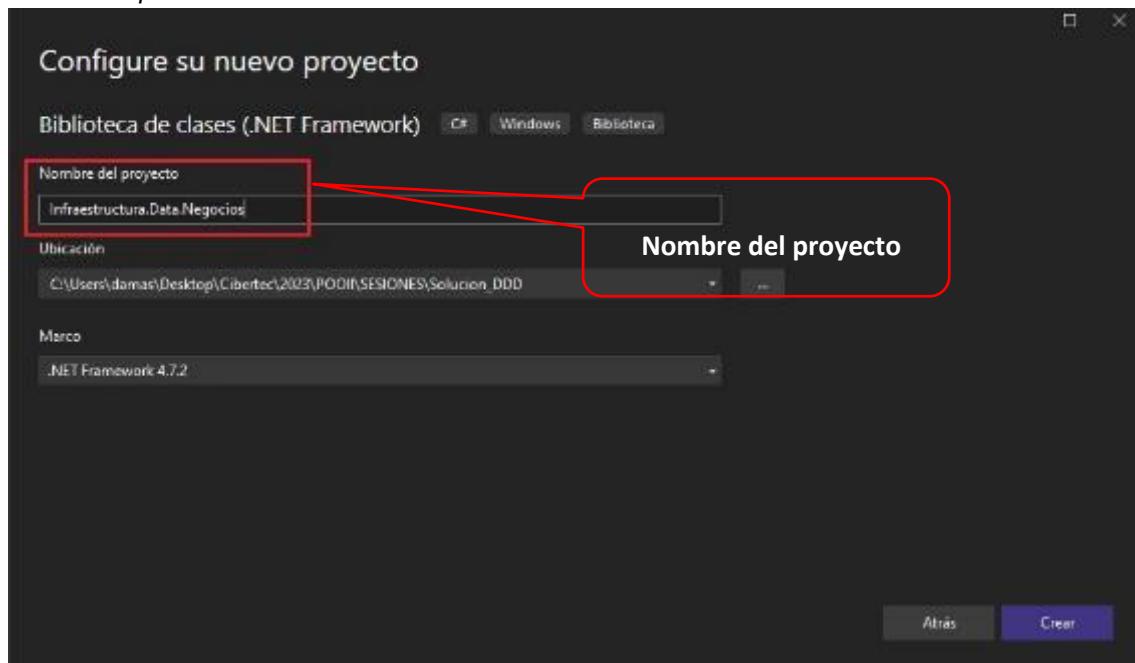
Figura 255
Desarrollo práctico



Nota. Elaboración propia.

Asigne el nombre al proyecto: Infraestructura.Data.Negocios. Presiona el botón CREAR.

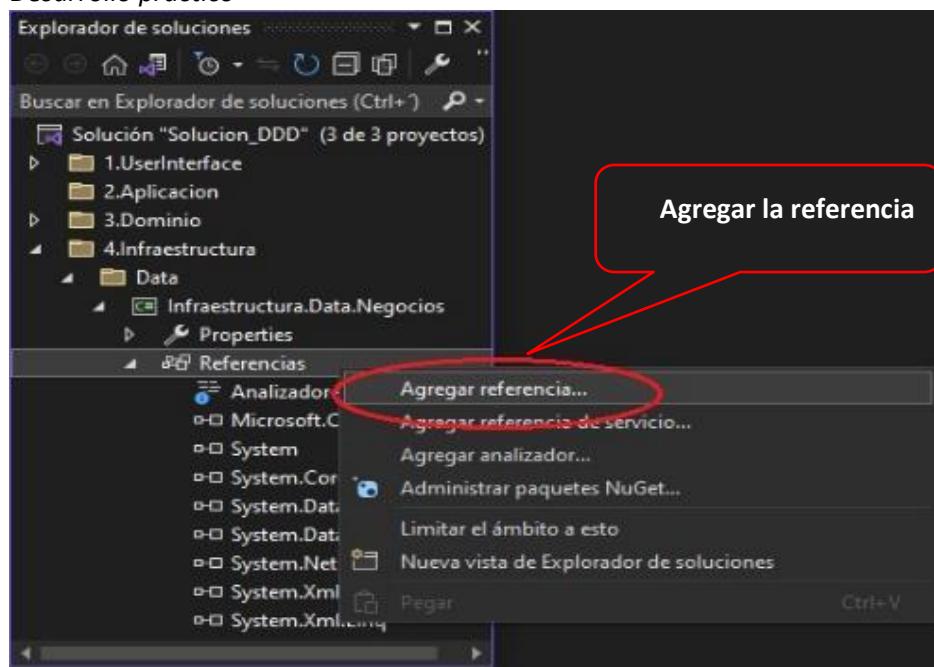
Figura 256
Desarrollo práctico



Nota. Elaboración propia.

Creado el proyecto, procedemos a agregar una referencia al proyecto.
Dominio.Entidad.Negocios

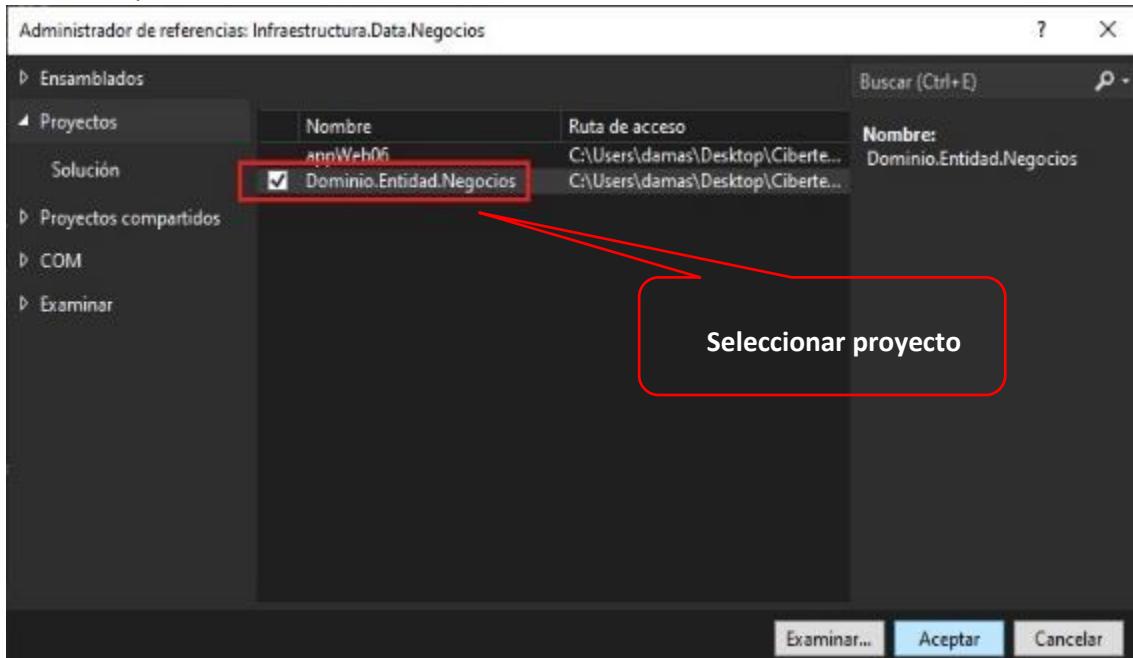
Figura 257
Desarrollo práctico



Nota. Elaboración propia.

Para utilizar el Modelo del Dominio, referenciamos, desde el proyecto Infraestructura.DAO.Negocios, hacia el proyecto Dominio.Entidad.Negocio, tal como se muestra, presionar el botón ACEPTAR.

Figura 258
Desarrollo práctico

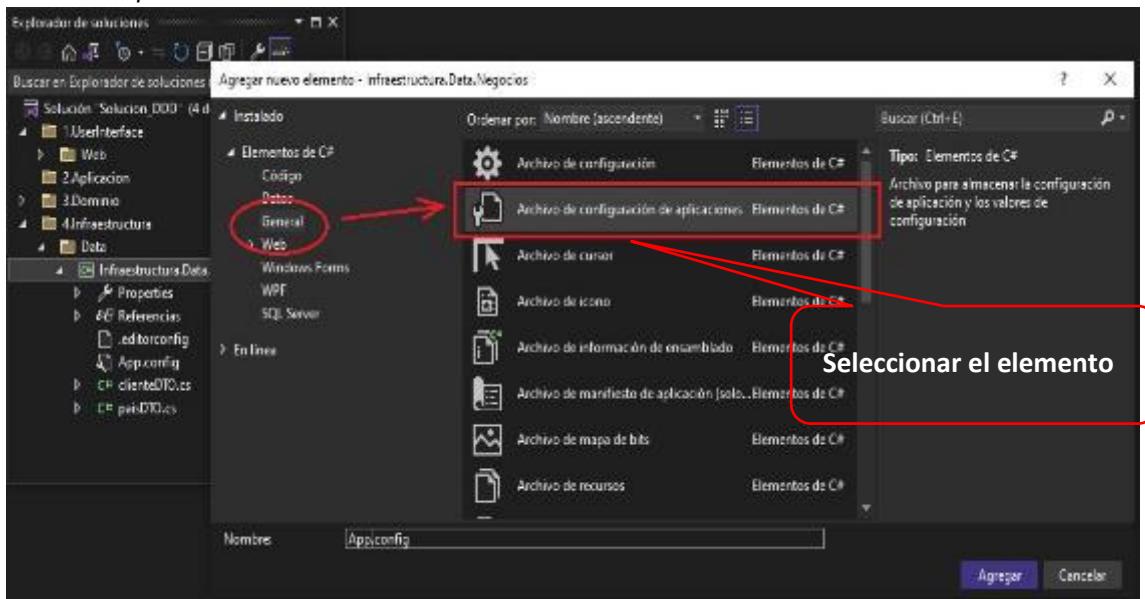


Nota. Elaboración propia.

Agregando App.config

En el proyecto vamos a agregar un archivo de configuración de aplicaciones, la cual se llamará App.config.

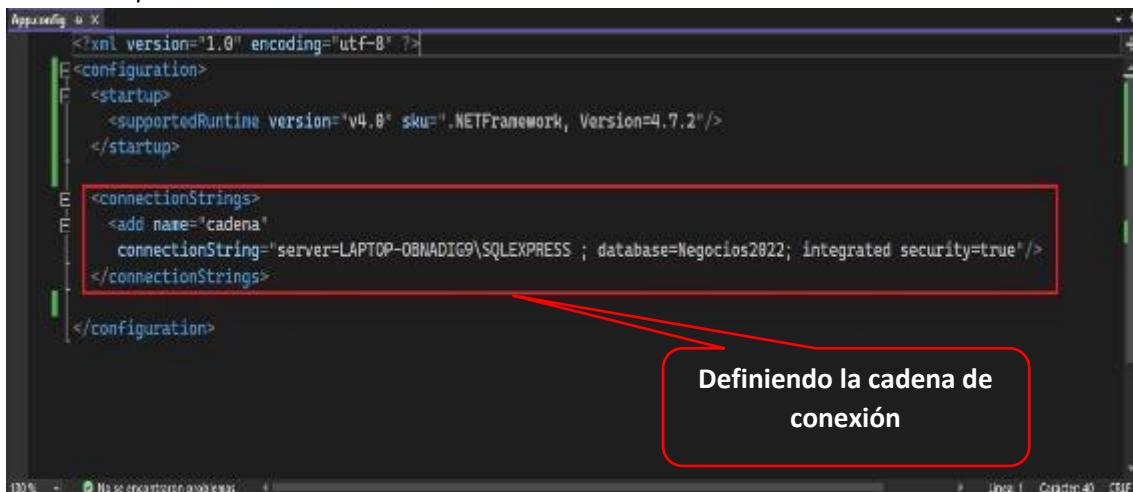
Figura 259
Desarrollo práctico



Nota. Elaboración propia.

En el App.config, defina la cadena de conexión, tal como se muestra.

Figura 260
Desarrollo práctico



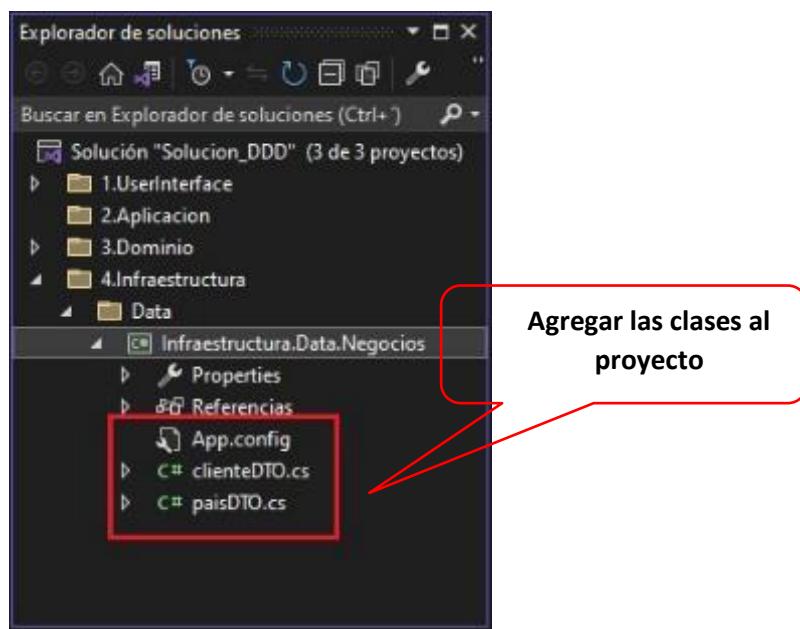
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework, Version=4.7.2"/>
  </startup>
  <connectionStrings>
    <add name="cadena"
      connectionString="server=LAPTOP-DBNADIG9\SQLEXPRESS ; database=Negocios2022; integrated security=true" />
  </connectionStrings>
</configuration>
```

Nota. Elaboración propia.

Definiendo las Clases del proyecto

Seguidamente, agregar las clases: clienteDTO.cs y paisDTO.cs, tal como se muestra.

Figura 261
Desarrollo práctico



Nota. Elaboración propia.

Programando la clase paisDTO

Referencia las librerías y el proyecto de entidades y abstracciones. Definimos esta clase como una clase heredada de conexionDAO e IPais (para implementar su método getPaises()).

Figura 262*Desarrollo práctico*

```

1 ① Using System;
2 ② using System.Collections.Generic;
3 ③ using System.Linq;
4 ④ using System.Text;
5 ⑤ using System.Threading.Tasks;
6 ⑥ using System.Data;
7 ⑦ using System.Data.SqlClient;
8 ⑧ using System.Configuration;
9 ⑨ using Dominio.Entidad.Negocios.Entidad;
10 ⑩ using Dominio.Entidad.Negocios.Abstraccion;
11 ⑪ namespace Infraestructura.Data.Negocios
12 {
13     ⑫     public class paisDTO : IPais
14     {
15         ⑬         public IEnumerable<Pais> GetAll()
16         {
17             ⑭             List<Pais> temporal = new List<Pais>();
18             ⑮             using (SqlConnection cn = new SqlConnection(
19                 ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
20             {
21                 ⑯                 cn.Open();
22                 ⑰                 SqlCommand cmd = new SqlCommand("exec usp_paises", cn);
23                 ⑱                 SqlDataReader dr = cmd.ExecuteReader();
24                 ⑲                 while (dr.Read())
25                 {
26                     ⑳                     temporal.Add(new Pais()
27                     {
28                         ⑴                         idpais = dr.GetString(0),
29                         ⑵                         nomrepais = dr.GetString(1)
30                     });
31                 }
32             }
33         }
34     }
35 }

```

Nota. Elaboración propia.

Programe el método getAll() donde ejecuta el procedure de tb_países, retornando los registros.

Figura 263*Desarrollo práctico*

```

15     ⑫     public IEnumerable<Pais> GetAll()
16     {
17         ⑬         List<Pais> temporal = new List<Pais>();
18         ⑮         using (SqlConnection cn = new SqlConnection(
19             ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
20         {
21             ⑯             cn.Open();
22             ⑰             SqlCommand cmd = new SqlCommand("exec usp_paises", cn);
23             ⑱             SqlDataReader dr = cmd.ExecuteReader();
24             ⑲             while (dr.Read())
25             {
26                 ⑳                 temporal.Add(new Pais()
27                 {
28                     ⑴                     idpais = dr.GetString(0),
29                     ⑵                     nomrepais = dr.GetString(1)
30                 });
31             }
32         }
33     }
34 }

```

Nota. Elaboración propia.

Programando la clase clienteDTO

Referencia las librerías. Defina esta clase como clase heredada de ICliente, e implemente los métodos de la ICliente. A continuación, vamos a programar cada uno de ellos.

Figura 264
Desarrollo práctico

```

clienteDTO.cs - X
Infraestructura.Data.Negocios
@Infrastructure.Data.Negocios.clienteDTO
AddCliente registro

4 @ using System.Text;
5 using System.Threading.Tasks;
6 using System.Data;
7 using System.Data.SqlClient;
8 using System.Configuration;
9 using Dominio.Entidad.Negocios.Entidad;
10 using Dominio.Entidad.Negocios.Abstraccion;
11
12 namespace Infraestructura.Data.Negocios
13 {
14     public class clienteDTO : IClientes
15     {
16         1 referencia
17         public string Add(Cliente registro) ...
18         1 referencia
19         public bool Delete(Cliente registro) ...
20         1 referencia
21         public Cliente Find(string id) ...
22         1 referencia
23         public IEnumerable<Cliente> GetAll() ...
24         1 referencia
25         public IEnumerable<Cliente> GetByName(string nombre) ...
26         1 referencia
27         public string Update(Cliente registro) ...
28     }
29 }

```

Nota. Elaboración propia.

Programa el método GetAll(), el cual retorna los registros de la tabla tb_clientes ejecutando el procedimiento almacenado usp_clientes.

Figura 265
Desarrollo práctico

```

clienteDTO.cs - X
Infraestructura.Data.Negocios
@Infrastructure.Data.Negocios.clienteDTO
AddCliente registro

28     1 referencia
29     public IEnumerable<Cliente> GetAll()
30     {
31         List<Cliente> temporal = new List<Cliente>();
32         using (SqlConnection cn = new SqlConnection(
33             ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
34         {
35             cn.Open();
36             SqlCommand cmd = new SqlCommand("exec usp_clientes", cn);
37             SqlDataReader dr = cmd.ExecuteReader();
38             while (dr.Read())
39             {
40                 temporal.Add(new Cliente()
41                 {
42                     idcliente = dr.GetString(0),
43                     nombre = dr.GetString(1),
44                     direccion = dr.GetString(2),
45                     idpais = dr.GetString(3),
46                     fono = dr.GetString(4),
47                 });
48             }
49             dr.Close();
50         }
51         return temporal;
52     }

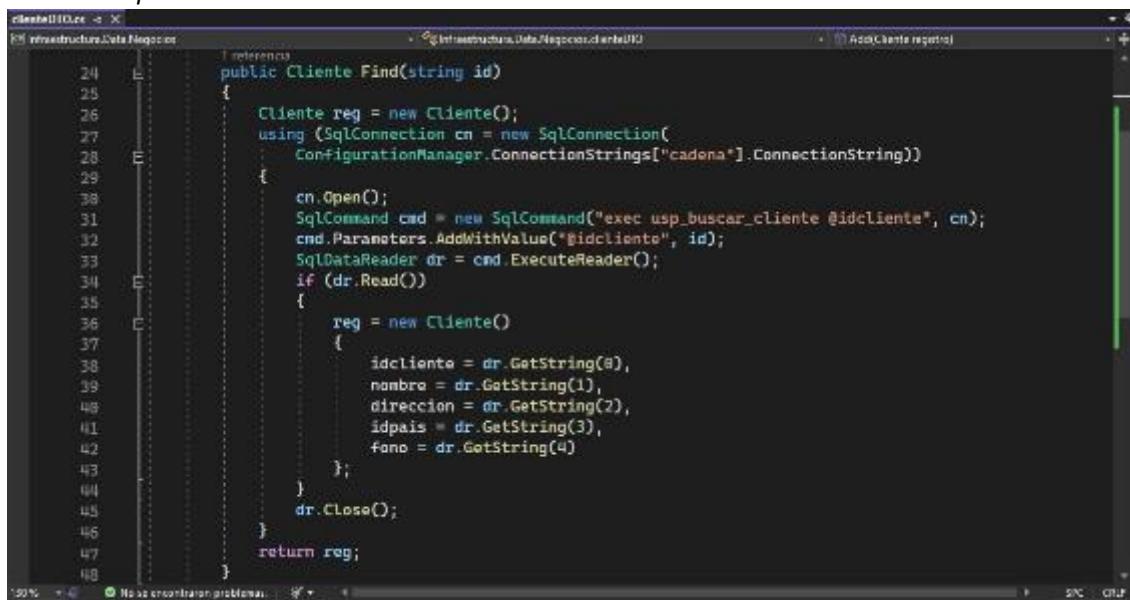
```

Nota. Elaboración propia.

Programa el método Find(string id), el cual retorna el registro de la tabla tb_clientes ejecutando el procedimiento almacenado usp_buscar_cliente.

Figura 266

Desarrollo práctico



```

clienteDTO.cs - X
Infraestructura.Datos.Negocios.ClienteDTO
    public Cliente Find(string id)
    {
        Cliente reg = new Cliente();
        using (SqlConnection cn = new SqlConnection(
            ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
        {
            cn.Open();
            SqlCommand cmd = new SqlCommand("exec usp_buscar_cliente @idcliente", cn);
            cmd.Parameters.AddWithValue("@idcliente", id);
            SqlDataReader dr = cmd.ExecuteReader();
            if (dr.Read())
            {
                reg = new Cliente()
                {
                    idcliente = dr.GetString(0),
                    nombre = dr.GetString(1),
                    direccion = dr.GetString(2),
                    idpais = dr.GetString(3),
                    fono = dr.GetString(4)
                };
            }
            dr.Close();
        }
        return reg;
    }

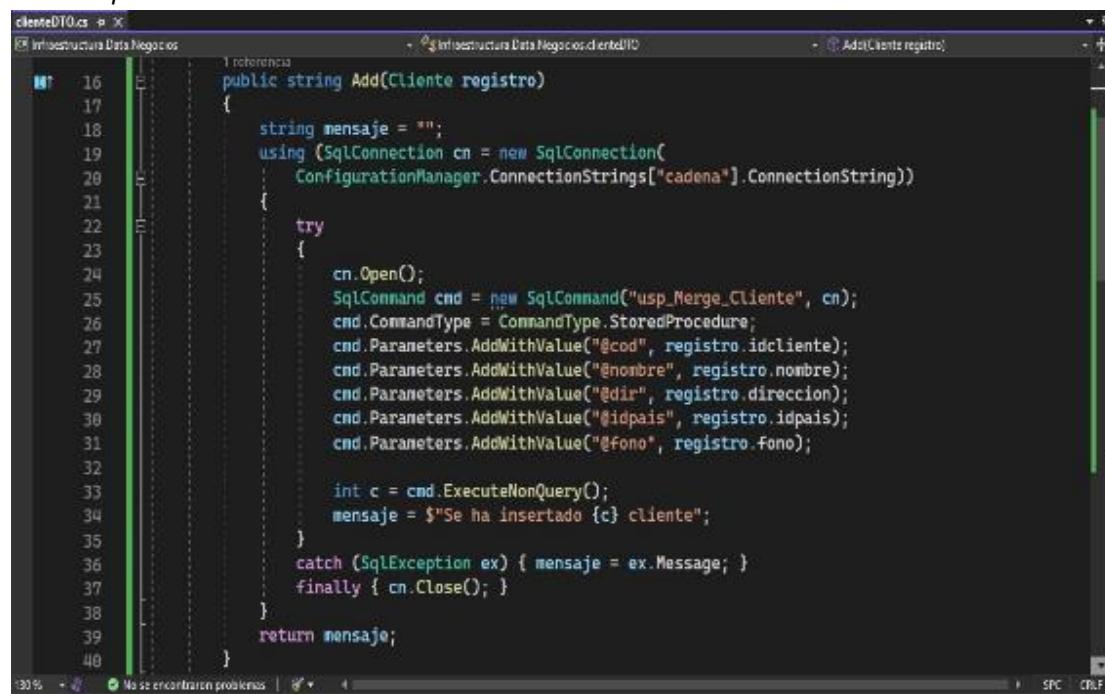
```

Nota. Elaboración propia.

Programa el método Add, el cual ejecuta el procedimiento almacenado usp_Merge_Cliente para agregar un cliente, en dicho proceso retornamos el mensaje de confirmación, en caso de error enviamos el mensaje de error.

Figura 267

Desarrollo práctico



```

clienteDTO.cs - X
Infraestructura.Datos.Negocios.ClienteDTO
    public string Add(Cliente registro)
    {
        string mensaje = "";
        using (SqlConnection cn = new SqlConnection(
            ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
        {
            try
            {
                cn.Open();
                SqlCommand cmd = new SqlCommand("usp_Merge_Cliente", cn);
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.AddWithValue("@cod", registro.idcliente);
                cmd.Parameters.AddWithValue("@nombre", registro.nombre);
                cmd.Parameters.AddWithValue("@dir", registro.direccion);
                cmd.Parameters.AddWithValue("@idpais", registro.idpais);
                cmd.Parameters.AddWithValue("@fono", registro.fono);

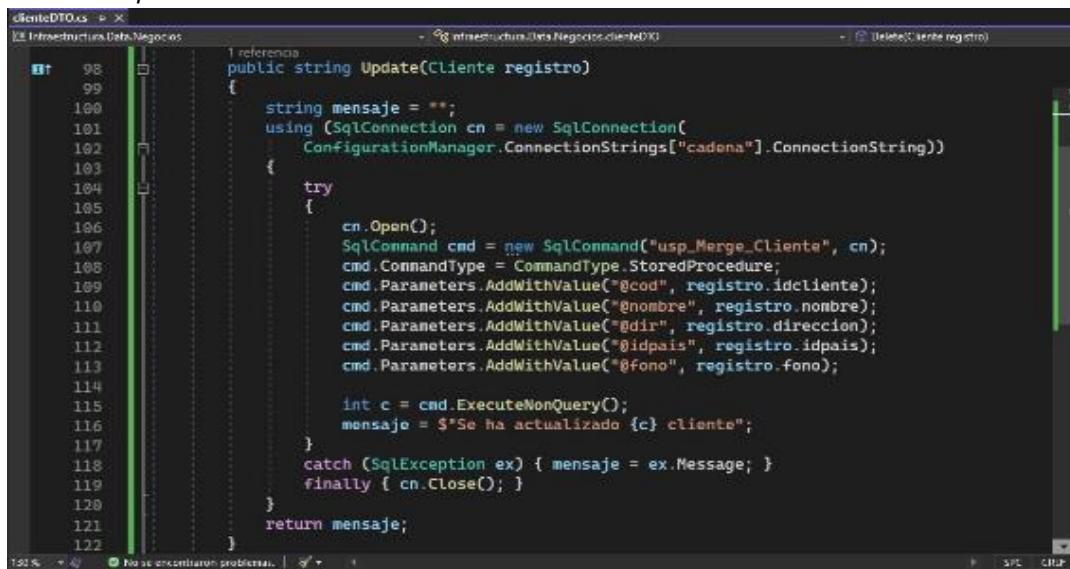
                int c = cmd.ExecuteNonQuery();
                mensaje = $"Se ha insertado {c} cliente";
            }
            catch (SqlException ex) { mensaje = ex.Message; }
            finally { cn.Close(); }
        }
        return mensaje;
    }

```

Nota. Elaboración propia.

Programa el método Update, el cual ejecuta el procedimiento almacenado usp_Merge_Cliente para actualizar los datos de un cliente, en dicho proceso retornamos el mensaje de confirmación, en caso de error enviamos el mensaje de error.

Figura 268
Desarrollo práctico



```

public string Update(Cliente registro)
{
    string mensaje = "";
    using (SqlConnection cn = new SqlConnection(
        ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
    {
        try
        {
            cn.Open();
            SqlCommand cmd = new SqlCommand("usp_Merge_Cliente", cn);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@cod", registro.idcliente);
            cmd.Parameters.AddWithValue("@nombre", registro.nombre);
            cmd.Parameters.AddWithValue("@dir", registro.direccion);
            cmd.Parameters.AddWithValue("@idpais", registro.idpais);
            cmd.Parameters.AddWithValue("@fono", registro.fono);

            int c = cmd.ExecuteNonQuery();
            mensaje = $"Se ha actualizado {c} cliente";
        }
        catch (SqlException ex) { mensaje = ex.Message; }
        finally { cn.Close(); }
    }
    return mensaje;
}

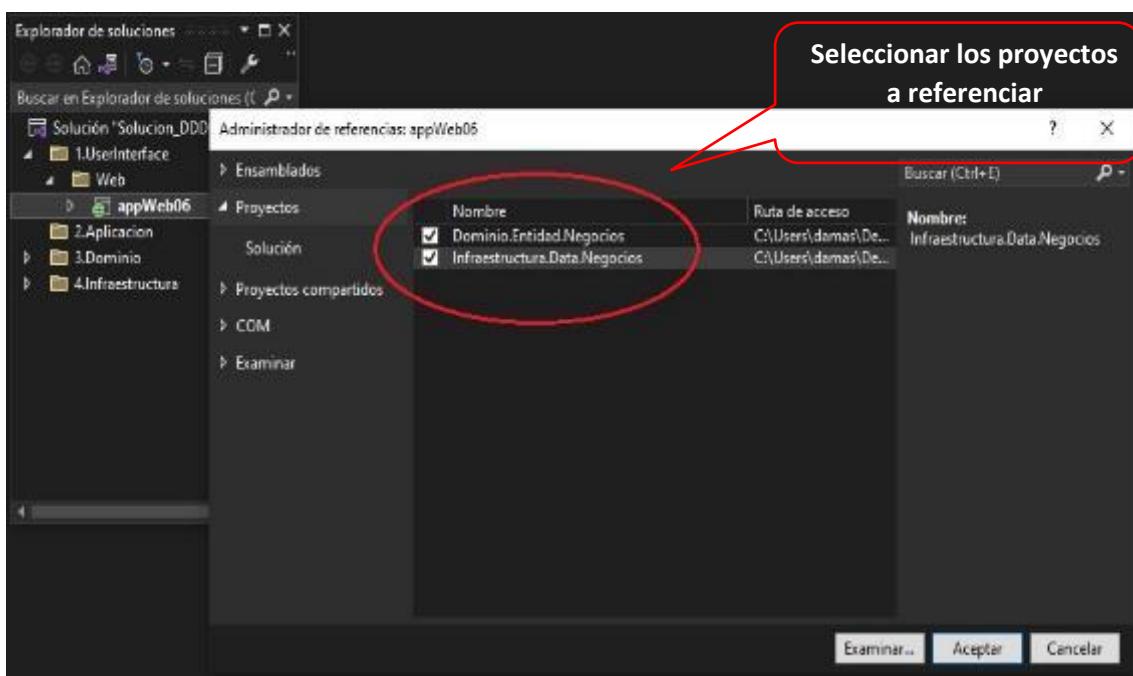
```

Nota. Elaboración propia.

Trabajando con la Aplicación Web ASP.NET MVC

En la aplicación AppWeb06, vamos a referenciar los proyectos dll Dominio.Entidad.Negocios e Infraestructura.Data.Negocios, seleccionando, desde la pestaña Proyectos.

Figura 269
Desarrollo práctico



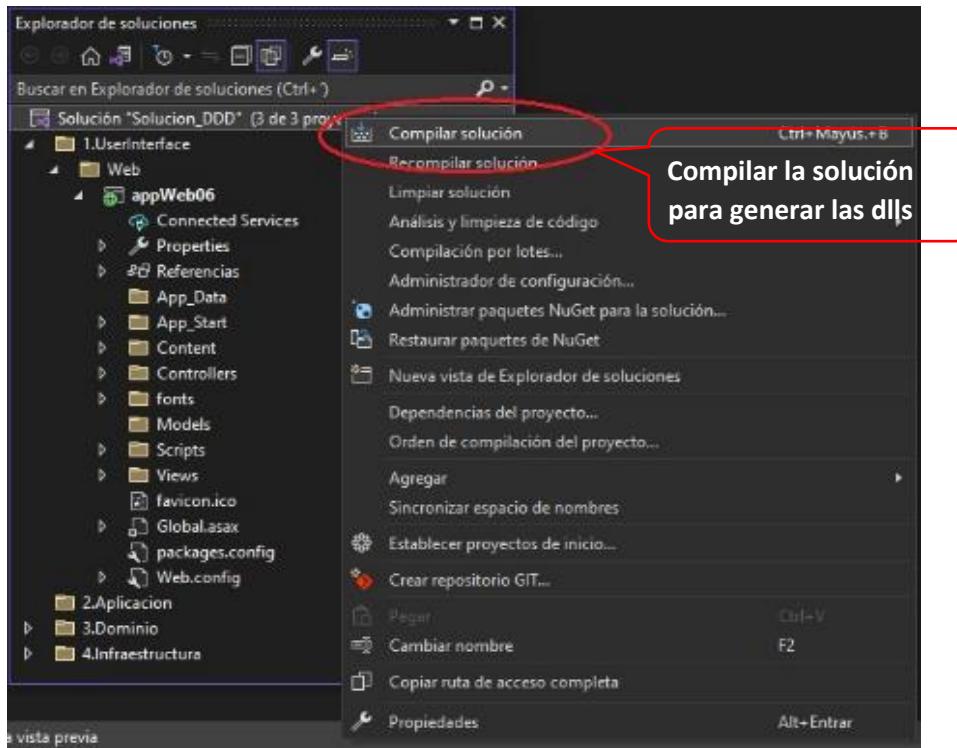
Nota. Elaboración propia.

Creación de un Controlador

Antes de comenzar, Compilar la Solución, para generar las dlls de los proyectos desarrollados.

Figura 270

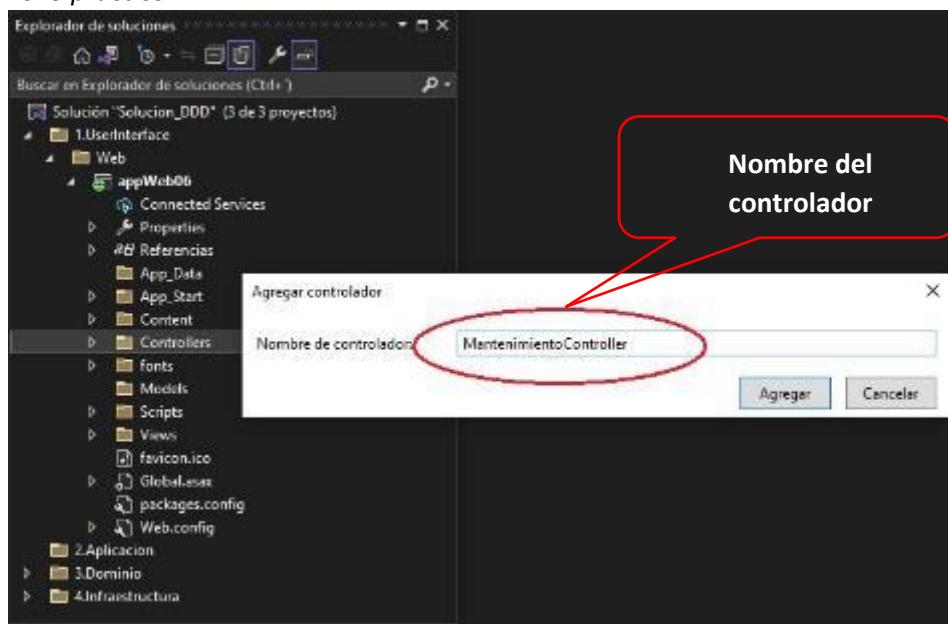
Desarrollo práctico



Nota. Elaboración propia.

A continuación, cree un controlador: Desde la carpeta Controllers, seleccione la opción Agregar, Controlador, MVC 5 en blanco.

Figura 271
Desarrollo práctico



Nota. Elaboración propia.

Programando el Controlador y sus Vistas

Primero, definir las referencias de las bibliotecas de código: Dominio e Infraestructura, tal como se muestra.

Figura 272
Desarrollo práctico

The screenshot shows the Visual Studio code editor with the file 'MantenimientoController.cs' open. The code defines a controller class 'MantenimientoController' that inherits from 'Controller'. It includes several 'using' statements for namespaces related to System, Collections, Linq, Web, and MVC, as well as specific domain and infrastructure entities. A red callout points to the 'using' statements with the text 'Referencia a los proyectos' (Reference to projects). Another red callout points to the controller class definition with the text 'Instanciar los DTO' (Instantiate the DTOs). The code editor shows syntax highlighting and code completion suggestions.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using Dominio.Entidad.Negocios.Entidad;
7  using Infraestructura.Data.Negocios;
8
9  namespace appWeb06.Controllers
10 {
11     public class MantenimientoController : Controller
12     {
13         paisDTO _pais=new paisDTO();
14         clienteDTO _cliente=new clienteDTO();
15     }
16 }

```

Nota. Elaboración propia.

Instanciar las clases DTO: país y cliente. A continuación, defina el ActionResult Index(), retorna los registros de tb_clientes.

Figura 273

Desarrollo práctico

```
MantenimientoController.cs  X: clienteDTO.cs
appWeb05                         appWeb05.Controllers.MantenimientoController
1  @using ...
2
3
4
5
6
7
8
9
10  namespace appWeb05.Controllers
11  {
12      public class MantenimientoController : Controller
13      {
14          paisDTO _pais=new paisDTO();
15          clienteDTO _cliente=new clienteDTO();
16
17          public ActionResult Index()
18          {
19              return View(_cliente.GetAll());
20          }
21      }
22  }

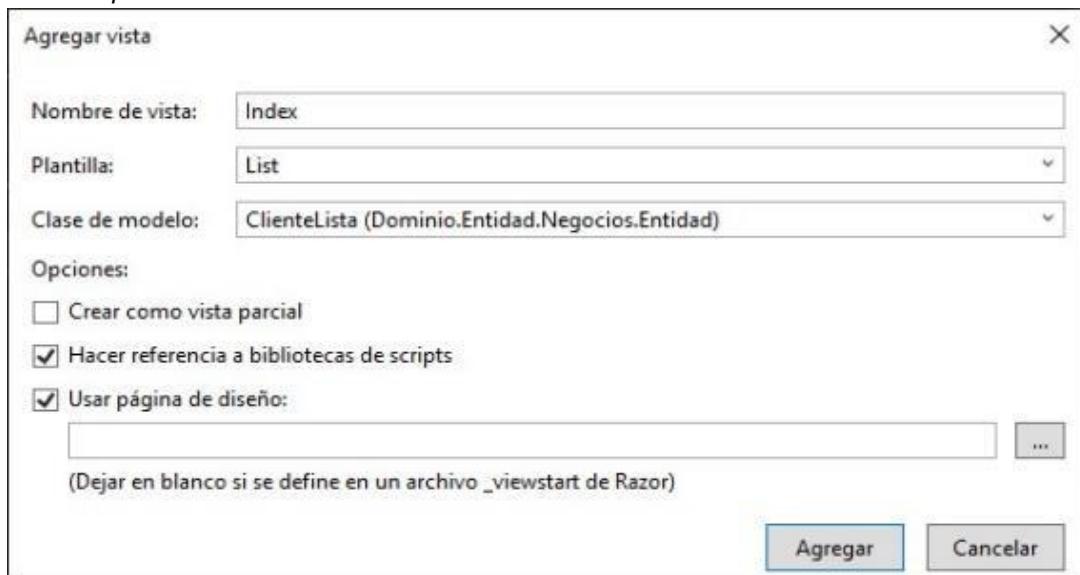
ActionResult donde envía la lista de los registros de tb_clientes
```

Nota. Elaboración propia.

Agregar vista de Index, cuya plantilla es List y clase de modelo: ClienteLista, tal como se muestra.

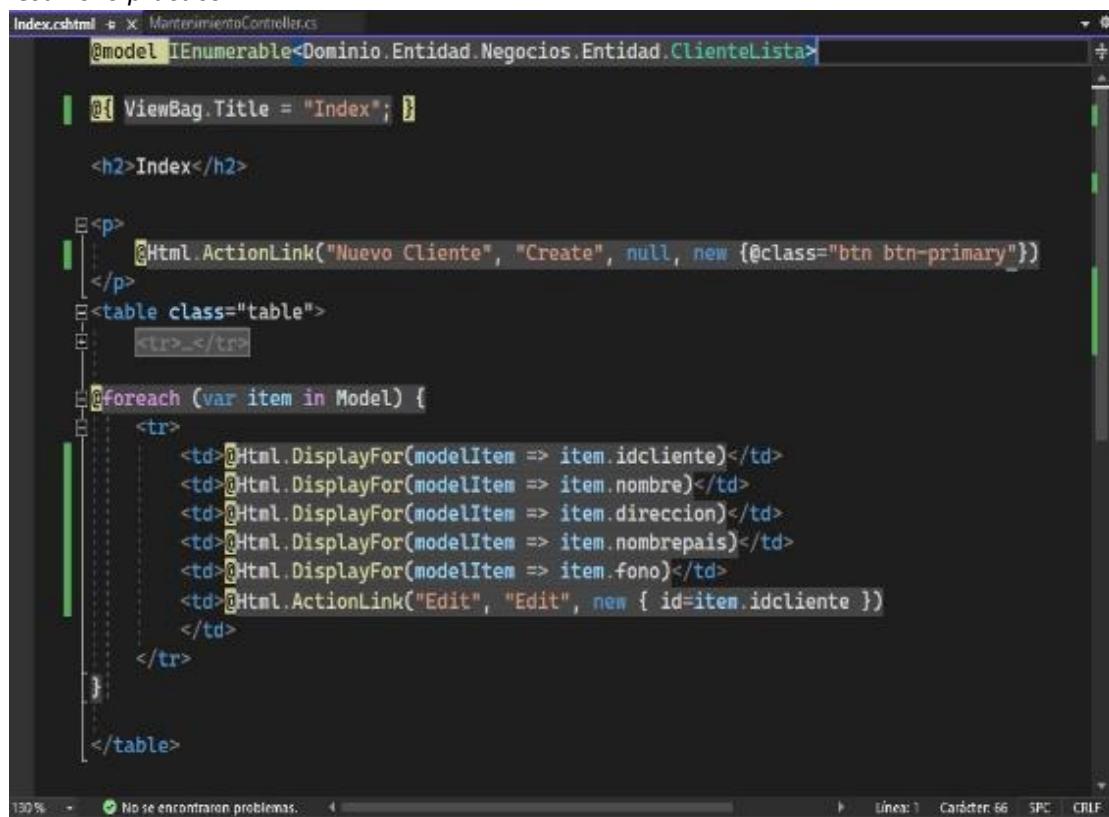
Figura 274

Desarrollo práctico



Nota. Elaboración propia.

Diseña la vista Index, tal como se muestra.

Figura 275*Desarrollo práctico*


```

Index.cshtml
@model IEnumerable<Dominio.Entidad.Negocios.Entidad.ClienteLista>

@{ ViewBag.Title = "Index"; }



## Index



Nuevo Cliente

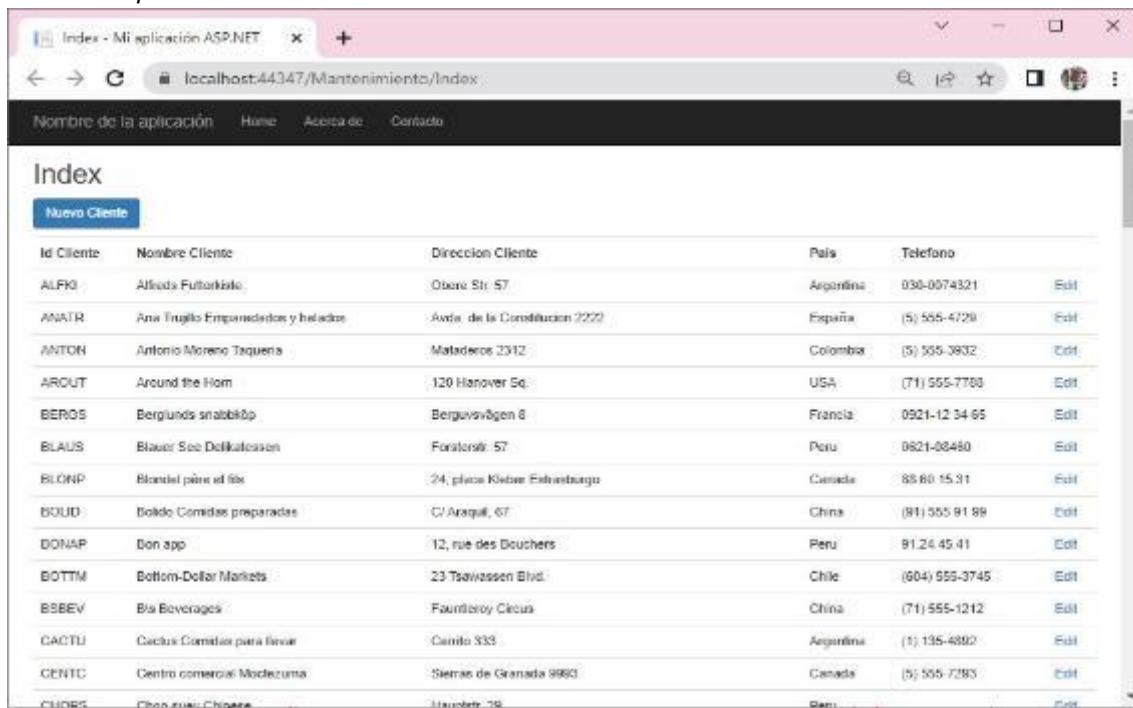


| ID Cliente | Nombre Cliente                      | Dirección Cliente            | País      | Teléfono       | Editar               |
|------------|-------------------------------------|------------------------------|-----------|----------------|----------------------|
| ALFKI      | Alfreds Futterkiste                 | Oberkampfstr. 57             | Argentina | 030-0974321    | <a href="#">Edit</a> |
| ANATR      | Ana Trujillo Empedrado y Hijo Ltda. | Ave. de la Constitución 2222 | España    | (5) 555-4729   | <a href="#">Edit</a> |
| ANTON      | Antonio Moreno Taquera              | Mataderos 2312               | Colombia  | (5) 555-3932   | <a href="#">Edit</a> |
| AROUT      | Around the Horn                     | 120 Hanover Sq.              | USA       | (71) 555-7788  | <a href="#">Edit</a> |
| BERGS      | Berglunds snabbköp                  | Berguvsvägen 8               | Francia   | 0921-12 34 65  | <a href="#">Edit</a> |
| BLAUS      | Blauer See Delikatessen             | Fasanvägen 57                | Perú      | 0821-08480     | <a href="#">Edit</a> |
| BLONP      | Blondel pâtes et fûts               | 24, place Kléber, Strasbourg | Canadá    | 88 80 15 31    | <a href="#">Edit</a> |
| BOUD       | Bolido Comidas preparadas           | C/ Atocha, 67                | China     | (81) 555 81 89 | <a href="#">Edit</a> |
| DONAP      | Don app                             | 12, rue des Douchers         | Perú      | 81 24 45 41    | <a href="#">Edit</a> |
| BOTTM      | Bottom-Dollar Markets               | 23 Tsawassen Blvd.           | Chile     | (604) 555-3745 | <a href="#">Edit</a> |
| BSBEV      | B's Beverages                       | Faximile: Círculo            | China     | (71) 555-1212  | <a href="#">Edit</a> |
| CACTU      | Cactus Comidas para llevar          | Camino 333                   | Argentina | (1) 135-4892   | <a href="#">Edit</a> |
| CENTC      | Centro comercial Moctezuma          | Sierras de Granada 9990      | Canadá    | (5) 555-7293   | <a href="#">Edit</a> |
| CHOPS      | Chop-suey Chophouse                 | Uxmalader 29                 | Rusia     | 800-123-4567   | <a href="#">Edit</a> |


```

Nota. Elaboración propia.

Ejecuta la Vista donde visualizamos los registros de los clientes.

Figura 276*Desarrollo práctico*


ID Cliente	Nombre Cliente	Dirección Cliente	País	Teléfono	Editar
ALFKI	Alfreds Futterkiste	Oberkampfstr. 57	Argentina	030-0974321	Edit
ANATR	Ana Trujillo Empedrado y Hijo Ltda.	Ave. de la Constitución 2222	España	(5) 555-4729	Edit
ANTON	Antonio Moreno Taquera	Mataderos 2312	Colombia	(5) 555-3932	Edit
AROUT	Around the Horn	120 Hanover Sq.	USA	(71) 555-7788	Edit
BERGS	Berglunds snabbköp	Berguvsvägen 8	Francia	0921-12 34 65	Edit
BLAUS	Blauer See Delikatessen	Fasanvägen 57	Perú	0821-08480	Edit
BLONP	Blondel pâtes et fûts	24, place Kléber, Strasbourg	Canadá	88 80 15 31	Edit
BOUD	Bolido Comidas preparadas	C/ Atocha, 67	China	(81) 555 81 89	Edit
DONAP	Don app	12, rue des Douchers	Perú	81 24 45 41	Edit
BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	Chile	(604) 555-3745	Edit
BSBEV	B's Beverages	Faximile: Círculo	China	(71) 555-1212	Edit
CACTU	Cactus Comidas para llevar	Camino 333	Argentina	(1) 135-4892	Edit
CENTC	Centro comercial Moctezuma	Sierras de Granada 9990	Canadá	(5) 555-7293	Edit
CHOPS	Chop-suey Chophouse	Uxmalader 29	Rusia	800-123-4567	Edit

Nota. Elaboración propia.

Trabajando con el ActionResult Create

Get: envía la lista de países en un ViewBag, y envía a la vista un nuevo Cliente.

Post: ejecuta el método Agregar, donde retorna el mensaje, y envío la lista de países y el registro del cliente.

Figura 277

Desarrollo práctico

```

Index.cshtml      MantenimientoController.cs
-----          -----
11 12           public class MantenimientoController : Controller
13 14           {
15 16               paisDTO _pais=new paisDTO();
17 18               clienteDTO _cliente=new clienteDTO();
19 20           public ActionResult Index()
21 22           {
23 24               ViewBag.paises=new SelectList(_pais.getPaíses(),"idpais","nombrepais");
25 26               return View(new Cliente());
27 28           }
29 30       }
31 32   }

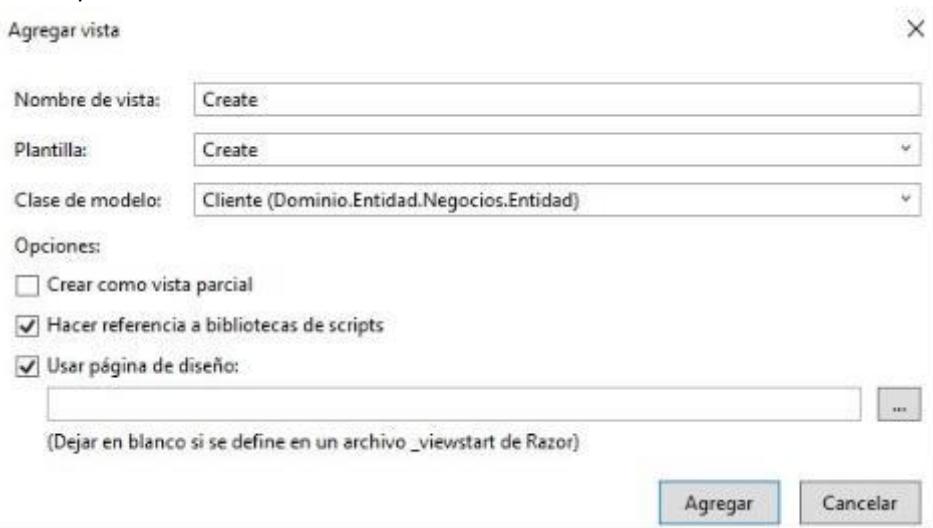
[HttpPost]
public ActionResult Create(Cliente reg)
{
    ViewBag.mensaje = _cliente.Agregar(reg);
    ViewBag.paises = new SelectList(_pais.getPaíses(), "idpais", "nombrepais",reg.idpais);
    return View(reg);
}

```

Nota. Elaboración propia.

Agregamos la vista del Action Create, tal como se muestra.

Figura 278
Desarrollo práctico



Nota. Elaboración propia.

Rediseña la vista, tal como se muestra.

Figura 279
Desarrollo práctico

```

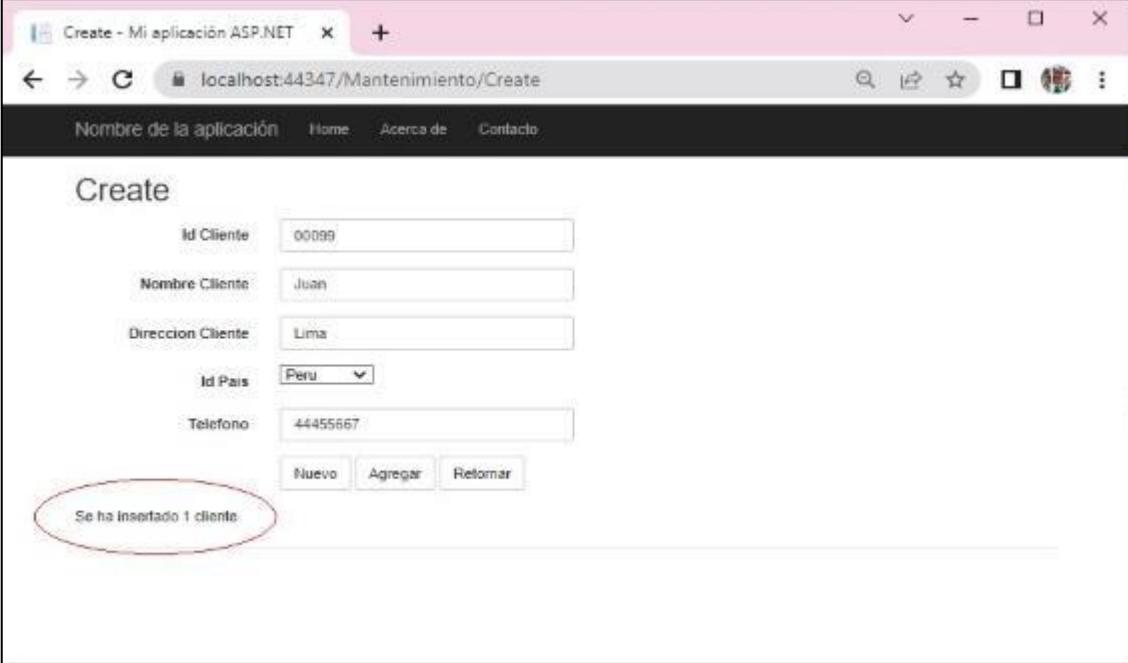
@using (Html.BeginForm())
{
    <div class="form-horizontal">
        <div class="form-group">
            <div class="col-md-2">
                @Html.LabelFor(model => model.idpais, htmlAttributes: new { @class = "control-label col-md-2" })
            </div>
            <div class="col-md-10">
                @Html.DropDownList("idpais", ViewBag.paises as SelectList)
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                @Html.ActionLink("Nuevo", "Create", null, new { @class = "btn btn-default" })
                <input type="submit" value="Agregar" class="btn btn-default" />
                @Html.ActionLink("Retornar", "Index", null, new { @class = "btn btn-default" })
            </div>
        </div>
    </div>
    <div>
        @ViewBag.mensaje
    </div>
}

```

Nota. Elaboración propia.

Ejecuta la vista, ingresa los datos, al presionar el botón Create, visualizamos el mensaje de confirmación del proceso, tal como se muestra.

Figura 280
Desarrollo práctico



The screenshot shows a web browser window titled "Create - Mi aplicación ASP.NET". The address bar displays "localhost:44347/Mantenimiento/Create". The page has a header with links to "Nombre de la aplicación", "Home", "Acerca de", and "Contacto". The main content area is titled "Create" and contains a form for adding a client. The form includes fields for "Id Cliente" (value: 00099), "Nombre Cliente" (value: Juan), "Dirección Cliente" (value: Lima), "Id País" (dropdown menu showing Peru), and "Teléfono" (value: 44455667). Below the form, there are three buttons: "Nuevo", "Agregar", and "Retornar". A message "Se ha insertado 1 cliente" is displayed in an oval shape. The browser interface includes standard navigation buttons (back, forward, search) and a toolbar.

Nota. Elaboración propia.

Trabajando con el ActionResult Edit

Get: envía la lista de países en un ViewBag, y envía a la vista un nuevo Cliente

Post: ejecuta el método Agregar, donde retorna el mensaje, y envío la lista de países y el registro del cliente.

Figura 281
Desarrollo práctico

```

12     public class MantenimientoController : Controller
13     {
14         paisDTO _pais=new paisDTO();
15         clienteDTO _cliente=new clienteDTO();
16         public ActionResult Index()
17         {
18             ViewBag.paises = new SelectList(_pais.getPaíses(), "idpais", "nombrepais", reg.idpais);
19             return View(reg);
20         }
21         [HttpPost]
22         public ActionResult Edit(Cliente reg)
23         {
24             ViewBag.mensaje = _cliente.Actualizar(reg);
25             ViewBag.paises = new SelectList(_pais.getPaíses(), "idpais", "nombrepais", reg.idpais);
26             return View(reg);
27         }
28     }

```

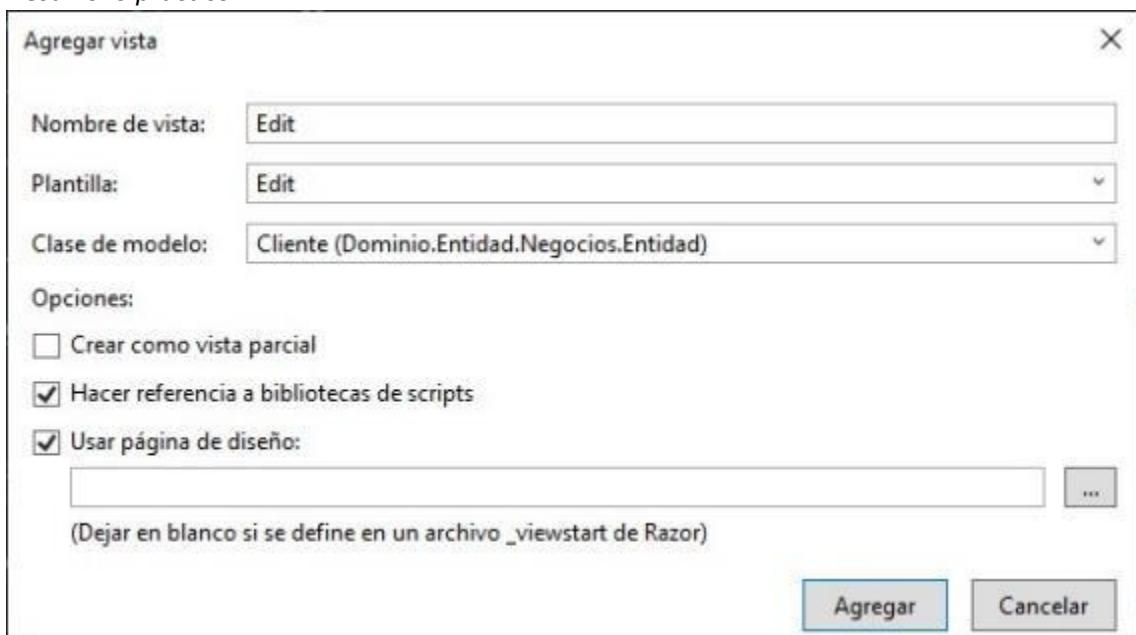
Get envía los países y el cliente seleccionado

Post recibe los datos y ejecuta el método

Nota. Elaboración propia.

Agregamos la vista del Action Edit, tal como se muestra.

Figura 282
Desarrollo práctico



Nota. Elaboración propia.

Rediseña la vista, tal como se muestra.

Figura 283*Desarrollo práctico*

```

@using (Html.BeginForm())
{
    <div class="form-horizontal">

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

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

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

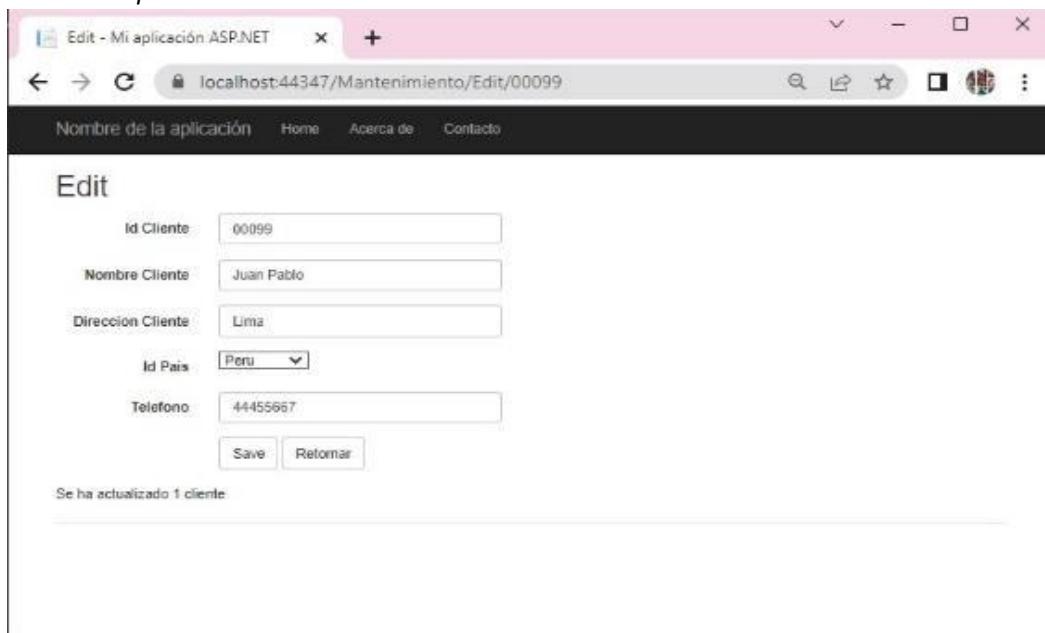
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
                @Html.ActionLink("Retornar", "Index", null, new { @class = "btn btn-default" })
            </div>
        </div>
    </div>

    <div>
        @ViewBag.mensaje
    </div>
}

```

Nota. Elaboración propia.

Ejecuta la vista Index, selecciona un cliente desde la opción EDIT, modifique los datos, al presionar el botón Save, visualizamos el mensaje de confirmación del proceso, tal como se muestra.

Figura 284*Desarrollo práctico**Nota.* Elaboración propia.

Resumen

1. El dominio es modelo de una aplicación y su comportamiento, donde se definen los procesos y la reglas de negocio al que está 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.
2. 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 Ubícuo).
3. 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.
4. 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.

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <http://nfanjul.blogspot.com/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.com/2014/01/ddd-la-logica-de-dominio-es-el-corazon.html>
- <http://www.eltavo.net/2014/08/patrones-implementando-patron.html>



IMPLEMENTANDO E-COMMERCE EN ASP.NET MVC

LOGRO DE LA UNIDAD DE APRENDIZAJE

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

TEMARIO

- 3.1 Tema 7 : Implementando una aplicación e-commerce**
 - 3.1.1 : Proceso del e-commerce
 - 3.1.2 : Persistencia de datos: uso del objeto Session, TempData
 - 3.1.3 : Implementando el proceso del e-commerce: diseño, acceso a datos, lógica de negocios, validación de usuario y transacciones

ACTIVIDADES PROPUESTAS

- Los alumnos crean aplicación e-commerce para realizar una transacción comercial.
- Los alumnos desarrollan el laboratorio de la semana.

3.1. IMPLEMENTANDO UNA APLICACIÓN E-COMMERCE

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 285
e-commerce



Nota. Tomado de Tendencias de crecimiento del e-commerce en México 2020, por DPL News, 2020, (<https://dplnews.com/tendencias-de-crecimiento-del-e-commerce-en-mexico-2020/>)

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. (IHMC Public Cmaps, s.f.)

“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” (Improdex Desarrollo Empresarial, s.f.).

Figura 286*Definición de e-commerce*

Nota. Tomado de *Qué es el eCommerce: definición modelos y ventajas*, por Diego Enrique Uzeta Ovalle, 2021, LinkedIn, (<https://es.linkedin.com/pulse/qu%C3%A9-es-el-eCommerce-definici%C3%B3n-modelos-y-ventajas-uzeta-ovalle>)

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
- fidelizar
- vender 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. (Redacción Albanova, s.f.).

3.1.1. 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 un conjunto de 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 287
Proceso de e-commerce



Nota. Tomado de *Proceso de comercio en línea*, por Freepik, s.f., (https://www.freepik.es/vector-gratis/proceso-comercio-linea_1359031.htm)

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 288
Carrito de compras



Nota. Tomado de [¿Cómo hacer una página web con carrito de compras?](https://www.tiendanube.com/blog/pagina-web-con-carrito-de-compras/), por Luma León, 2024, (<https://www.tiendanube.com/blog/pagina-web-con-carrito-de-compras/>)

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. (Tiendas Virtuales, s.f.)

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 los Commerce o una amplia variedad de carritos de compras pagos.

3.1.2. Persistencia de datos: uso del objeto Session, TempData

En los programas que hemos visto hasta ahora, hemos utilizado variables que solo existían en el archivo que era ejecutado. Cuando cargamos otra página distinta, los valores de estas variables se perdían a menos que los pasásemos a través de la URL o inscribirlos en las cookies o en un formulario para su posterior explotación. Estos métodos, aunque útiles, no son todo lo prácticos que podrían en determinados casos en los que la variable que queremos conservar ha de ser utilizada en varios scripts diferentes y distantes los unos de los otros.

Podríamos pensar que ese problema puede quedar resuelto con las cookies ya que se trata de variables que pueden ser invocadas en cualquier momento. El problema, ya lo hemos dicho, es

que las cookies no son aceptadas ni por la totalidad de los usuarios ni por la totalidad de los navegadores lo cual implica que una aplicación que se sirviera de las cookies para pasar variables de un archivo a otro no sería 100% infalible.

Nos resulta pues necesario el poder declarar ciertas variables que puedan ser reutilizadas tantas veces como queramos dentro de una misma sesión. Imaginemos un sitio multilingüe en el que cada vez que queremos imprimir un mensaje en cualquier página necesitamos saber en qué idioma debe hacerse. Podríamos introducir un script identificador de la lengua del navegador en cada uno de los archivos o bien declarar una variable que fuese válida para toda la sesión y que tuviese como valor el idioma reconocido en un primer momento. (Alvarez, 2021)

Estas variables que son válidas durante una sesión y que luego son "olvidadas" son definidas con el objeto **Session** de la siguiente forma:

Session[*"nombre de la variable"*] = *valor de la variable*

Una vez definida, la variable **Session**, será almacenada en memoria y podrá ser empleada en cualquier script del sitio web.

La duración de una sesión viene definida por defecto en 20 minutos. Esto quiere decir que, si en 20 minutos no realizamos ninguna acción, el servidor dará por finalizada la sesión y todas las variables Session serán abandonadas. Esta duración puede ser modificada con la propiedad **Timeout**:

Session.Timeout = *nº de minutos que queramos que dure*

Una forma de borrar las variables Session sin necesidad de esperar que pase este plazo es a partir del método **Abandon**:

Session.Abandon

De este modo todas las variables Session serán borradas y la sesión se dará por finalizada. Este método puede resultar práctico cuando estemos haciendo pruebas con el script y necesitemos reiniciar las variables.

TempData es una propiedad donde pasamos los datos de un request a otro, donde es su principal cualidad, la duración hasta el siguiente request, es decir que si estamos en una acción (dentro del controller), podemos guardar el dato y utilizarlo en el siguiente request, al igual que el **ViewData**, este se utiliza como un vector.

3.1.3. Implementando el proceso del e-commerce: diseño, acceso a datos, lógica de negocios, validación de usuario y transacciones

Figura 289

Implementando el carrito de compras



Nota. Tomado de Mejores Prácticas para el Carrito de Compras en Tiendas Online, por Felipe, 2013, (<https://www.hostingplus.pe/blog/mejores-practicas-para-el-carrito-de-compras-en-tiendas-online/>)

1. Primero a diferencia del carrito del super el de nosotros no es necesario que se tome (crearlo) al principio; no más llegue a nuestro sitio el cliente. Sino que el cliente puede revisar el catálogo de productos y hasta que esté 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 producto a 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 todos sus productos esto lo hace mentalmente, de esta forma decide si dejar algo porque 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 último en el carro de compras me debe permitir actualizar la cantidad de producto si quiero más de un producto del mismo tipo o quiero dejar de ese producto uno.

LABORATORIO 7.1.: Implementando el proceso del e-commerce en ASP.NET MVC

Implemente una aplicación ASP.NET MVC que permita registrar las ventas por internet.

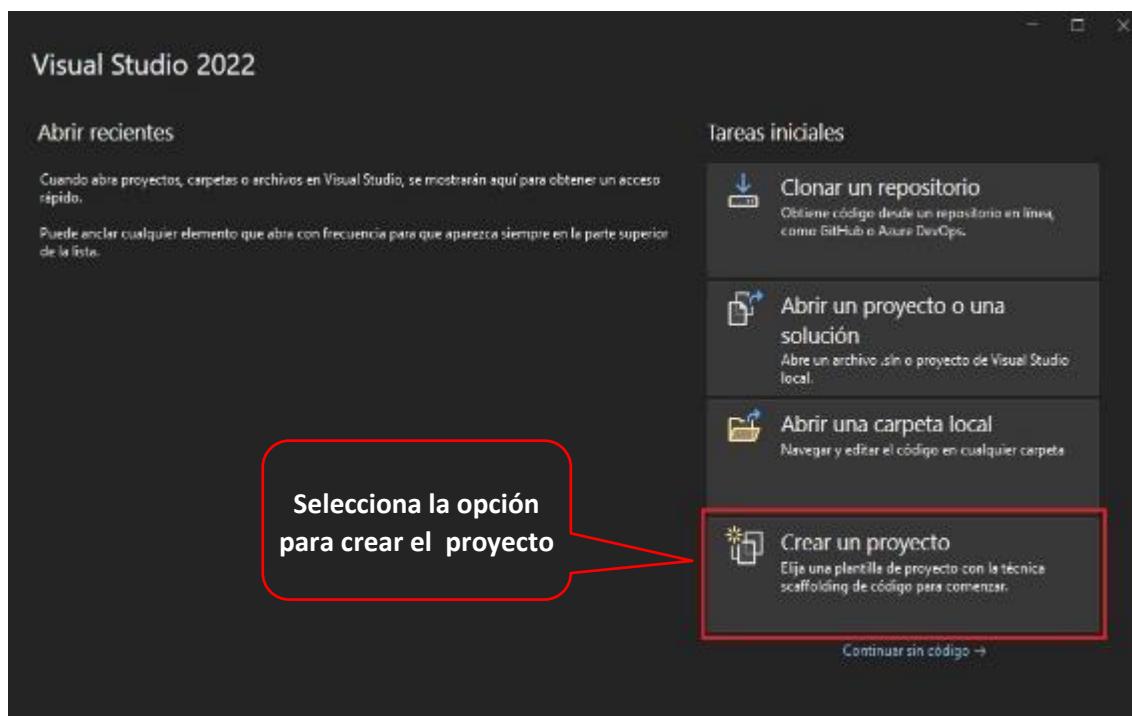
Creando un Proyecto ASP.NET MVC

Inicio del Proyecto

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

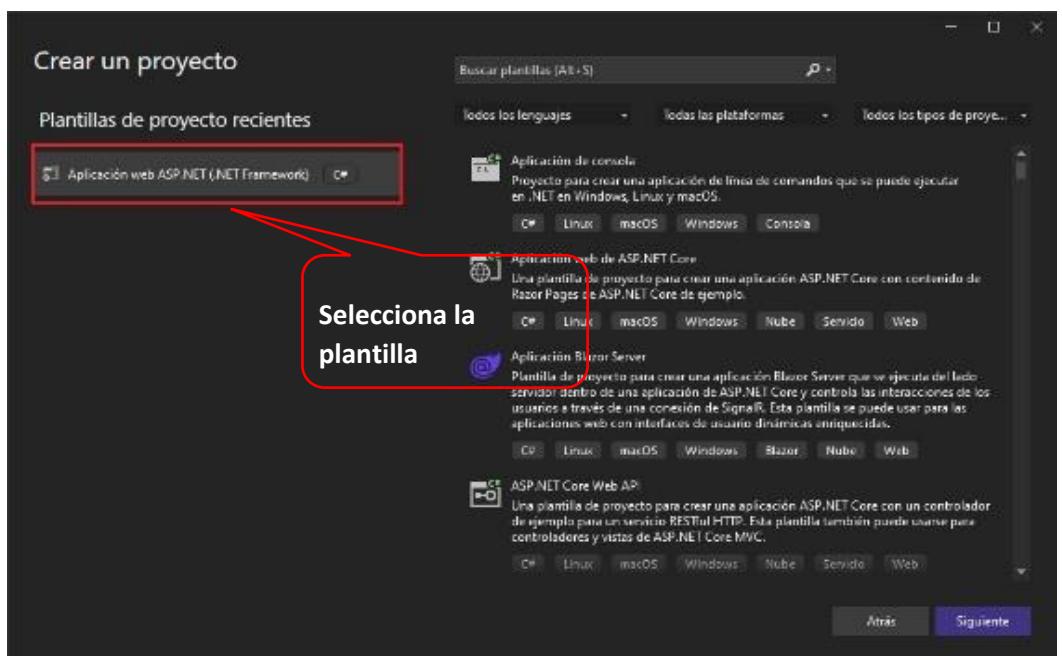
Figura 290

Desarrollo Práctico

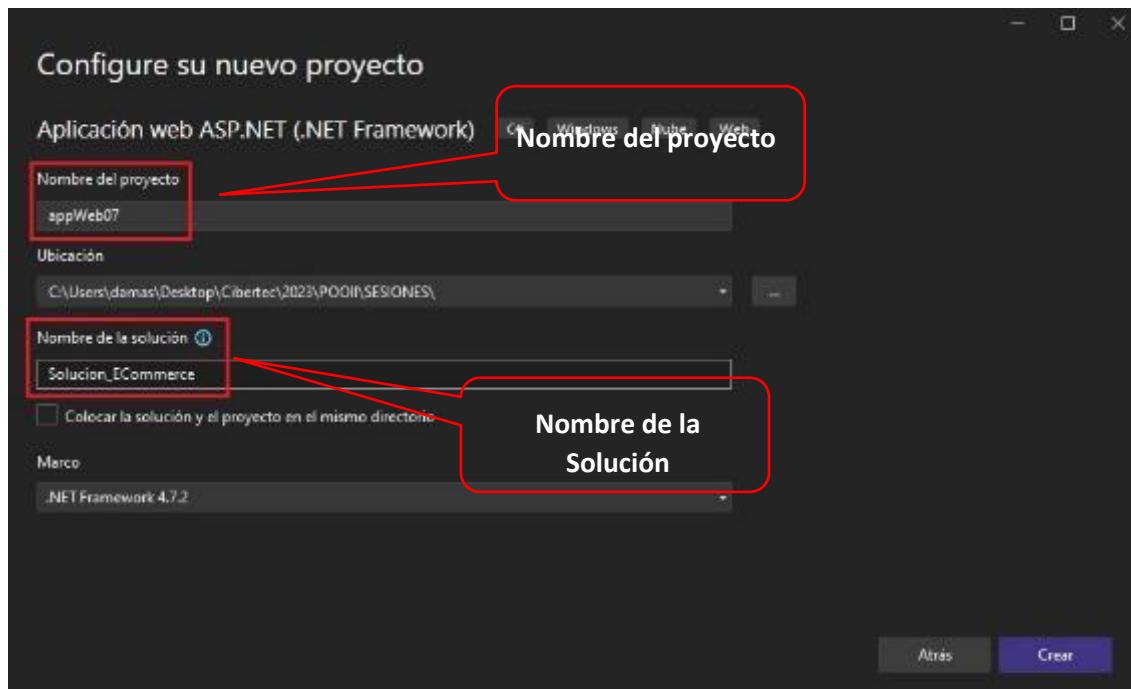


Nota. Elaboración propia.

Selecciona la plantilla de la aplicación ASP.NET(.NET Framework), presionar la opción siguiente:

Figura 291*Desarrollo Práctico**Nota. Elaboración propia.*

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciona la ubicación del mismo, y el nombre de la solución, al terminar presiona la opción **Crear**.

Figura 292*Desarrollo Práctico**Nota. Elaboración propia.*

Para finalizar, selecciona la plantilla de tipo de proyecto MVC, tal como se muestra. Presiona el botón CREAR.

Figura 293
Desarrollo Práctico

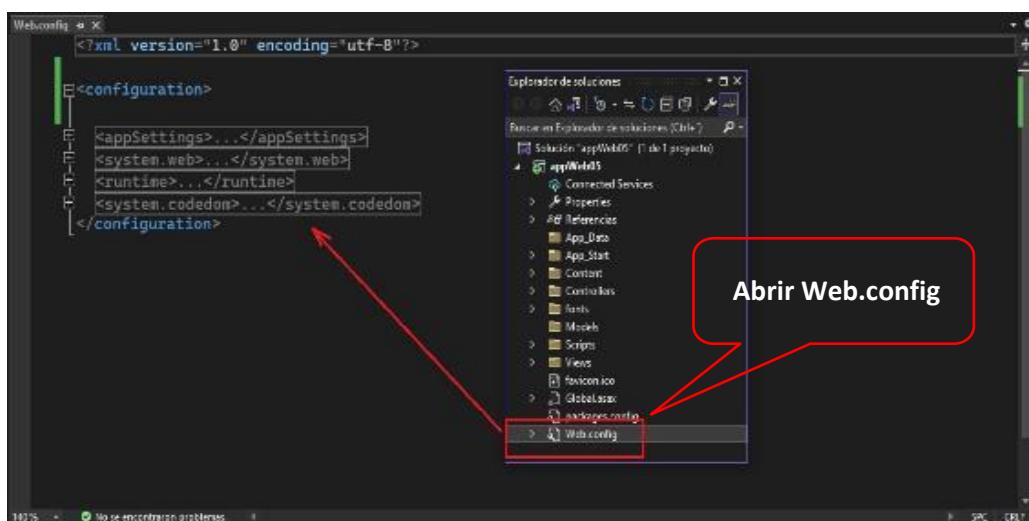


Nota. Elaboración propia.

Publicando la cadena de conexión

Desde el Explorador de proyecto, abrir el archivo Web.config, tal como se muestra.

Figura 294
Desarrollo Práctico



Nota. Elaboración propia.

Defina la etiqueta <connectionStrings> agregando una cadena <add> cuyo nombre es “cadena” y definiendo la cadena de conexión, tal como se muestra.

Figura 295*Desarrollo Práctico*

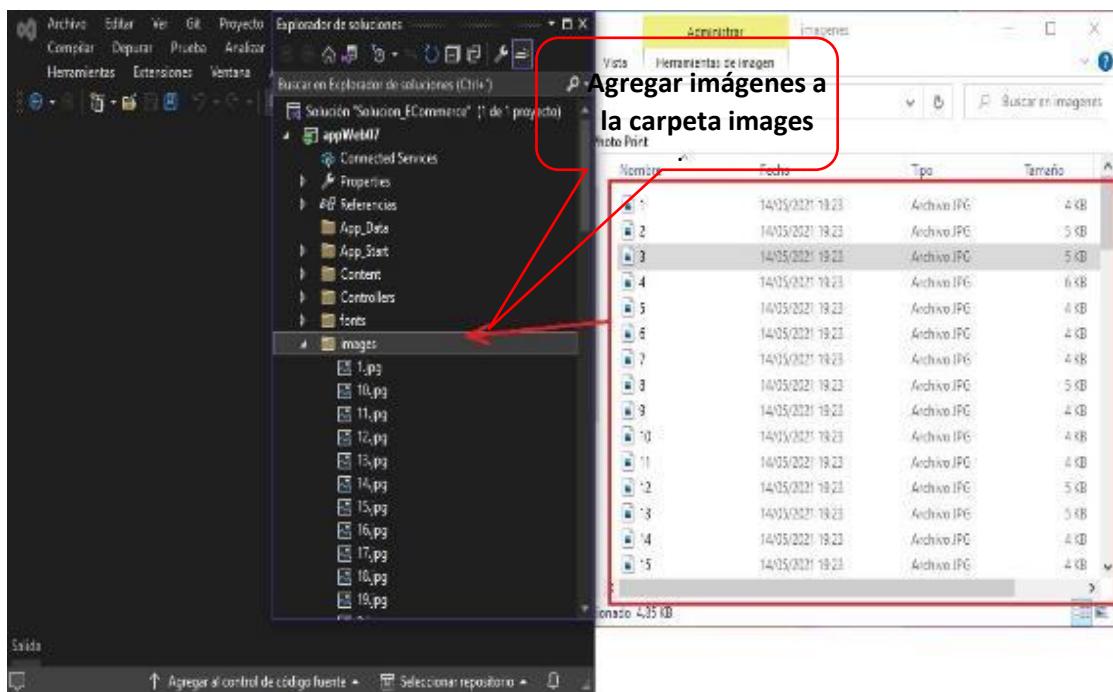
```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="cadena"
      connectionString="server=.; database=Negocios2022; integrated security=true"/>
  </connectionStrings>
  <appSettings>...</appSettings>
  <system.web>...</system.web>
  <runtime>...</runtime>
  <system.codedom>...</system.codedom>
</configuration>

```

Nota. Elaboración propia.**Agregando Imágenes al Proyecto**

En el proyecto agregar la carpeta images, arrastre los archivos de los productos que vamos a listar en el sitio de ventas.

Figura 296*Desarrollo Práctico**Nota.* Elaboración propia.**Agregando las clases para la plataforma de Ventas**

En la carpeta Models agregar la clase Producto, para listar los productos en la plataforma de ventas y su selección.

Figura 297*Desarrollo Práctico*

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace appWeb07.Models
7  {
8      public class Producto
9      {
10         public int codigo { get; set; }
11         public string descripcion { get; set; }
12         public string categoria { get; set; }
13         public decimal precio { get; set; }
14         public int stock { get; set; }
15     }
16 }

```

121% No se encontraron problemas. | Línea: 1 SPC CRLF

Nota. Elaboración propia.

A continuación, agregar la clase Registro que será utilizado en el Session para almacenar los productos seleccionados.

Figura 298*Desarrollo Práctico*

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5
6  namespace appWeb07.Models
7  {
8      public class Registro
9      {
10         public int codigo { get; set; }
11         public string descripcion { get; set; }
12         public string categoria { get; set; }
13         public decimal precio { get; set; }
14         public int cantidad { get; set; } /*cantidad del pedido*/
15         public decimal monto { get { return precio * cantidad; } } /*valor calculado*/
16     }
17 }

```

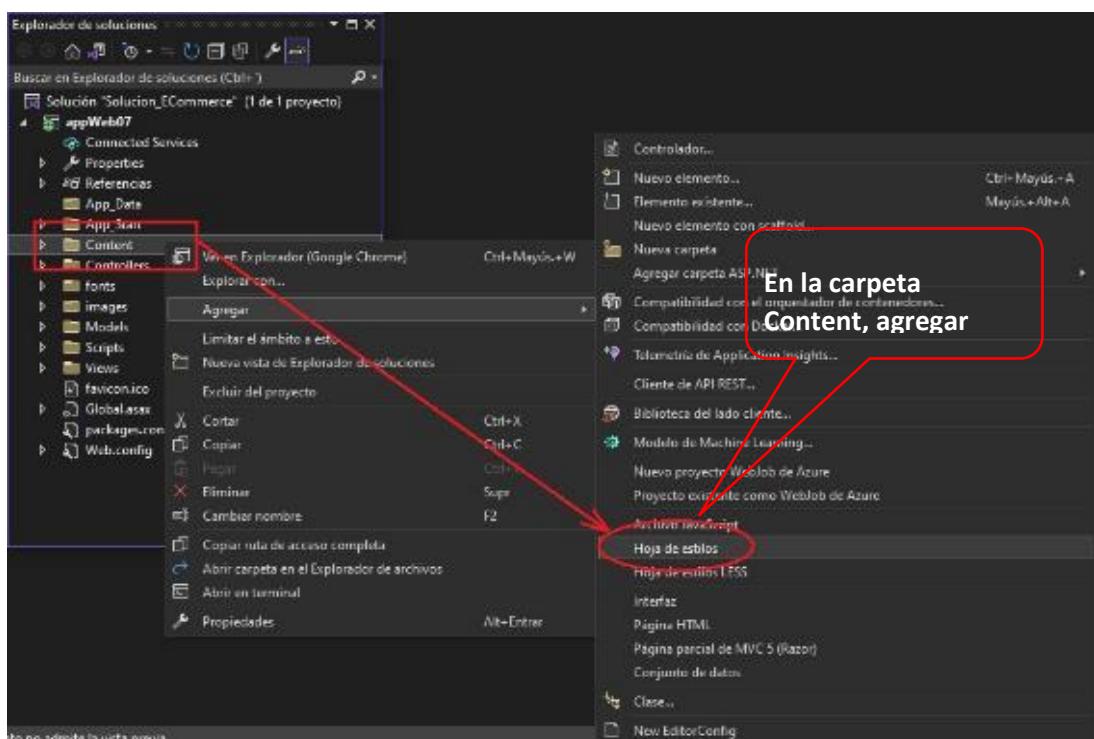
121% No se encontraron problemas. | Línea: 1 Código 14 SPC CRLF

Nota. Elaboración propia.

Agregando una hoja de estilo

En la carpeta Content de nuestro proyecto ASP.NET MVC agregar una hoja de Estilo.

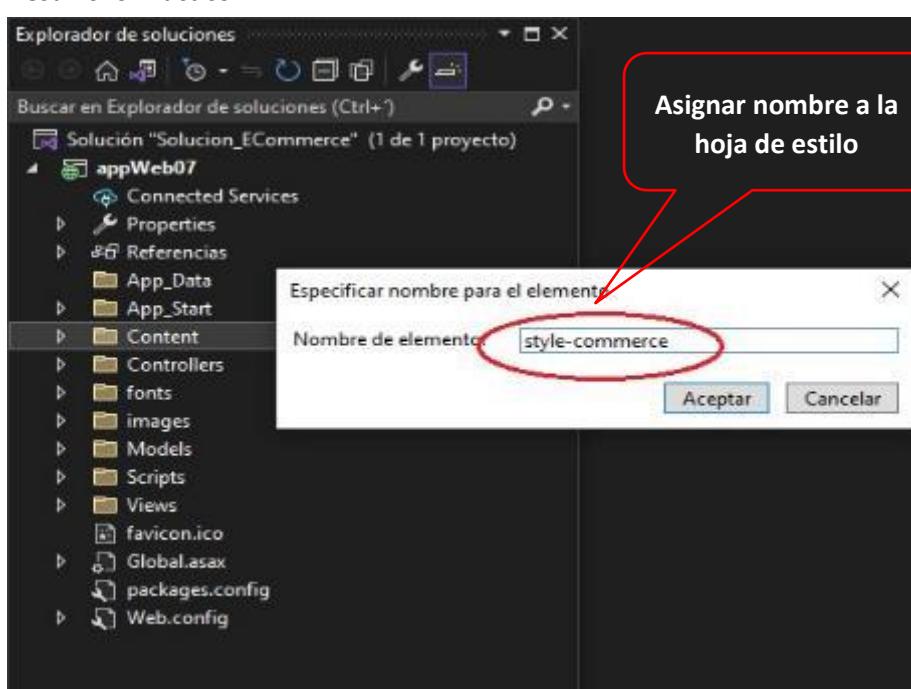
Figura 299
Desarrollo Práctico



Nota. Elaboración propia.

Asigne el nombre del elemento: style-commerce, presiona el botón **Aceptar**.

Figura 300
Desarrollo Práctico



Nota. Elaboración propia.

Defina los selectores que se encuentran en la hoja de estilo tal como se muestra:

Figura 301

Desarrollo Práctico

```
style-commerce.css  ✘ X
.portal {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    column-gap: 3px;
    row-gap: 3px;
    background-color: salmon;
}

.registro {
    min-height: 300px;
    text-align: center;
    background-color: white;
    margin: 3%;
}

.deshabilita {
    visibility: hidden;
}

h2 {
    text-align: center;
}

img {
    width: 100px;
    height: 100px;
    margin-left: auto;
    margin-right: auto;
}

.div-agregar {
    display: grid;
    grid-template-columns: repeat(2, 1fr);
    width: 80%;
    margin: 0 10% 10px 10%;
}

.img-producto{
    width: 80%;
    margin: 0 10% 0 10%;
    border-radius: 20px;
    height: 300px
}

.div-agregar-form {
    position: relative;
    width: 80%;
    margin: 10px 10% 10px 10%;
}
```

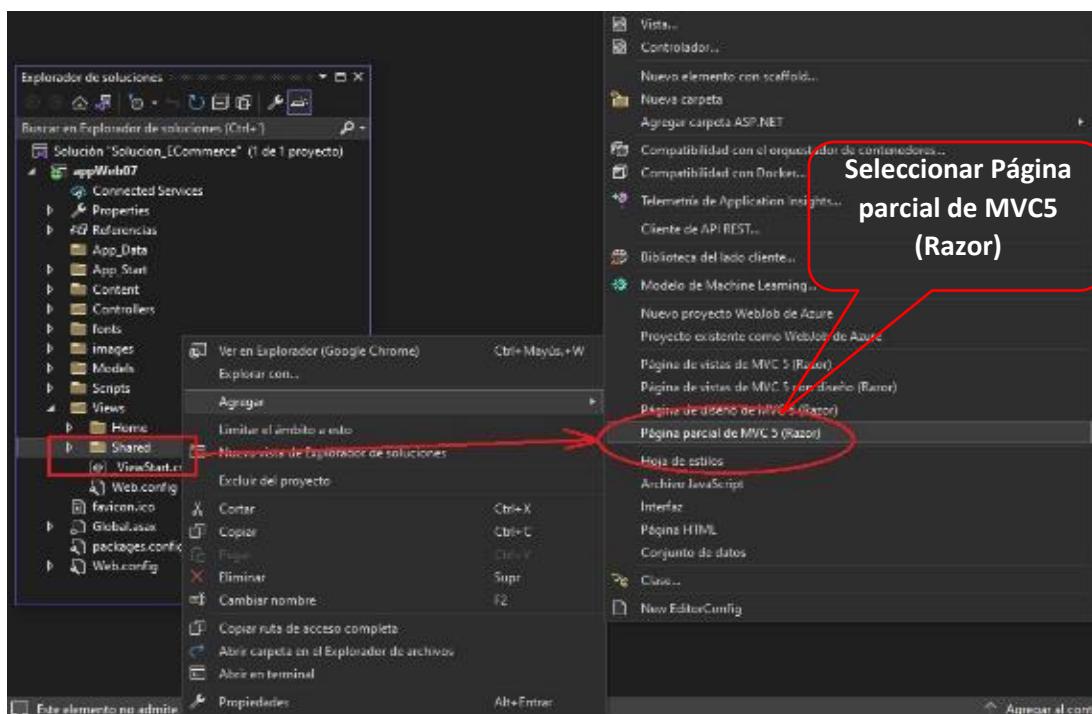
130% No se encontraron problemas. Lines: 15 Carácter: 1 SPC CRLF

Nota. Elaboración propia.

Agregando una Vista Parcial

En la carpeta Shared, agrega una Página parcial de MVC5 (Razor), llamada _PartialVenta, tal como se muestra.

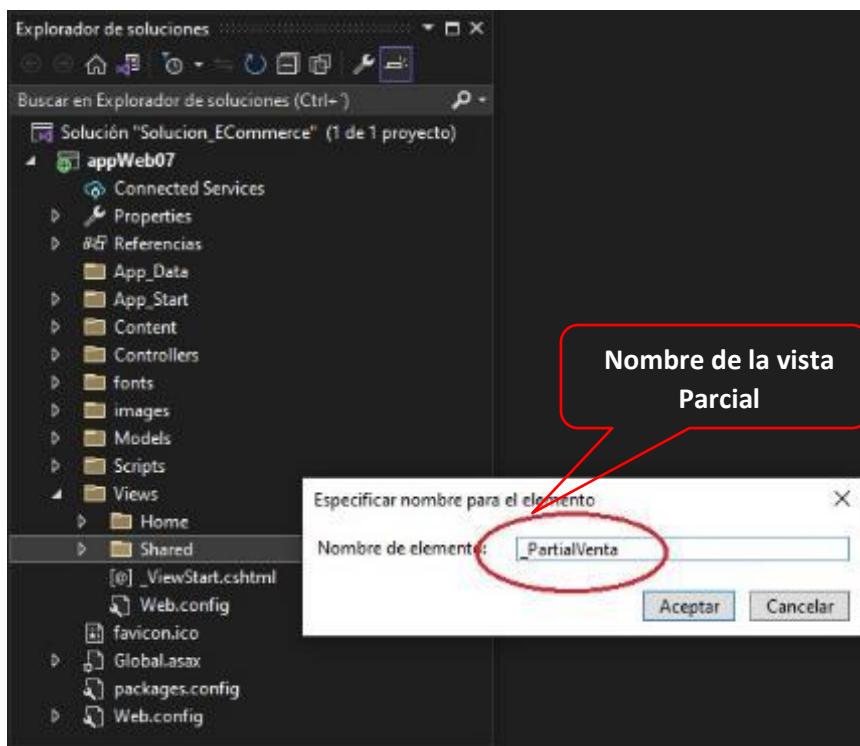
Figura 302
Desarrollo Práctico



Nota. Elaboración propia.

Asignar el nombre a la Vista Parcial, presionar el botón **Aceptar**.

Figura 303
Desarrollo Práctico



Nota. Elaboración propia.

En la clase partial defina el diseño de la página para listar los productos, en el ActionResult Agregar solo se visualizará si el valor de stock es mayor a cero, tal como se muestra.

Figura 304

Desarrollo Práctico

```

@model IEnumerable<appWeb07.Models.Producto>
<link rel="stylesheet" href="~/Content/style-commerce.css" />

<p><Html.ActionLink("Ir al Carrito", "Carrito", null, new { @class = "btn btn-success" })</p>

<div class="portal">
    <foreach var item in Model>
    {
        <div class="registro">
            <em>Codigo:</em><Html.DisplayFor(modelItem => item.codigo)>
            <br />
            <em>Descripción:</em><Html.DisplayFor(modelItem => item.descripcion)>
            <br />
            <em>Categoria:</em><Html.DisplayFor(modelItem => item.categoría)>
            <br />
            <em>Precio:</em><Html.DisplayFor(modelItem => item.precio)>
            <br />
            <em>Stock:</em><Html.DisplayFor(modelItem => item.stock)>
            <br />
            
            <br />
            <Html.ActionLink("Select", "Select", new { id = item.codigo }, new { @class = {item.stock == 0 ? "deshabilitado" : ""} })</div>
        }
    </div>

```

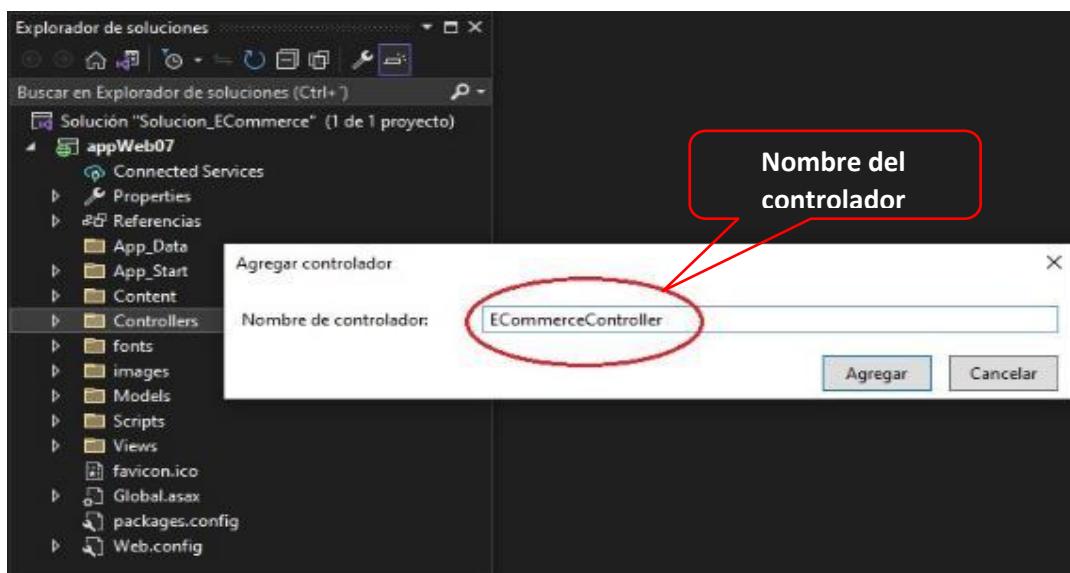
Nota. Elaboración propia.

Agregando un Controlador

A continuación, agregamos el controlador MVC5 en blanco llamado EcommerceController.

Figura 305

Desarrollo Práctico



Nota. Elaboración propia.

En el controlador importar las librerías para el desarrollo de la aplicación e-commerce.

Figura 306*Desarrollo Práctico*

```

ECommerceController.cs  X
appWeb07                appWeb07.Controllers.ECommerceController    productos()
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6  using System.Configuration;
7  using System.Data;
8  using System.Data.SqlClient;
9  using appWeb07.Models;
10 using Newtonsoft.Json;           Importar Newtonsoft.Json
11 namespace appWeb07.Controllers
12 {
13     public class ECommerceController : ...
14 }

```

130% No se encontraron problemas. | Lines:1 Character:14 SPC CRLF

Nota. Elaboración propia.

Dentro del controlador, defina una lista numerada llamada productos, donde retorna los registros de tb_productos ejecutando el procedimiento almacenado usp_productos.

Figura 307*Desarrollo Práctico*

```

ECommerceController.cs  X
appWeb07                appWeb07.Controllers.ECommerceController    productos()
14  IEnumerable<Producto> productos()
15  {
16      List<Producto> temporal = new List<Producto>();
17      using (SqlConnection cn = new SqlConnection(
18          ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
19      {
20          cn.Open();
21          SqlCommand cm = new SqlCommand("exec usp_productos", cn);
22          SqlDataReader dr = cm.ExecuteReader();
23          while (dr.Read())
24          {
25              Producto producto = new Producto()
26              {
27                  codigo = dr.GetInt32(0),
28                  descripcion = dr.GetString(1),
29                  categoria = dr.GetString(2),
30                  precio = dr.GetDecimal(3),
31                  stock = dr.GetInt32(4),
32              };
33              temporal.Add(producto);
34          }
35      }
36      return temporal;
37  }
38

```

130% No se encontraron problemas. | Lines:1 Character:14 SPC CRLF

Nota. Elaboración propia.

En el ActionResult Index, primero evaluamos si existe el Session["carrito"] para inicializarlo, este proceso se ejecuta una sola vez, luego enviamos a la vista la lista productos(), tal como se muestra.

Figura 308*Desarrollo Práctico*

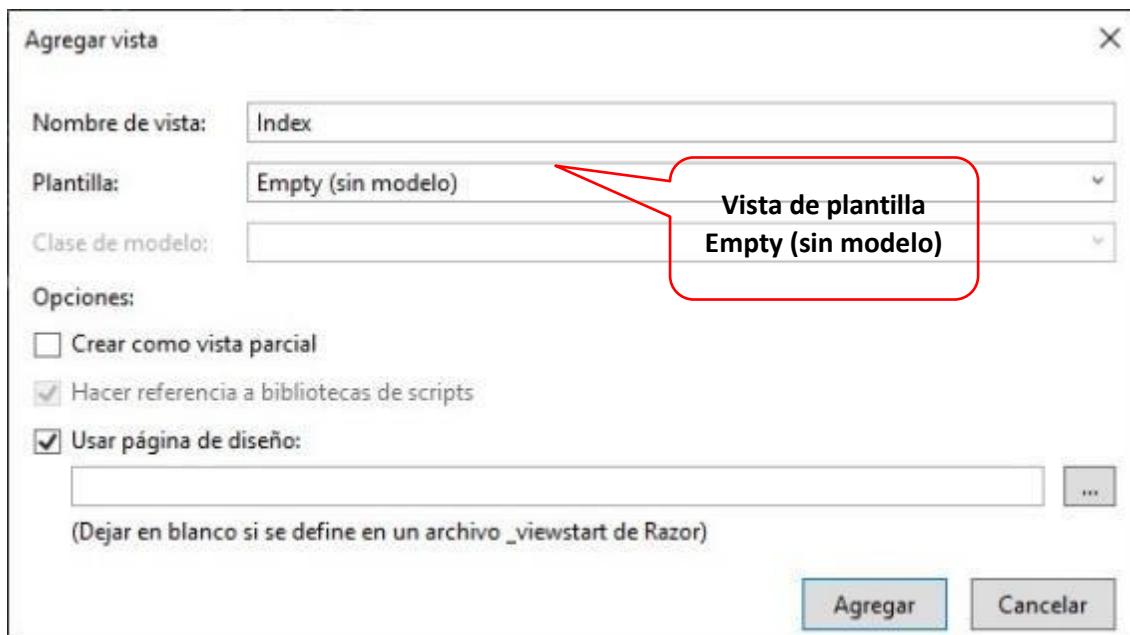
```

1  using ...
11 namespace appWeb07.Controllers
12 {
13     public class ECommerceController : Controller
14     {
15         IEaversable<Producto> productos();
16 
17         public ActionResult Index()
18         {
19             if (Session["carrito"] == null)
20             {
21                 Session["carrito"] = JsonConvert.SerializeObject(new List<Producto>());
22             }
23             return View(productos());
24         }
25     }
26 }

```

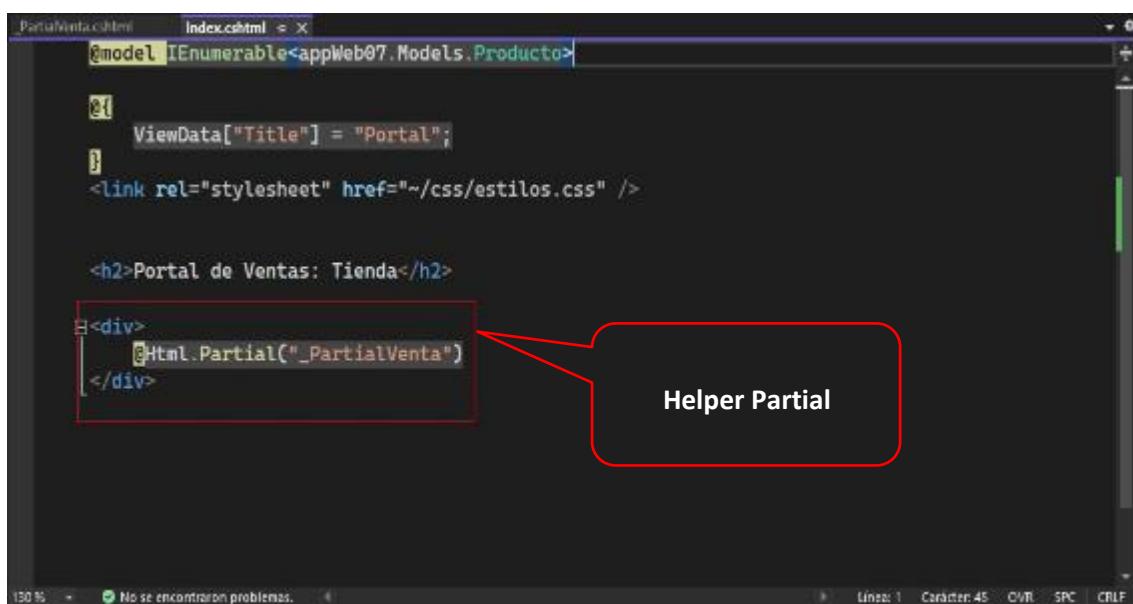
Nota. Elaboración propia.

Agregar la Vista Index, selecciona la plantilla Empty (sin modelo), tal como se muestra.

Figura 309*Desarrollo Práctico*

Nota. Elaboración propia.

A continuación, diseña la Vista del Action Index, donde agregar el Helper Partial para llamar a la vista parcial _PartialVenta.

Figura 310*Desarrollo Práctico*


```

@model IEnumerable<appWeb07.Models.Producto>

@{
    ViewData["Title"] = "Portal";
}

<link rel="stylesheet" href("~/css/estilos.css" />

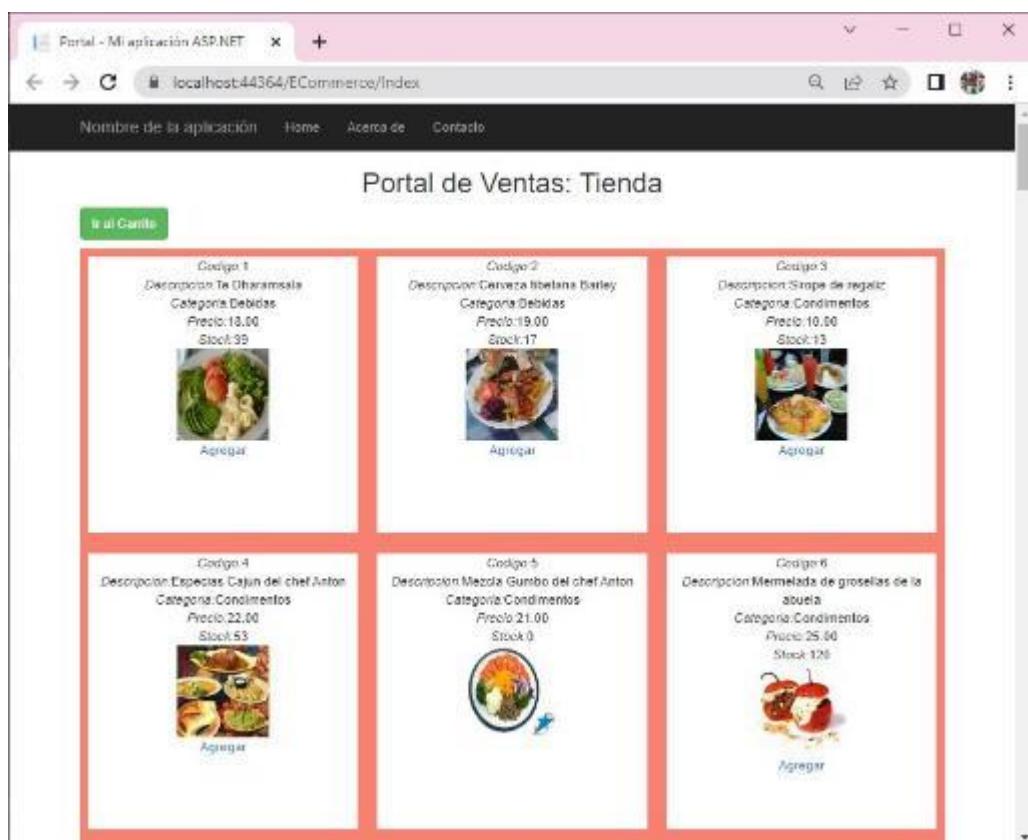
<h2>Portal de Ventas: Tienda</h2>

<div>
    @Html.Partial("_PartialVenta")
</div>

```

Nota. Elaboración propia.

Ejecuta la aplicación desde la Vista Index, donde se visualiza la lista de productos.

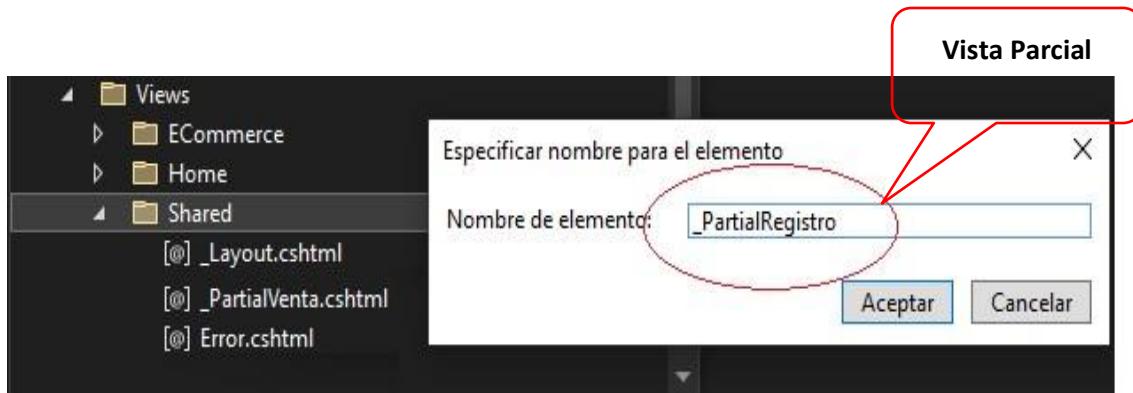
Figura 311*Desarrollo Práctico**Nota. Elaboración propia.*

Agregando una Vista Parcial para la Vista Select

En la carpeta Shared, agregar la Vista Parcial _PartialRegistro, tal como se muestra.

Figura 312

Desarrollo Práctico



Nota. Elaboración propia.

Diseña la Vista Parcial para visualizar los datos del producto seleccionar e ingresar la cantidad del producto a registrar.

Figura 313

Desarrollo Práctico

```

Index.cshtml      PartialView.cshtml  Select.cshtml
@model appWeb07.Models.Producto

<link rel="stylesheet" href="/Content/style-commerce.css" />


<dt>@Html.DisplayNameFor(model => model.codigo)</dt>
<dd>@Html.DisplayFor(model => model.codigo)</dd>
<dt>@Html.DisplayNameFor(model => model.descripcion)</dt>
<dd>@Html.DisplayFor(model => model.descripcion)</dd>
<dt>@Html.DisplayNameFor(model => model.categoría)</dt>
<dd>@Html.DisplayFor(model => model.categoría)</dd>
<dt>@Html.DisplayNameFor(model => model.precio)</dt>
<dd>@Html.DisplayFor(model => model.precio)</dd>
<dt>@Html.DisplayNameFor(model => model.stock)</dt>
<dd>@Html.DisplayFor(model => model.stock)</dd>






```

The screenshot shows a code editor with three tabs: 'Index.cshtml', 'PartialView.cshtml', and 'Select.cshtml'. The 'PartialView.cshtml' tab is active, displaying the following code:

```

Index.cshtml      PartialView.cshtml  Select.cshtml
@model appWeb07.Models.Producto

<link rel="stylesheet" href="/Content/style-commerce.css" />


<dt>@Html.DisplayNameFor(model => model.codigo)</dt>
<dd>@Html.DisplayFor(model => model.codigo)</dd>
<dt>@Html.DisplayNameFor(model => model.descripcion)</dt>
<dd>@Html.DisplayFor(model => model.descripcion)</dd>
<dt>@Html.DisplayNameFor(model => model.categoría)</dt>
<dd>@Html.DisplayFor(model => model.categoría)</dd>
<dt>@Html.DisplayNameFor(model => model.precio)</dt>
<dd>@Html.DisplayFor(model => model.precio)</dd>
<dt>@Html.DisplayNameFor(model => model.stock)</dt>
<dd>@Html.DisplayFor(model => model.stock)</dd>






```

Nota. Elaboración propia.

Trabajando con el ActionResult Select

En el controlador Ecommerce, defina el método buscar donde retorna el registro del producto por su campo código.

Figura 314*Desarrollo Práctico*

```

Index.cshtml      ECommerceController.cs  p X
appWeb07          appWeb07.Controllers.ECommerceController           S productos()
12   0 referencias
13
14   2 referencias
15     IEnumarable<Producto> productos()...
16
17   0 referencias
18     public ActionResult Index()...
19
20   0 referencias
21     Producto buscar(int id)
22   {
23       return productos().FirstOrDefault(c =>c.codigo== id);
24   }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

```

Nota. Elaboración propia.

Defina el ActionResult Select, método GET, donde envía el producto seleccionado a través del parámetro opcional id.

Figura 315*Desarrollo Práctico*

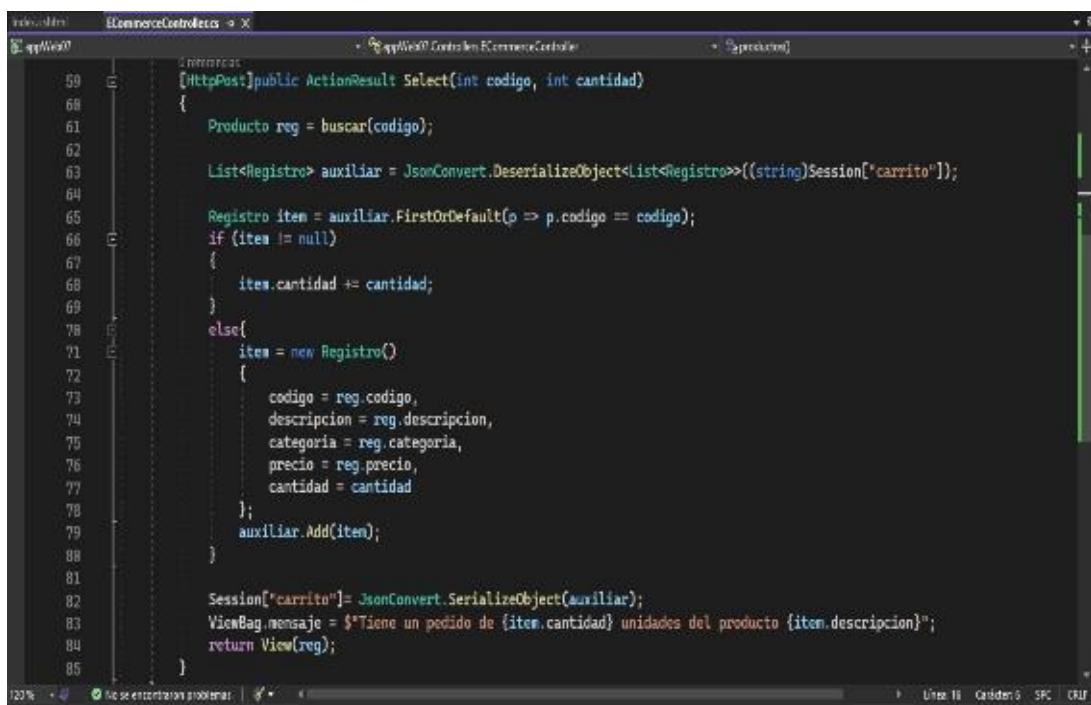
```

Index.cshtml      ECommerceController.cs  p X
appWeb07          appWeb07.Controllers.ECommerceController           S productos()
12   0 referencias
13
14   2 referencias
15     IEnumarable<Producto> productos()...
16
17   0 referencias
18     public ActionResult Index()...
19
20   1 referencia
21     Producto buscar(int id)...
22
23   0 referencias
24     public ActionResult Select(int ? id = null)
25   {
26         if (id == null) return RedirectToAction("Index");
27
28         return View(buscar(id.Value));
29     }
30
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
50
51
52

```

Nota. Elaboración propia.

Defina el ActionResult Select, método POST, donde registramos el producto seleccionado en el Session “carrito”, para ello evaluamos si el producto ya se encuentra en el Session donde acumulado la cantidad, sino lo agregamos al Session.

Figura 316*Desarrollo Práctico*


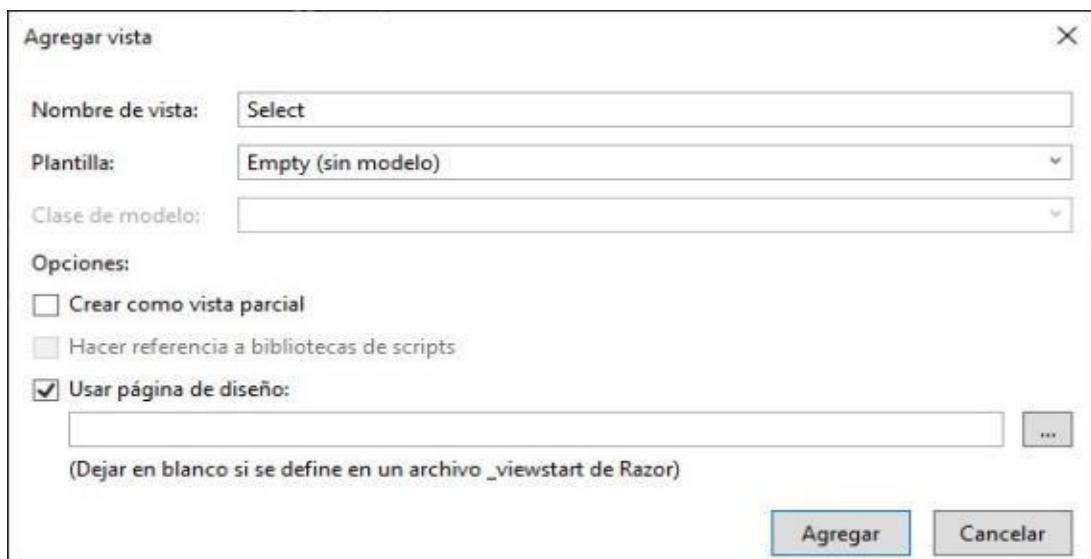
```

59     [HttpPost]
60     public ActionResult Select(int codigo, int cantidad)
61     {
62         Producto reg = buscar(codigo);
63
64         List<Registro> auxiliar = JsonConvert.DeserializeObject<List<Registro>>((string)Session["carrito"]);
65
66         Registro item = auxiliar.FirstOrDefault(p => p.codigo == codigo);
67         if (item != null)
68         {
69             item.cantidad += cantidad;
70         }
71         else{
72             item = new Registro()
73             {
74                 codigo = reg.codigo,
75                 descripcion = reg.descripcion,
76                 categoria = reg.categoría,
77                 precio = reg.precio,
78                 cantidad = cantidad
79             };
80             auxiliar.Add(item);
81         }
82
83         Session["carrito"] = JsonConvert.SerializeObject(auxiliar);
84         ViewBag.mensaje = $"Tiene un pedido de {item.cantidad} unidades del producto {item.descripcion}";
85         return View(reg);
86     }

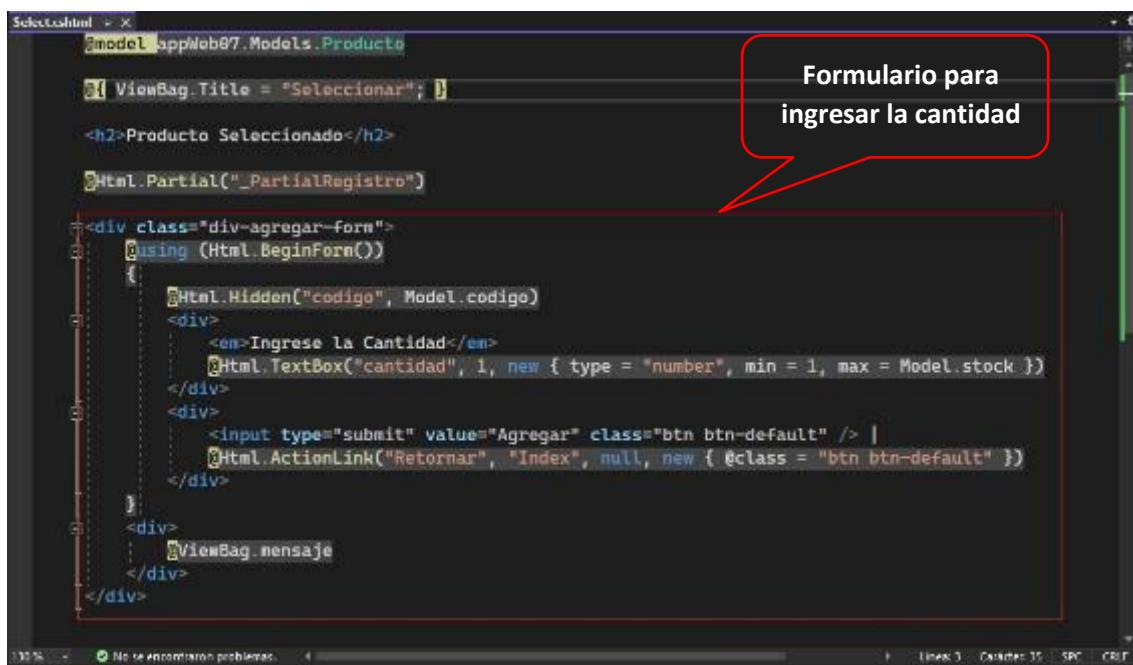
```

Nota. Elaboración propia.

A continuación, agregar la Vista Select cuya plantilla es Empty (sin modelo), tal como se muestra.

Figura 317*Desarrollo Práctico**Nota.* Elaboración propia.

Defina la vista Select, agregar el Helper partial para llamar a “_PartialRegistro”, defina un formulario para pasar el código del producto y la cantidad.

Figura 318*Desarrollo Práctico*


```

@model appWeb07.Models.Producto

@{ ViewBag.Title = "Seleccionar"; }



## Producto Seleccionado



@Html.Partial("_PartialRegistro")

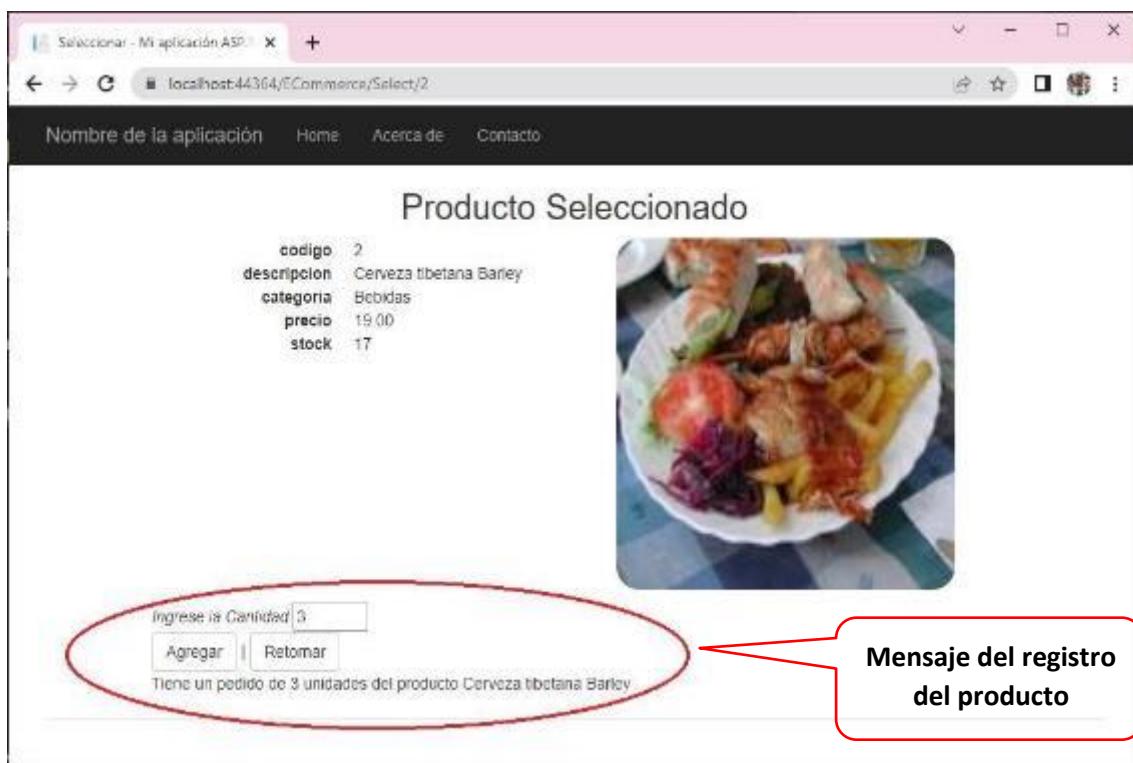


@using (Html.BeginForm())
    {
        @Html.Hidden("codigo", Model.codigo)
        <div>
            <em>Ingrese la Cantidad</em>
            @Html.TextBox("cantidad", 1, new { type = "number", min = 1, max = Model.stock })
        </div>
        <div>
            <input type="submit" value="Agregar" class="btn btn-default" /> |
            @Html.ActionLink("Retornar", "Index", null, new { @class = "btn btn-default" })
        </div>
    }
    <div>
        @ViewBag.mensaje
    </div>


```

Nota. Elaboración propia.

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

Figura 319*Desarrollo Práctico**Nota.* Elaboración propia.

Trabajando con el Session “carrito”

Para el desarrollo de este proceso, creamos una vista parcial llamada _PartialCarrito la cual lista los registros almacenados en el Session y al finalizar totalizar la columna Monto.

Figura 320

Desarrollo Práctico

```

<table class="table">
    <tr>
        <td>@Html.DisplayFor(modelItem => item.codigo)</td>
        <td>@Html.DisplayFor(modelItem => item.descripcion)</td>
        <td>@Html.DisplayFor(modelItem => item.categoría)</td>
        <td>@Html.DisplayFor(modelItem => item.precio)</td>
        <td>@Html.DisplayFor(modelItem => item.cantidad)</td>
        <td>@Html.DisplayFor(modelItem => item.monto)</td>
        <td>@Html.ActionLink("Delete", "Delete", new { id = item.codigo })</td>
    </tr>
    <tr>
        <td colspan="5" style="text-align:right">Monto del Pedido</td>
        <td>@Model.Sum(x => x.monto) </td>
    </tr>
</table>

```

Nota. Elaboración propia.

En el controlador defina el ActionResult Carrito(); el cual retorna los registros de Session[“carrito”] a la Vista, tal como se muestra.

Figura 321

Desarrollo Práctico

```

public ActionResult Carrito()
{
    //no se apertura el carrito
    if (Session["carrito"] == null)
        return RedirectToAction("Index");

    //si se apertura, pero no hay registros
    List<Registro> auxiliar =
        JsonConvert.DeserializeObject<List<Registro>>((string)Session["carrito"]);

    if (auxiliar.Count == 0)
        return RedirectToAction("Index");

    //si tienes registros
    return View(auxiliar);
}

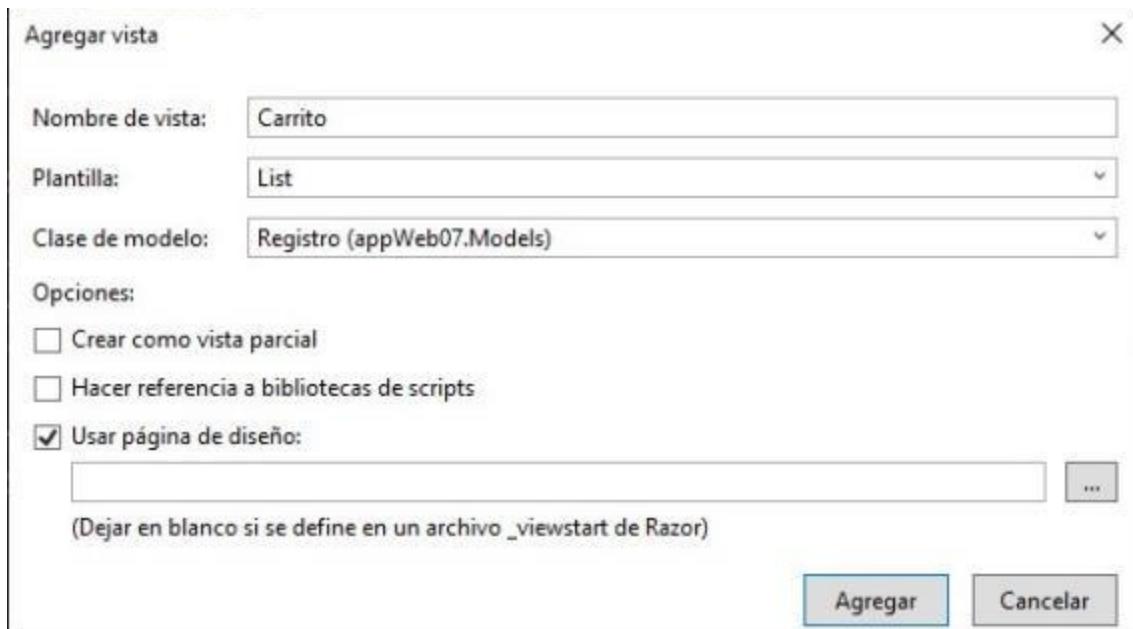
```

Nota. Elaboración propia.

Defina la vista Carrito de plantilla List y clase de modelo **Registro**, tal como se muestra:

Figura 322

Desarrollo Práctico



Nota. Elaboración propia.

A continuación, defina la vista Carrito donde agregamos dos ActionLink: “**Seguir Comprando**” y “**Registrar Pedido**”, tal como se muestra. Agregar el helper Parcial para listar los productos seleccionados en el Session.

Figura 323

Desarrollo Práctico

```

Carrito.cshtml < X \CommerceController.cs
model Ienumerable<appWeb07.Models.Registro>

<h2>Carrito de Compras</h2>

<p>
    @Html.ActionLink("Seguir Comprando", "Index", null, new { @class = "btn btn-success" })
    @Html.ActionLink("Registrar Pedido", "Pedido", null, new { @class = "btn btn-success" })
</p>

<div>
    @Html.Partial("_PartialCarrito")
</div>

```

Listar los pedidos a través de la Vista Parcial _PartialCarrito

Nota. Elaboración propia.

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

Figura 324

Desarrollo Práctico

codigo	descripcion	categoria	precio	cantidad	monto	Delete
1	Te Dhararamsala	Bebidas	18.00	1	18.00	Delete
2	Cerveza tibetana Barley	Bebidas	19.00	1	19.00	Delete
3	Sirope de regaliz	Condimentos	10.00	3	30.00	Delete

Totaliza la columna monto

Monto del Pedido 67.0

Nota. Elaboración propia.

Defina el ActionResult Delete para eliminar el registro del producto seleccionado en el carrito, una vez eliminado direccionar al Action Carrito.

Figura 325

Desarrollo Práctico

```

Carrito.cshtml    ECommerceController.cs  Index.cshtml
appWeb07          ECommerceController.cs
  86:     public ActionResult Carrito()
  87:     {
  88:         //recupero el contenido del Session
  89:         List<Registro> auxiliar =
  90:             JsonConvert.DeserializeObject<List<Registro>>((string)Session["carrito"]);
  91:
  92:         //buscar el registro por codigo y eliminar
  93:         Registro item = auxiliar.FirstOrDefault(s => s.codigo == id);
  94:         auxiliar.Remove(item);
  95:
  96:         //volver a almacenar auxiliar en Session carrito
  97:         Session["carrito"] = JsonConvert.SerializeObject(auxiliar);
  98:
  99:         return RedirectToAction("Carrito");
 100:    }
 101:
 102: }

```

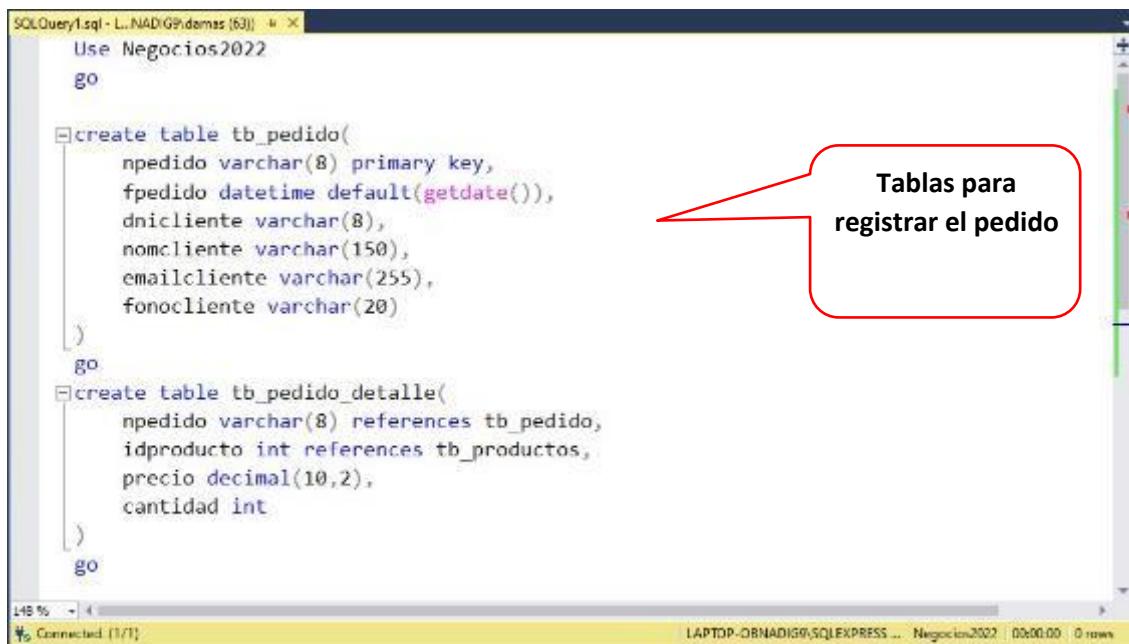
Nota. Elaboración propia.

Implementando el proceso para Registrar Pedido a la Base de Datos

En la base de datos creamos la tabla tb_pedido y tb_pedido_detalle para registrar los productos seleccionados desde la aplicación Ecommerce, tal como se muestra.

Figura 326

Desarrollo Práctico



```

SQLQuery1.sql - L_NADIG9\damas (63) 148 % 4
Use Negocios2022
go

create table tb_pedido(
    npedido varchar(8) primary key,
    fpedido datetime default(getdate()),
    dnicliente varchar(8),
    nomcliente varchar(150),
    emailcliente varchar(255),
    fonocliente varchar(20)
)
go
create table tb_pedido_detalle(
    npedido varchar(8) references tb_pedido,
    idproducto int references tb_productos,
    precio decimal(10,2),
    cantidad int
)
go

```

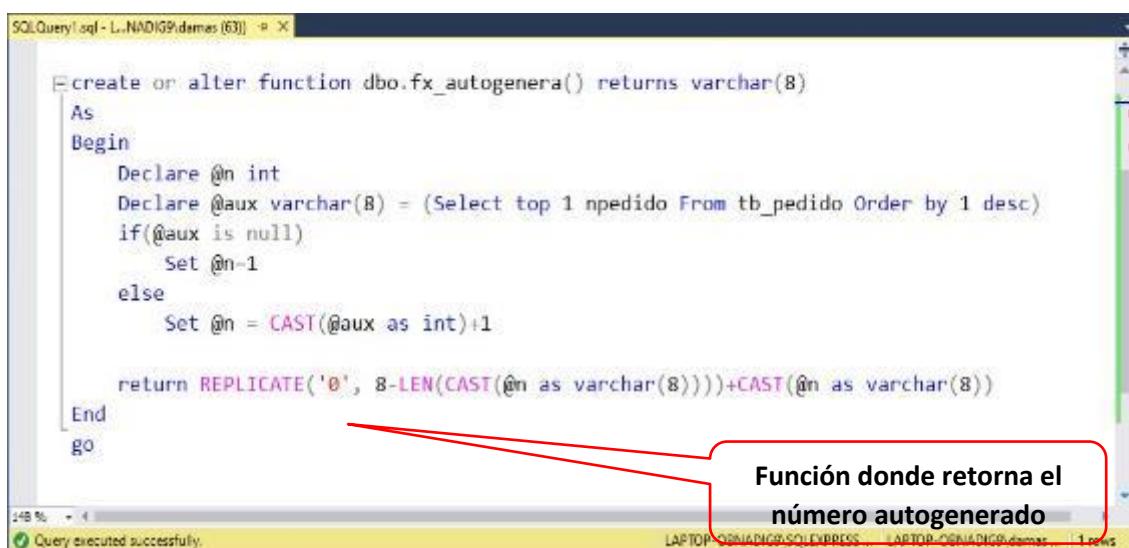
Connected (1/1) LAPTOP-ORNADIGO\SQLEXPRESS ... Negocios2022 00:00:00 | 0 rows

Nota. Elaboración propia.

Defina la función fx_autogenera() donde retorna el número del pedido definido por un formato para la salida de su valor.

Figura 327

Desarrollo Práctico



```

SQLQuery1.sql - L_NADIG9\damas (63) 148 % 4
create or alter function dbo.fx_autogenera() returns varchar(8)
As
Begin
    Declare @n int
    Declare @aux varchar(8) = (Select top 1 npedido From tb_pedido Order by 1 desc)
    if(@aux is null)
        Set @n=1
    else
        Set @n = CAST(@aux as int)+1

    return REPLICATE('0', 8-LEN(CAST(@n as varchar(8))))+CAST(@n as varchar(8))
End
go

```

Query executed successfully. LAPTOP-ORNADIGO\SQLEXPRESS ... Negocios2022 1 rows

Nota. Elaboración propia.

Defina los procedimientos almacenados para agregar registros a la tabla tb_pedido, y tb_pedido_detalle.

Figura 328

Desarrollo Práctico

```

create or alter proc usp_pedido_add
@npedido varchar(8) output,
@dncliente varchar(8),
@nomcliente varchar(150),
@emailcliente varchar(255),
@fonocliente varchar(20)
AS
Begin
    Set @npedido=dbo.fx_autogenera()
    Insert tb_pedido(npedido,dncliente,nomcliente,emailcliente,fonocliente)
    Values(@npedido,@dncliente,@nomcliente,@emailcliente,@fonocliente)
End
go

create or alter proc usp_pedido_detalle_add
@npedido varchar(8) output,
@idproducto int,
@pre decimal(10,2),
@cantidad int
AS
    Insert tb_pedido_detalle Values(@npedido,@idproducto,@pre,@cantidad)
go

```

Query executed successfully.

Nota. Elaboración propia.

Trabajando con el ActionResult Pedido

En la carpeta Models, agrega una clase llamada Pedido, defina su estructura, tal como se muestra.

Figura 329

Desarrollo Práctico

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
namespace appWeb07.Models
{
    public class Pedido
    {
        [Display(Name = "DNI Cliente")]
        public string dncliente { get; set; }
        [Display(Name = "Nombre Cliente")]
        public string nomcliente { get; set; }
        [Display(Name = "Email Cliente")]
        public string emailcliente { get; set; }
        [Display(Name = "Telefono")]
        public string fonocliente { get; set; }
    }
}

```

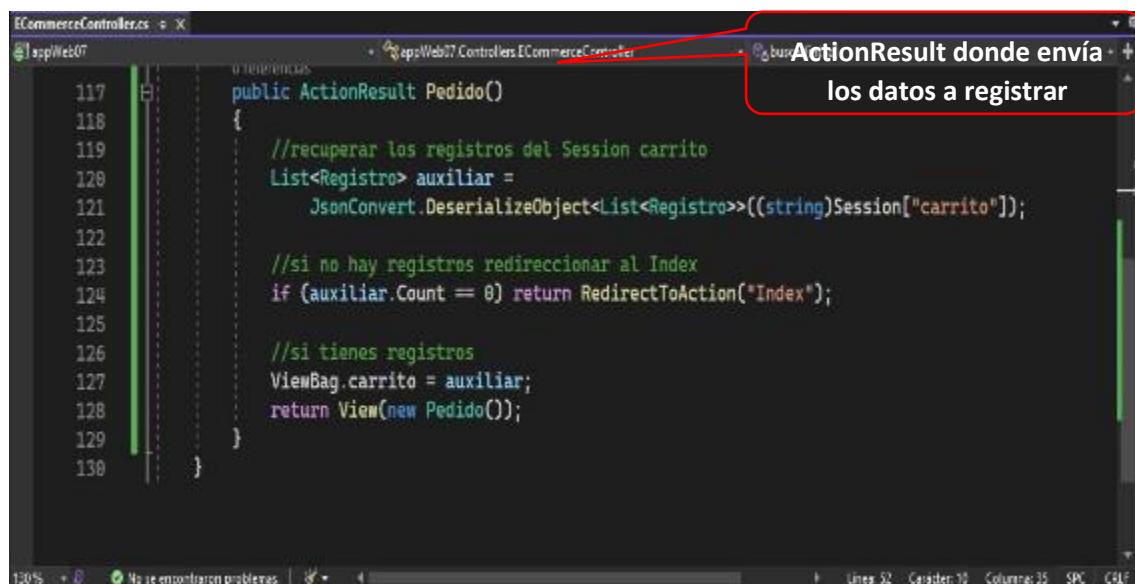
Definir la estructura de la clase Pedido

Nota. Elaboración propia.

Defina el ActionResult Pedido, el cual envía los datos registrados en el Session carrito y la instancia de un nuevo Pedido para el registro de los datos, tal como se muestra.

Figura 330

Desarrollo Práctico



```

ECommerceController.cs  X
aspWeb07  ECommerceController.cs  ActionResult donde envía
+  los datos a registrar
117     public ActionResult Pedido()
118     {
119         //recuperar los registros del Session carrito
120         List<Registro> auxiliar =
121             JsonConvert.DeserializeObject<List<Registro>>((string)Session["carrito"]);
122
123         //si no hay registros redireccionar al Index
124         if (auxiliar.Count == 0) return RedirectToAction("Index");
125
126         //si tienes registros
127         ViewBag.carrito = auxiliar;
128         return View(new Pedido());
129     }
130

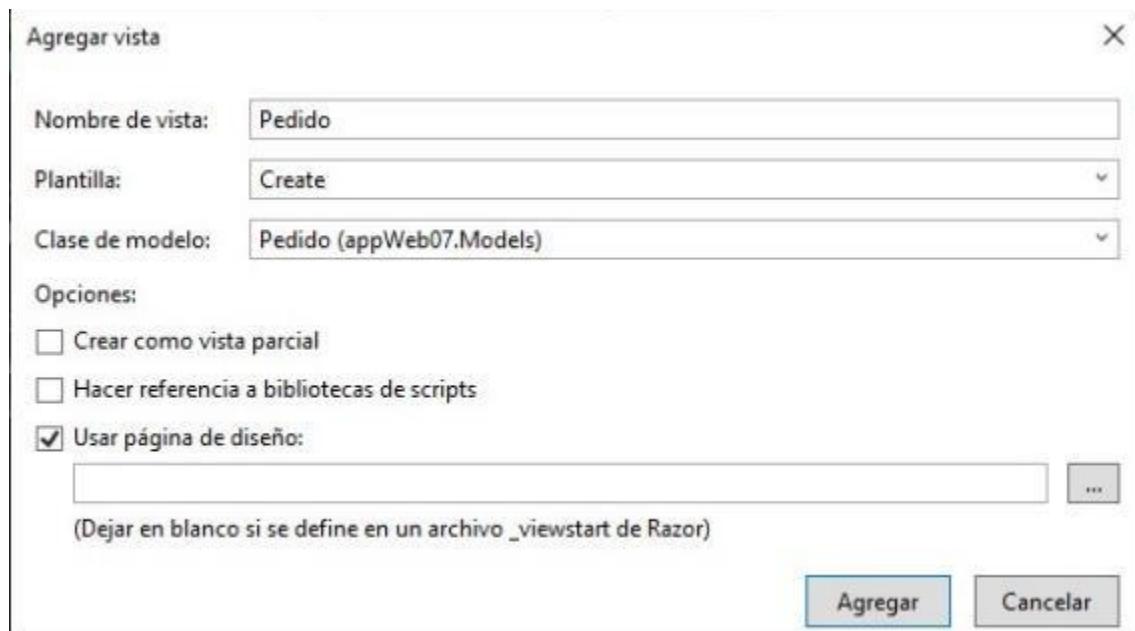
```

Nota. Elaboración propia.

Crear la vista del ActionResult Pedido, selecciona la plantilla Create de clase de Modelo Pedido.

Figura 331

Desarrollo Práctico



Nota. Elaboración propia.

Vista del ActionResult Pedido

En la vista Pedido, dividir la página en dos partes la primera visualizamos los datos del pedido y la segunda visualizamos los datos del Session carrito, tal como se muestra.

Figura 332

Desarrollo Práctico

```

Pedido.cshtml  |  ECommerceController.cs
using appWeb07.Models
@model Pedido

<Link href("~/Content/style-commerce.css" rel="stylesheet" />

@{ ViewBag.Title = "Pedido"; }

<h2>Registro de Pedidos</h2>

<div class="div-agregar">
<div>
@using (Html.BeginForm())
{
    <div class="form-horizontal"></div>
}
</div>
<div>
    @Html.Partial("_PartialCarrito", (List<Registro>)ViewBag.carrito)
</div>
</div>

```

Nota. Elaboración propia.

Al ejecutar la vista de Pedido, visualizamos los datos a registrar y en la derecha visualizamos el resumen de los pedidos seleccionados del Session carrito.

Figura 333

Desarrollo Práctico

codigo	descripcion	categoria	precio	cantidad	monto	Delete
1	Té Charmsala	Bebidas	18.00	1	18.00	Delete
2	Cerveza Blanca Bailey	Bebidas	19.00	3	57.00	Delete

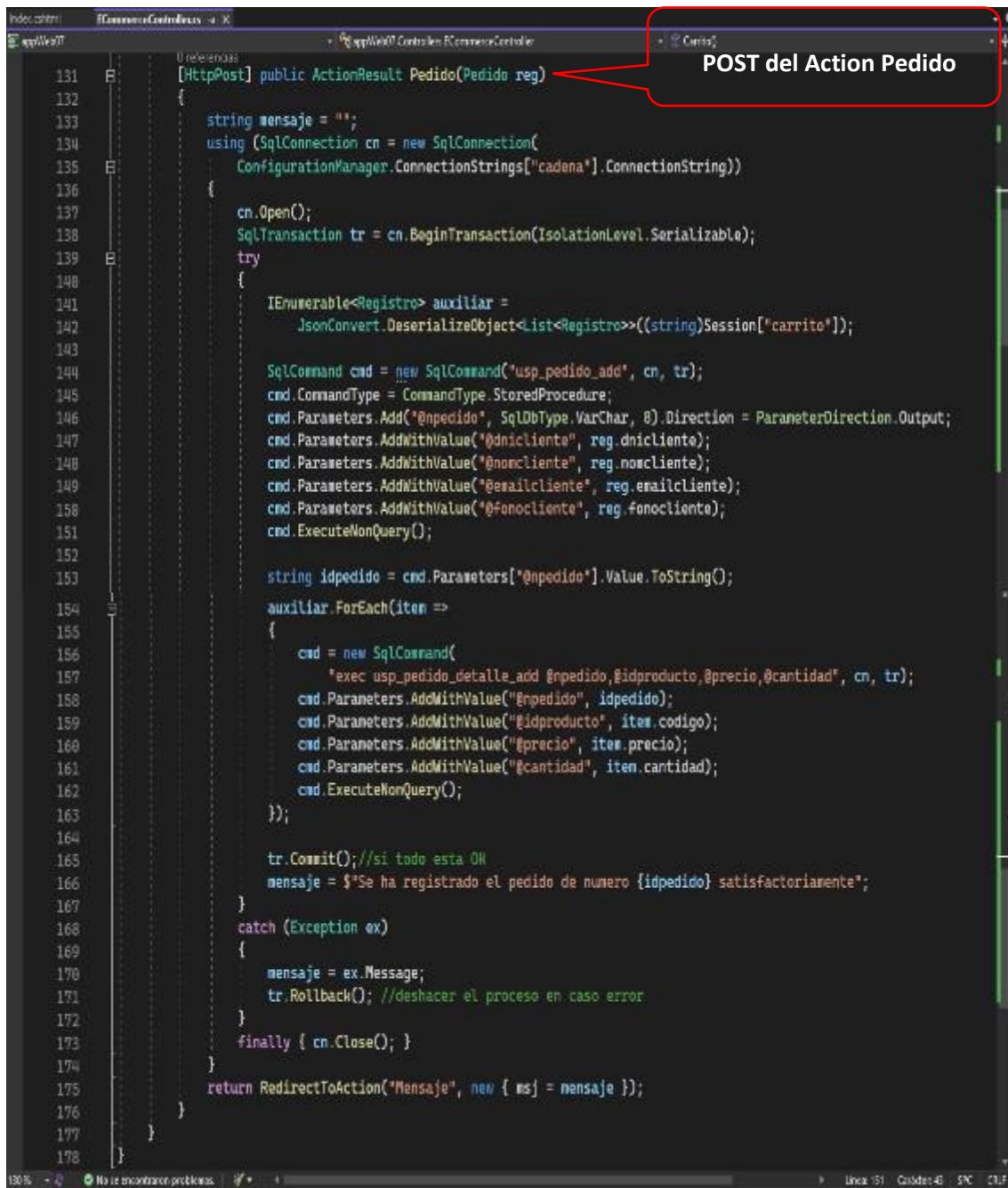
Monto del Pedido: 75.0

Nota. Elaboración propia.

A continuación, defina el método POST del ActionResult Pedido el cual ejecuta los procedimientos almacenados para agregar un registro a la tabla tb_pedido y los registros del pedido en tb_pedido_detalle ejecutando procedimientos almacenados definidos en la base de datos.

Figura 334

Desarrollo Práctico



```

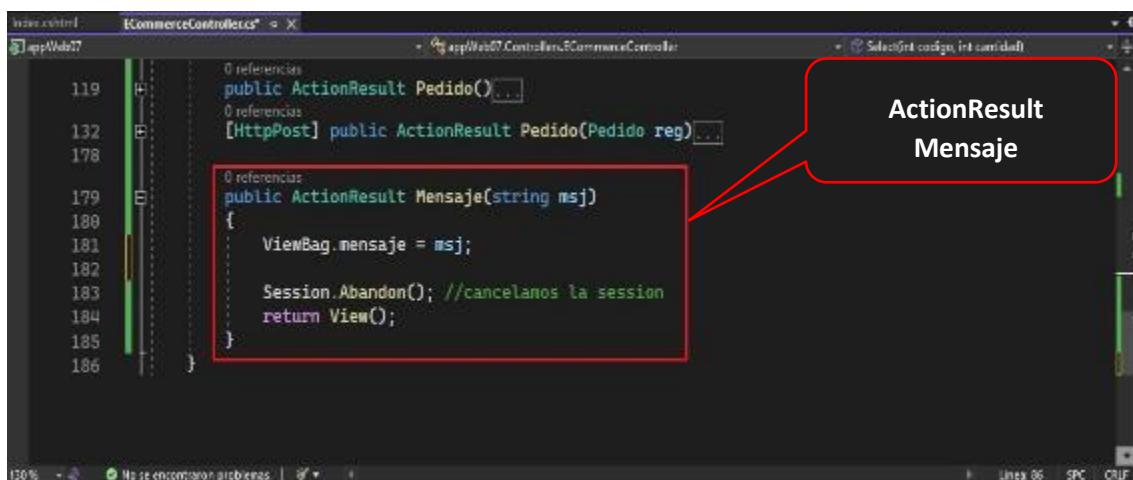
Index.cshtml      ECommerceController.cs
appWebSite          ECommerceController
  131  B  [HttpPost] public ActionResult Pedido(Pedido reg)
  132  {
  133  string mensaje = "";
  134  using (SqlConnection cn = new SqlConnection(
  135  ConfigurationManager.ConnectionStrings["cadena"].ConnectionString))
  136  {
  137  cn.Open();
  138  SqlTransaction tr = cn.BeginTransaction(IsolationLevel.Serializable);
  139  B  try
  140  {
  141  IEnumable<Registro> auxiliar =
  142  JsonConvert.DeserializeObject<List<Registro>>((string)Session["carrito"]);
  143
  144  SqlCommand cmd = new SqlCommand("usp_pedido_add", cn, tr);
  145  cmd.CommandType = CommandType.StoredProcedure;
  146  cmd.Parameters.AddWithValue("@npedido", SqlDbType.VarChar, 8).Direction = ParameterDirection.Output;
  147  cmd.Parameters.AddWithValue("@dncliente", reg.dncliente);
  148  cmd.Parameters.AddWithValue("@nomcliente", reg.nomcliente);
  149  cmd.Parameters.AddWithValue("@emailcliente", reg.emailcliente);
  150  cmd.Parameters.AddWithValue("@fonocliente", reg.fonocliente);
  151  cmd.ExecuteNonQuery();
  152
  153  string idpedido = cmd.Parameters["@npedido"].Value.ToString();
  154  auxiliar.ForEach(item =>
  155  {
  156  cmd = new SqlCommand(
  157  "exec usp_pedido_detalle_add @npedido,@idproducto,@precio,@cantidad", cn, tr);
  158  cmd.Parameters.AddWithValue("@npedido", idpedido);
  159  cmd.Parameters.AddWithValue("@idproducto", item.codigo);
  160  cmd.Parameters.AddWithValue("@precio", item.precio);
  161  cmd.Parameters.AddWithValue("@cantidad", item.cantidad);
  162  cmd.ExecuteNonQuery();
  163  });
  164
  165  tr.Commit(); //si todo esta OK
  166  mensaje = $"Se ha registrado el pedido de numero {idpedido} satisfactoriamente";
  167  }
  168  catch (Exception ex)
  169  {
  170  mensaje = ex.Message;
  171  tr.Rollback(); //deshacer el proceso en caso error
  172  }
  173  finally { cn.Close(); }
  174  }
  175  }
  176  }
  177  }
  178  }
}

```

Nota. Elaboración propia.

Trabajando con el ActionResult Mensaje

Para finalizar agregar el ActionResult llamado Mensaje donde salimos del Session “carrito” y enviamos a la vista el mensaje del proceso Pedido.

Figura 335*Desarrollo Práctico*


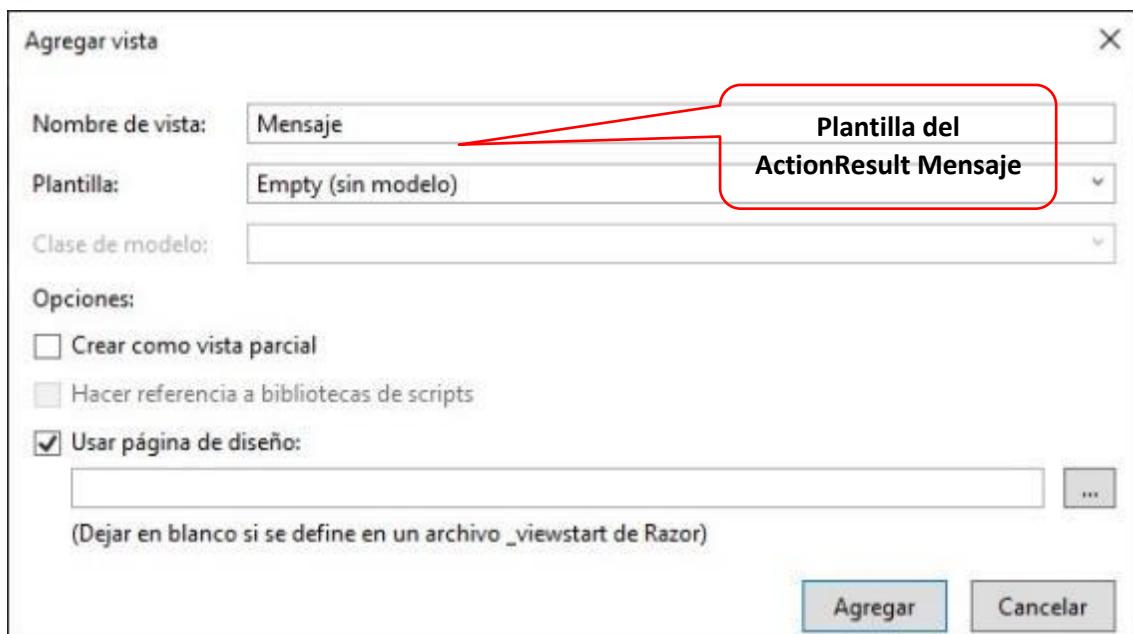
```

119     public ActionResult Pedido()
132     [HttpPost] public ActionResult Pedido(Pedido reg)
178
179     public ActionResult Mensaje(string msj)
180     {
181         ViewBag.mensaje = msj;
182
183         Session.Abandon(); //cancelamos la session
184         return View();
185     }
186

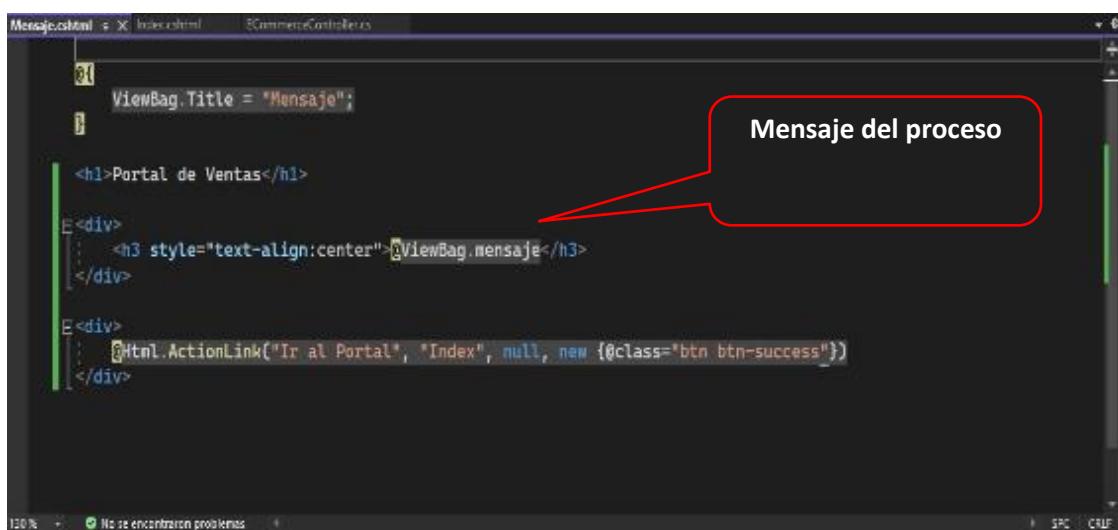
```

Nota. Elaboración propia.

Agregar la Vista del ActionResult Mensaje cuya plantilla es Empty (sin modelo), tal como se muestra.

Figura 336*Desarrollo Práctico**Nota.* Elaboración propia.

Diseña la vista donde visualizamos el mensaje del Proceso Pedido.

Figura 337*Desarrollo Práctico*


```

@{
    ViewBag.Title = "Mensaje";
}

<h1>Portal de Ventas</h1>

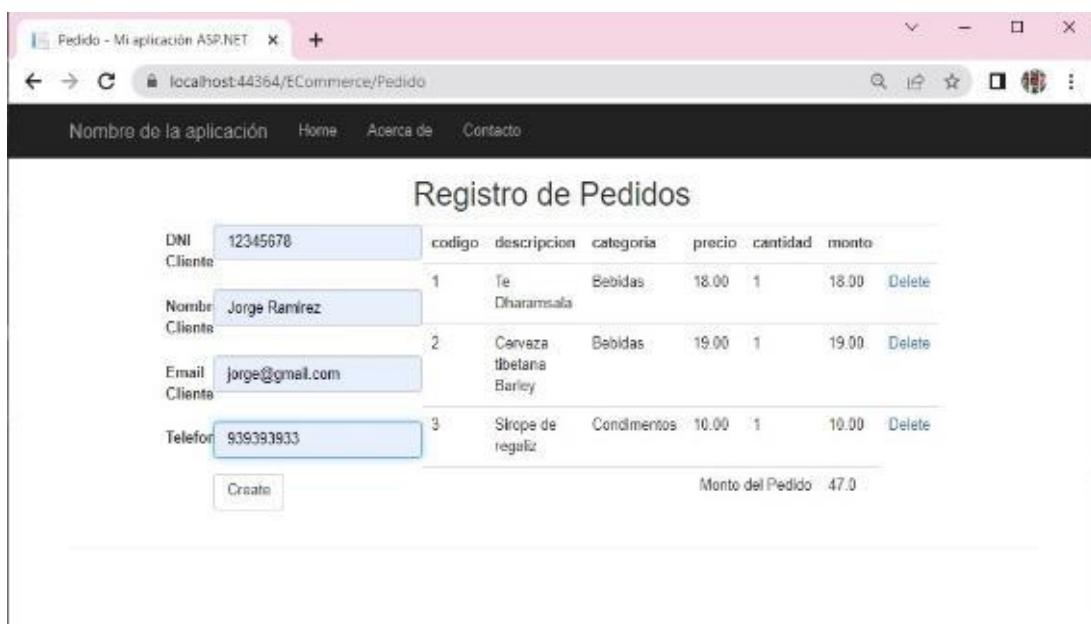
<div>
    <h3 style="text-align:center">@ViewBag.mensaje</h3>
</div>

<div>
    <a href="#" class="btn btn-success">Ir al Portal</a>
</div>

```

Nota. Elaboración propia.

Ejecutamos la vista Index, selecciona los datos al momento direccionar a la Vista Pedido registramos los datos.

Figura 338*Desarrollo Práctico*


DNI	Codigo	Descripción	Categoría	Precio	Cantidad	Monto	
12345678	1	Te Dharamsala	Bebidas	18.00	1	18.00	Delete
Jorge Ramirez	2	Cerveza tibetana	Bebidas	19.00	1	19.00	Delete
jorge@gmail.com	3	Strope de regaliz	Condimentos	10.00	1	10.00	Delete

[Create](#)

Monto del Pedido: 47.0

Nota. Elaboración propia.

Al presionar el botón Create visualizamos en la Vista Mensaje la confirmación del pedido y visualizamos el número de pedido.

Figura 339*Desarrollo Práctico*

Nota. Elaboración propia.

Resumen

1. 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".
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta unidad:

- <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>



TRABAJANDO CON ANGULAR JS EN ASP.NET

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno conoce las técnicas para el diseño de un SPA, para realizar operaciones de actualización de datos en ASP.NET MVC.

TEMARIO

4.1 Tema 8 : AngularJS

- 4.1.1 : Introducción al Angular
- 4.1.2 : Integrando Angular en ASP.NET MVC

ACTIVIDADES PROPUESTAS

- Los alumnos implementan un proyecto web para consultar datos en un SPA utilizando AngularJS.
- Los alumnos desarrollan los laboratorios de esta semana.

4.1. ANGULARJS

4.1.1. Introducción a Angular

Desde su creación, Angular ha sido el framework preferido por la mayoría de los desarrolladores JavaScript. Este éxito ha provocado que los desarrolladores quieran usar el framework para más y más cosas. (Basalo, 2016)

De ser una plataforma para la creación de Web Apps, ha evolucionado como motor de una enorme cantidad de proyectos del ámbito empresarial y de ahí para aplicaciones en la Web Mobile Híbrida, llevando la tecnología al límite de sus posibilidades. (Basalo, 2016)

La nueva herramienta está pensada para dar cabida a todos los usos dados por los desarrolladores, llevar a JavaScript a un nuevo nivel comparable a lenguajes más tradicionales, siendo además capaz de resolver de una manera adecuada las necesidades y problemas de la programación del lado del cliente. Entre las soluciones aportadas en Angular o Angular 2. (Basalo, 2016)

TypeScript / JavaScript:

Como base hemos puesto a JavaScript, ya que es el inicio de los problemas de escalabilidad del código. Ayuda poco a detectar errores y además produce con facilidad situaciones poco deseables.

La sugerencia de usar TypeScript para desarrollar en Angular es casi una imposición porque la documentación y los generadores de código están pensados en TypeScript. Se supone que en futuro también estarán disponibles para JavaScript, pero de momento no es así. De todos modos, para la tranquilidad de muchos, TypeScript no agrega más necesidad de procesamiento a las aplicaciones con Angular 2, ya que este lenguaje solamente lo utilizas en la etapa de desarrollo y todo el código que se ejecuta en el navegador es al final JavaScript, ya que existe una transpilación previa. (Basalo, 2016)

Lazy SPA:

Ahora el inyector de dependencias de Angular no necesita que estén en memoria todas las clases o código de todos los elementos que conforman una aplicación. Ahora con Lazy SPA el framework puede funcionar sin conocer todo el código de la aplicación, ofreciendo la posibilidad de cargar más adelante aquellas piezas que no necesitan todavía.

Renderizado Universal:

Angular nació para hacer web y renderizar en HTML en el navegador, pero ahora el renderizado universal nos permite que no solo se pueda renderizar una vista a HTML. Gracias a esto, alguien podría programar una aplicación y que el renderizado se haga, por ejemplo, en otro lenguaje nativo para un dispositivo dado. (Basalo, 2016)

Otra cosa que permite el renderizado universal es que se use el motor de renderizado de Angular del lado del servidor. Es una de las novedades más interesantes, ya que ahora podrás usar el framework para renderizar vistas del lado del servidor, permitiendo un mejor potencial de posicionamiento en buscadores de los contenidos de una aplicación. (Basalo, 2016)

Data Binding Flow:

En Angular 2, el flujo de datos ahora está mucho más controlado y el desarrollador puede direccionarlo fácilmente, permitiendo optimizar las aplicaciones. El resultado puede llegar a ser hasta 5 veces más rápidas.

Componentes:

La arquitectura de una aplicación Angular ahora se realiza mediante componentes. En este caso no se trata de una novedad de la versión 2, ya que en la versión de Angular 1.5 ya se introdujo el desarrollo basado en componentes.

Los componentes son estancos, no se comunican con el padre a no ser que se haga explícitamente por medio de los mecanismos disponibles, etc. Todo esto genera aplicaciones más mantenibles, donde se encapsula mejor la funcionalidad y cuyo funcionamiento es más previsible. Ahora se evita el acceso universal a cualquier cosa desde cualquier parte del código, vía herencia o cosas como el "Root Scope", que permitía en versiones tempranas de Angular modificar cualquier cosa de la aplicación desde cualquier sitio. (Basalo, 2016)

Características de Angular

Las principales características de este FrameWork para conocer mejor su funcionalidad y la manera en que puedes utilizarlo en el desarrollo de algún proyecto web próximo, lo mencionamos a continuación.

1. Uso de DOM regular:

Angular hace uso del modelo DOM, lo que permite una mejor organización conforme avanza el desarrollo web.

2. Enlace de datos o data binding:

El enlace de datos o data binding es un proceso donde los usuarios pueden manipular elementos de una página web a través de un navegador. Entre sus principales ventajas es que no requiere secuencias de comandos ni programaciones complejas, además de que emplea HTML dinámico. También permite una mejora en la visualización de una página web, sobre todo cuando contiene una gran cantidad de datos.

3. Compatibilidad móvil y de escritorio:

Angular funciona tanto para el desarrollo de aplicaciones móviles como de escritorio. Esto también significa que puede ejecutarse en la mayoría de los navegadores web.

4. Velocidad y rendimiento:

Cuenta con código de generación que permite convertir tus plantillas en códigos altamente optimizados. Esto te ofrece todos los beneficios del código escrito a mano con la productividad de un marco.

5. Productividad:

Permite la creación rápida de vistas de interfaz de usuarios con una sintaxis de plantilla muy sencilla y eficaz. Además, con sus herramientas de líneas de comandos puedes comenzar a construir en menor tiempo y agregar componentes, pruebas e implementaciones al instante.

6. Enlace bidireccional de datos:

Una de las principales ventajas es que enlaza JavaScript y HTML, donde el código de ambos está sincronizado, lo que ahorra mucho tiempo para los desarrolladores web.

7. Directivas:

Los archivos HTML se ven ampliados gracias a directivas habilitadas por los desarrolladores cuando agregan el prefijo ng- a los atributos HTML. Existen diferentes tipos de directivas muy útiles que puedes usar para diferentes acciones como vincular el contenido de un elemento HTML a los datos de aplicación, o especificar que el contenido de un texto debe reemplazarse con una plantilla.

8. Pruebas:

Hace uso del framework “Jasmine”, el cual proporciona diversas funcionalidades para escribir diferentes tipos de casos de prueba. De igual manera, el marco admite pruebas unitarias y de integración.

Ventajas de Angular:**1. Alta calidad de la aplicación:**

Si bien Angular es una plataforma compleja y difícil de aprender para algunos desarrolladores web principiantes, también representa una enorme ventaja, pues el éxito del producto está prácticamente asegurado.

2. Desarrollo multiplataforma:

Anteriormente, los desarrolladores front-end usaban la fórmula Ionic + Angular para el desarrollo multiplataforma, pero hoy en día es más común utilizar Angular + NativeScript. NativeScript te brinda acceso a las API nativas para desarrollar aplicaciones que pueden ejecutarse tanto en iOS como en Android.

3. Proceso de desarrollo web más rápido:

Permite crear aplicaciones de forma más rápida y eficiente, pues goza de ventajas técnicas como las siguientes:

- Documentación detallada. La documentación de Angular está cuidadosamente escrita y dotada de una gran variedad de ejemplos de código para tener mayor claridad en el proceso y permitir que los desarrolladores encuentren soluciones rápidas a cualquier problema conforme crean una aplicación.
- Interfaz de línea de comandos de Angular. La CLI (por sus siglas en inglés) facilita el trabajo de los desarrolladores al ofrecer un conjunto de herramientas de codificación. Además, esta línea de comandos puede ampliarse con bibliotecas de terceros para resolver problemas de software inusuales o muy complicados.
- Enlace de datos bidireccional. Como ya lo mencionamos, Angular cuenta con esta característica, lo que permite el ahorro de tiempo al automatizar algunos procesos de generación de código.
- Soporte de Google. Cuando definimos qué es Angular también destacamos que lo mantiene Google, por lo que puedes estar seguro de que recibirás soporte en tiempo y forma a cualquier duda o problema que presentes.

4. Aplicaciones web más ligeras:

Actualmente, se han hecho mejoras en los módulos de carga diferida y con el renderizador de Ivy, que permite hacer paquetes más pequeños para acelerar la aplicación.

5. Código legible y comprobable:

Podemos resaltar que Angular es un marco de trabajo consistente gracias a sus elementos estructurales como módulos, componentes, directivas, tuberías y servicios; así como la posibilidad de escribir aplicaciones con la arquitectura tradicional MVVM (Model-ViewViewModel) y MVC (Model-View-Controller). Ambas sirven para el mejoramiento de reutilización de código.

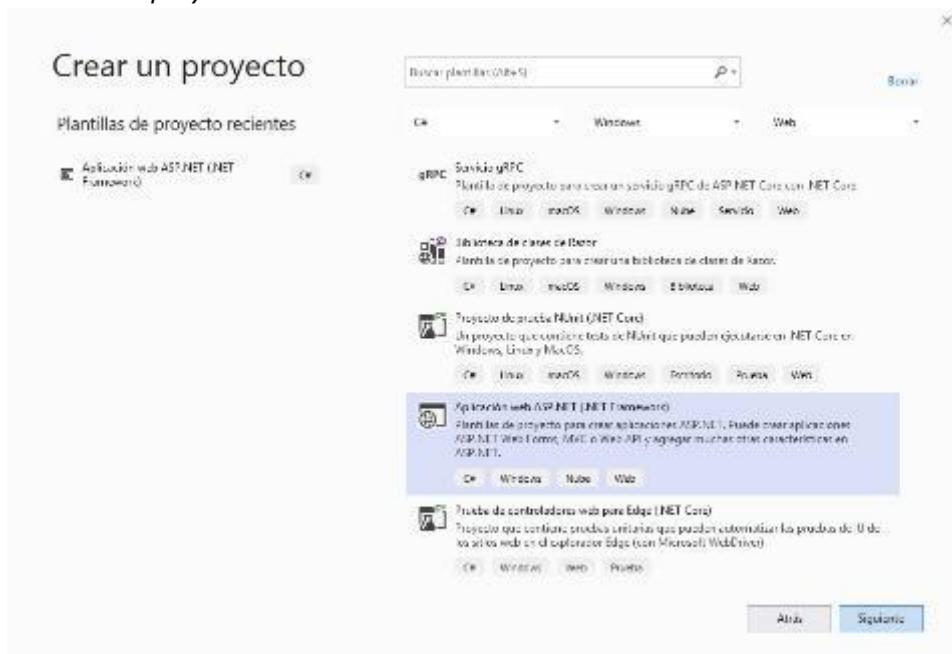
4.1.2. Integrando Angular en ASP.NET MVC

AngularJS es un framework de JavaScript que nos permite crear aplicaciones de “página única” o SPA. AngularJS también nos permite implementar el patrón de diseño MVVM.

Arrancamos el Visual Studio y creamos un nuevo proyecto de tipo Aplicación web ASP.NET .NET FrameWork con C#.

Figura 340

Creando un proyecto



Nota. Elaboración propia.

Al crear la aplicación, selecciona la opción MVC, tal como se muestra.

Figura 341
Creando un proyecto



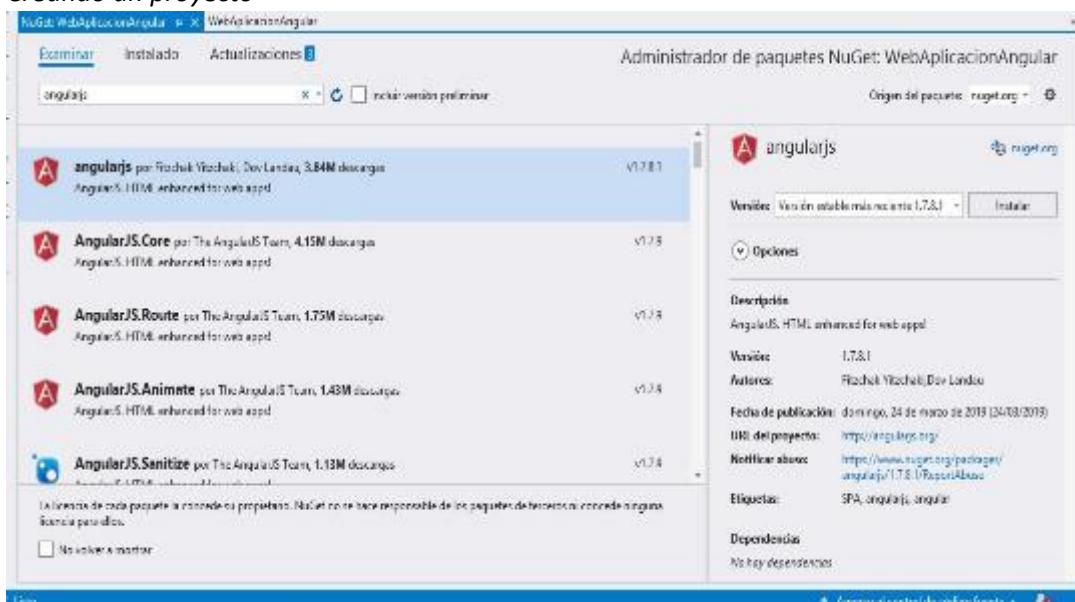
Nota. Elaboración propia.

Instalando AngularJS

Desde la opción de Proyecto, selecciona la opción Administrando paquetes NUGET.

En la opción EXAMINAR, escribe AngularJS y selecciona AngularJS, la versión más reciente, tal como se muestra

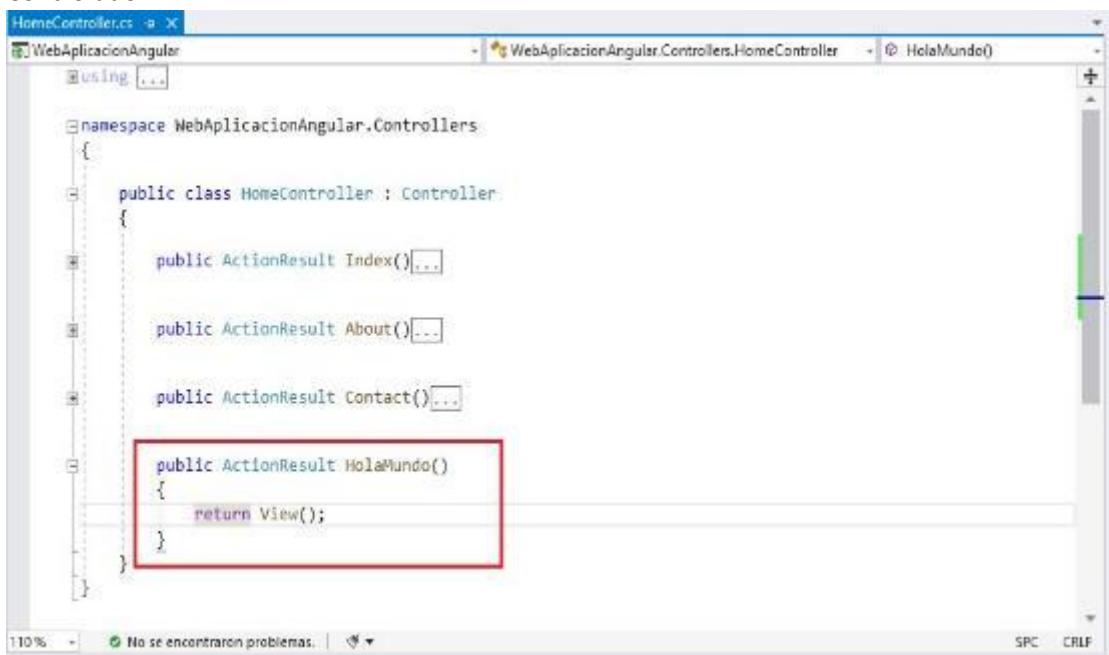
Figura 342
Creando un proyecto



Nota. Elaboración propia.

A continuación, agregamos un Action en el controlador Home, tal como se muestra.

Figura 343
Controlador



```

HomeController.cs  ↗ X
WebAplicacionAngular
  ↗ using ...
namespace WebAplicacionAngular.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()...
        public ActionResult About()...
        public ActionResult Contact()...
        public ActionResult HolaMundo()
        {
            return View();
        }
    }
}
  
```

110% No se encontraron problemas. SPC CRLF

Nota. Elaboración propia.

Agrega una Vista, la cual será vacía sin modelo, tal como se muestra.

Figura 344
Vista



Nota. Elaboración propia.

Aquí en el div, llama una directiva llamada "ng-controller", cuyo valor consiste en el nombre de la función JavaScript que se quiere aplicar a un Div, Span o cualquier elemento HTML específico.

Después, en el campo de texto utilizo la directiva "ng-model" que proporciona la unión entre la Vista y el Modelo. En ng-model, cuyo valor es "test", significa que el valor introducido en este campo se reemplazará con el valor predeterminado, establecido en la función JavaScript.

Hay que añadir la directiva "ng-app" en la etiqueta HTML. ng-app declara que esto es una página AngularJS. Si no agregas esta directiva en la etiqueta HTML, tu aplicación no funcionará correctamente porque no podrá detectar las demás directivas de AngularJS.

Figura 345

Vista

```
<html ng-app xmlns="http://www.w3.org/1999/xhtml">
<head>
    <script src="~/Scripts/angular.min.js"></script>
    <script>
        function HelloWorld($scope) {
            $scope.test = 'Mundo';
        }
    </script>
</head>
<body>
    <div ng-controller="HelloWorld">
        Hola {{test || "Mundo"}}!
        <br />
        Tu nombre: <input type="text" ng-model="test" />
    </div>
</body>
</html>
```

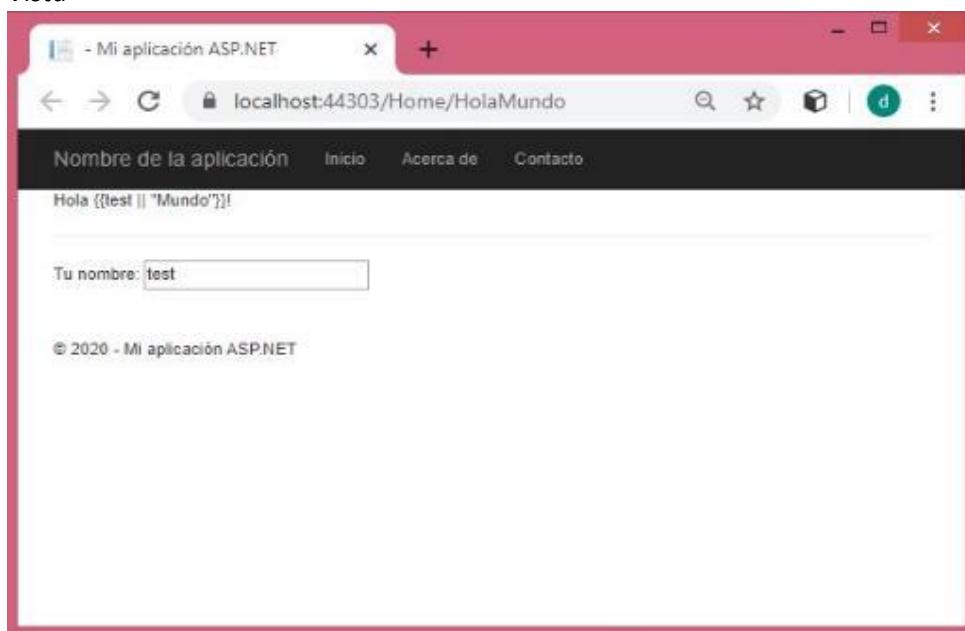
110% No se encontraron problemas. Línea: 12 Carácter: 34 SPC CR/LF

Nota. Elaboración propia.

Ahora nuestra aplicación de Hola Mundo está lista para ser ejecutada.

Figura 346

Vista

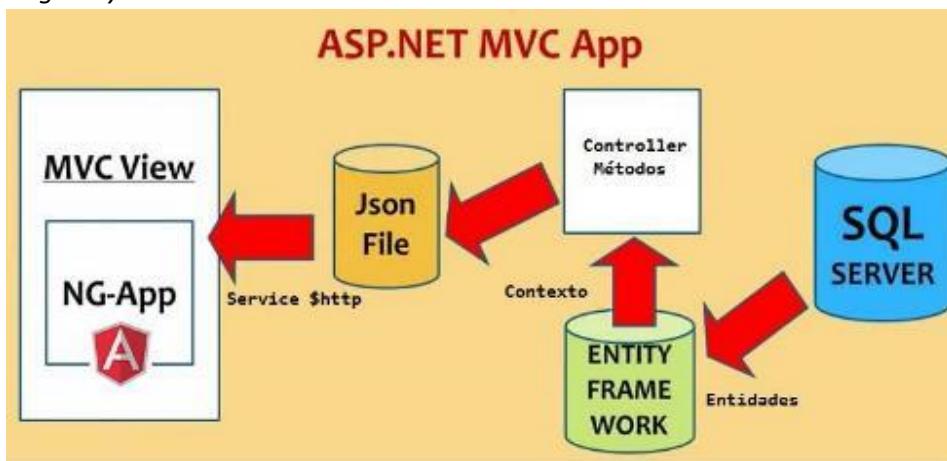


Nota. Elaboración propia.

Integrando con una base de datos y Angular

Figura 347

Angular y base de datos



Nota. Tomado de MVC, por Macoratti, 2016, Macoratti.net (http://www.macoratti.net/16/03/mvc_crudajs1.htm)

Una estrategia para trabajar con Angular en una aplicación ASP.NET MVC es definir un archivo js para la ejecución de sus métodos en las vistas.

A continuación, mostramos las siguientes directivas de AngularJS:

- **ng-controller:** define el controlador
- **ng-repeat** - repite la plantilla un cierto número de veces
- **ng-click:** agrega una función al evento de clic del controlador AngularJS
- **ng-model:** define un modelo de datos vinculado a nuestra vista
- **ng-disabled:** deshabilita / habilita el botón

LABORATORIO 8.1.:

Implementando “Bienvenido” en una aplicación ASP.NET MVC

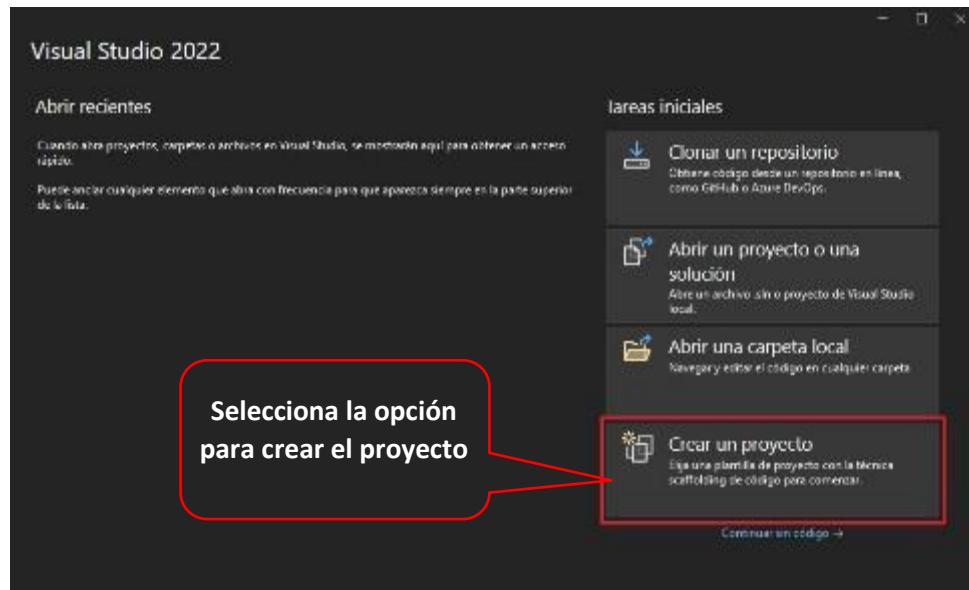
Implemente una aplicación en ASP.NET MVC donde imprima en una vista el mensaje “Bienvenido” utilizando Angular.

Creando un Proyecto ASP.NET MVC

Inicio del Proyecto

- Cargar la aplicación del Menú INICIO.
- Seleccione “Crear un proyecto”.

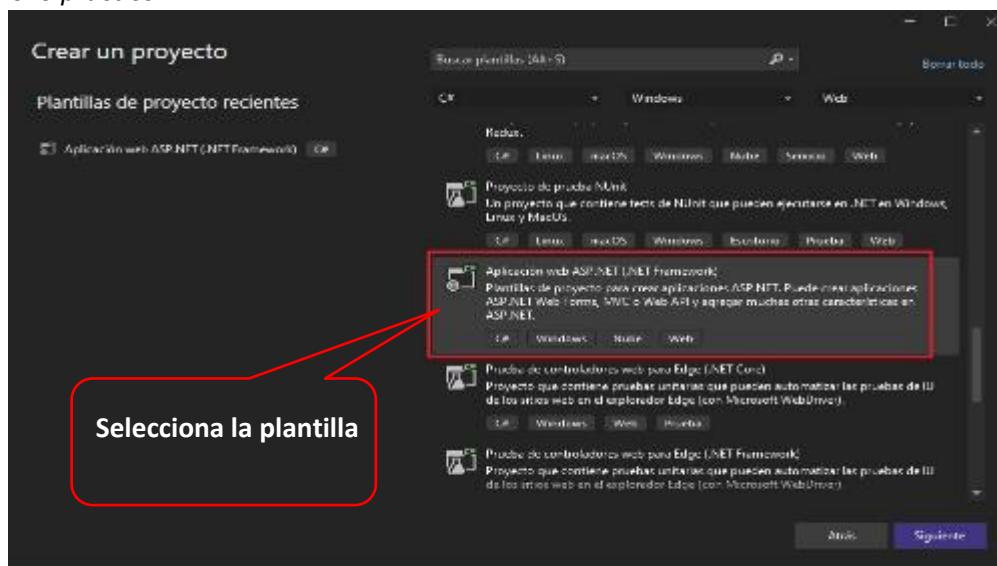
Figura 348
Desarrollo práctico



Nota. Elaboración propia.

Selecciona la plantilla de la aplicación web de ASP.NET MVC, presionar la opción Siguiente:

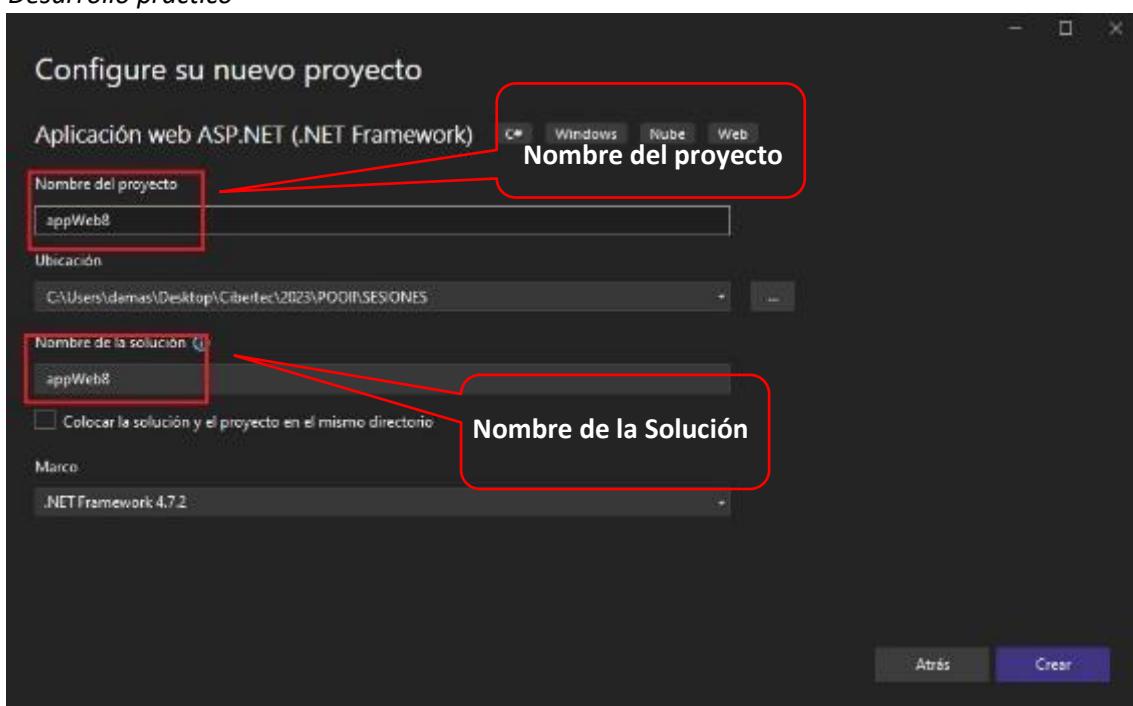
Figura 349
Desarrollo práctico



Nota. Elaboración propia.

En la ventana “Configure su nuevo proyecto”, ingrese el nombre del proyecto y selecciona la ubicación del mismo, y el nombre de la solución, al terminar presiona la opción **Crear**.

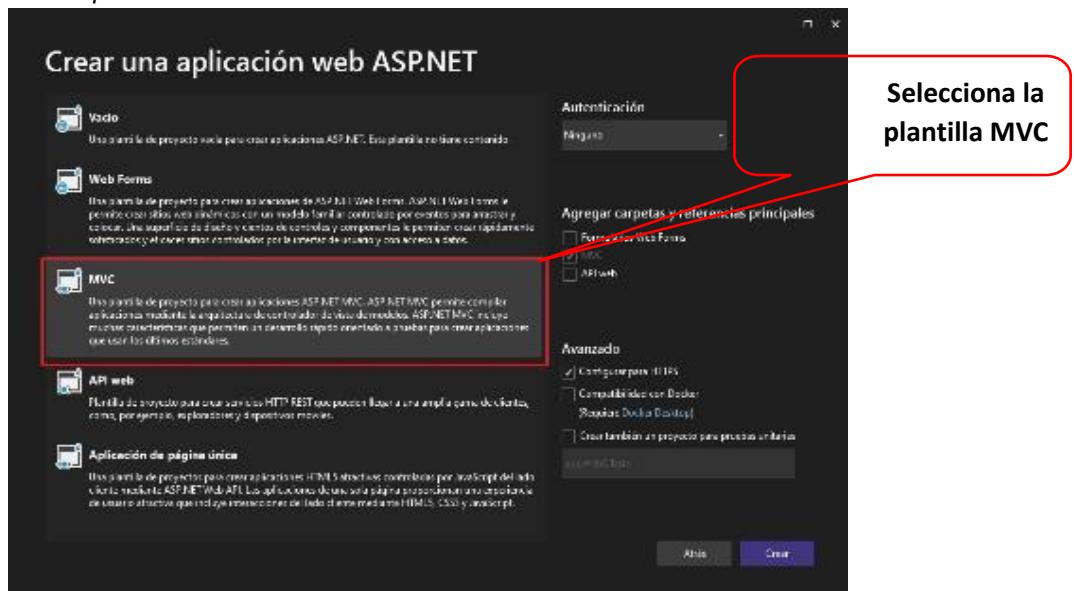
Figura 350
Desarrollo práctico



Nota. Elaboración propia.

A continuación, selecciona la plantilla MVC, tal como se muestra. Presiona el botón CREAR.

Figura 351
Desarrollo práctico

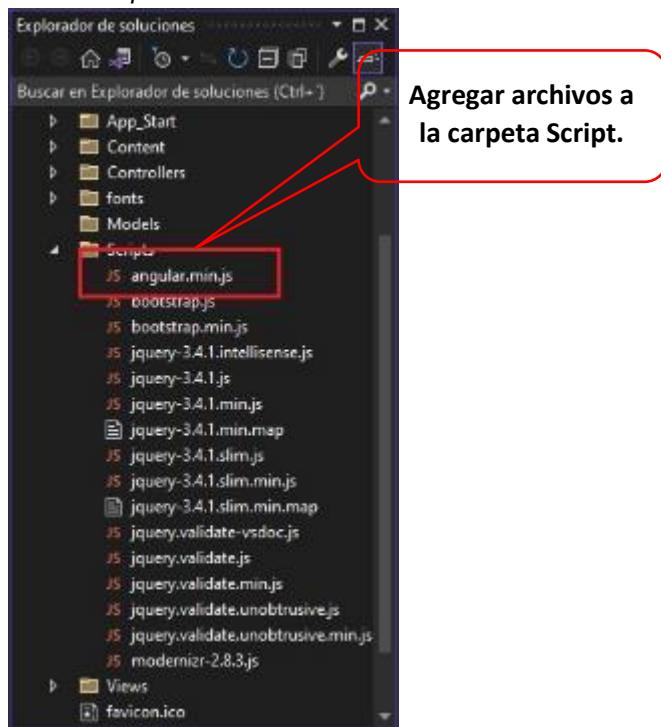


Nota. Elaboración propia.

Agregando el archivo angular.min.js

Desde el Explorador de proyecto, agregar el archivo angular.min.js en la carpeta Scripts, tal como se muestra.

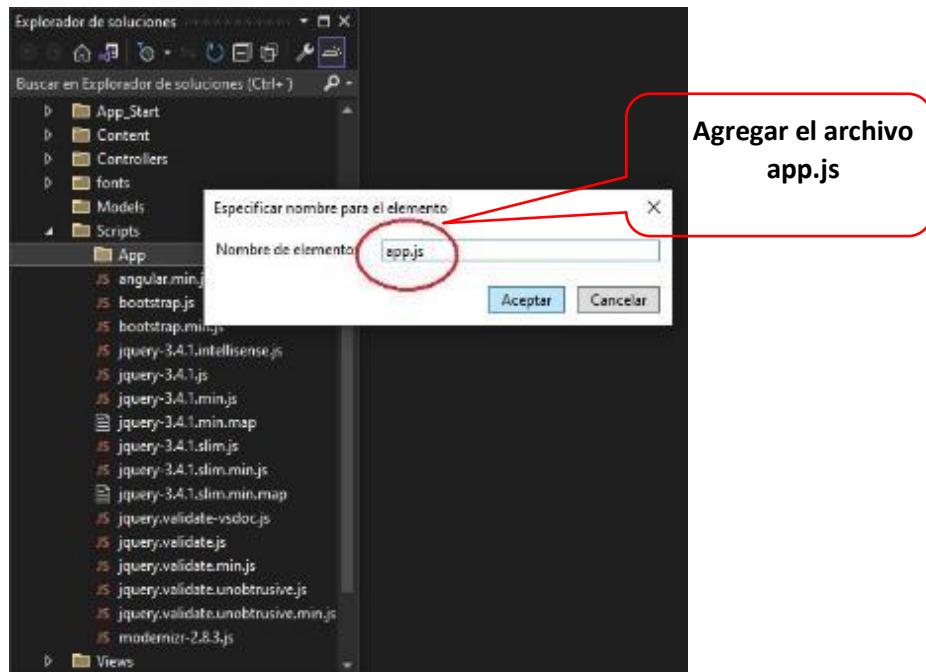
Figura 352
Desarrollo práctico



Nota. Elaboración propia.

A continuación, agregar desde la carpeta App un Nuevo Elemento: **Archivo JavaScript**, asignar el nombre **app**, tal como se muestra.

Figura 353
Desarrollo práctico



Nota. Elaboración propia.

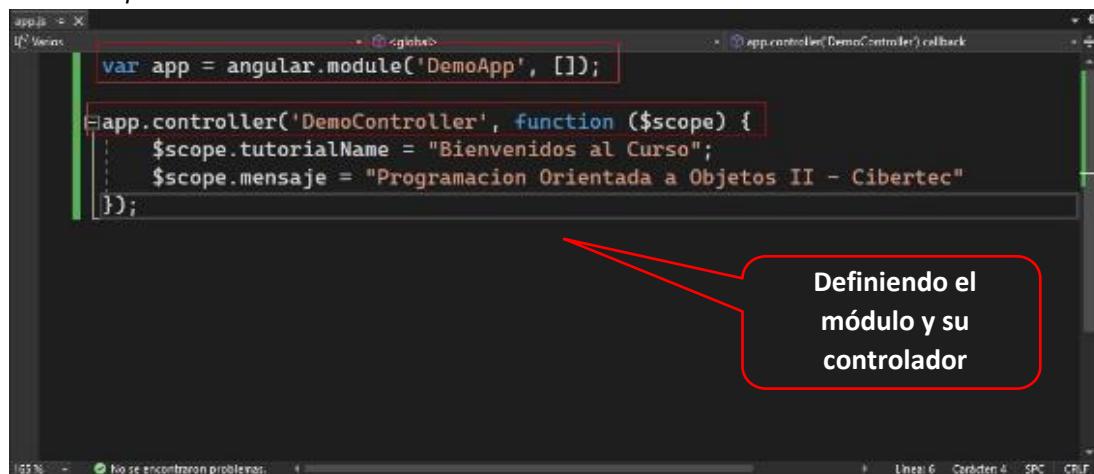
Trabajando con el archivo app.js

En el archivo app.js, agregue el siguiente código para su controlador:

- Defina el módulo “DemoApp” que mantendrá el controlador junto con la funcionalidad.
- Cree un controlador con el nombre “DemoController”. Este controlador se utilizará para tener una funcionalidad para mostrar los mensajes tutorialName y mensaje.
- El objeto Scope se usará para contener las variables tutorialName y mensaje, cada uno con su valor.

Figura 354

Desarrollo práctico



```
var app = angular.module('DemoApp', []);  
  
app.controller('DemoController', function ($scope) {  
    $scope.tutorialName = "Bienvenidos al Curso";  
    $scope.mensaje = "Programación Orientada a Objetos II - Cibertec"  
});
```

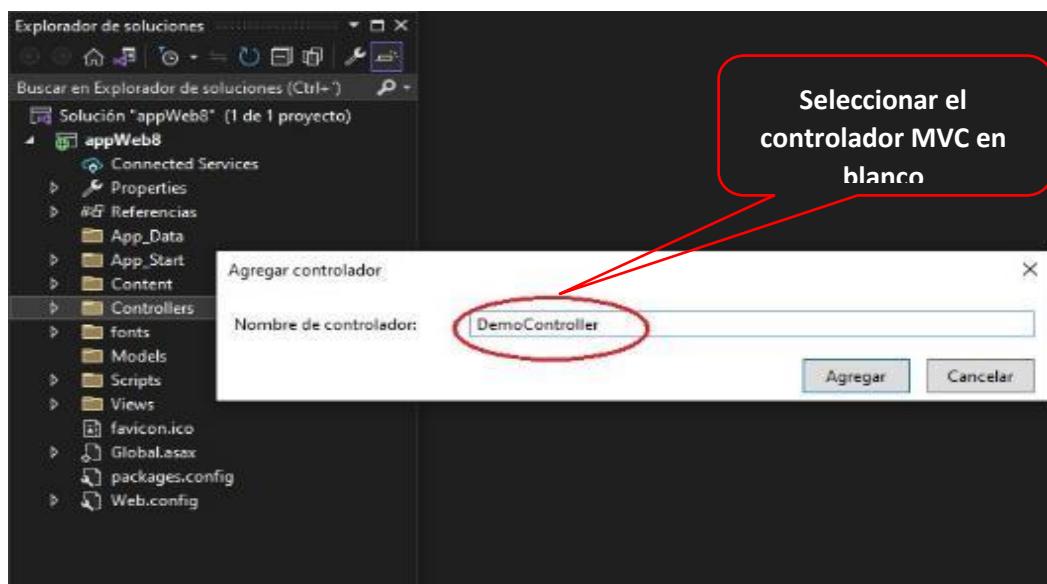
A red callout bubble points to the first two lines of code: 'var app = angular.module('DemoApp', []);'. The text inside the bubble reads: 'Definiendo el módulo y su controlador'.

Nota. Elaboración propia.

Agregando el controlador DemoController

En la carpeta Controller agregar el controlador DemoController, seleccionar el elemento Controlador de MVC: en blanco, asignar el nombre DemoController, tal como se muestra.

Figura 355
Desarrollo práctico

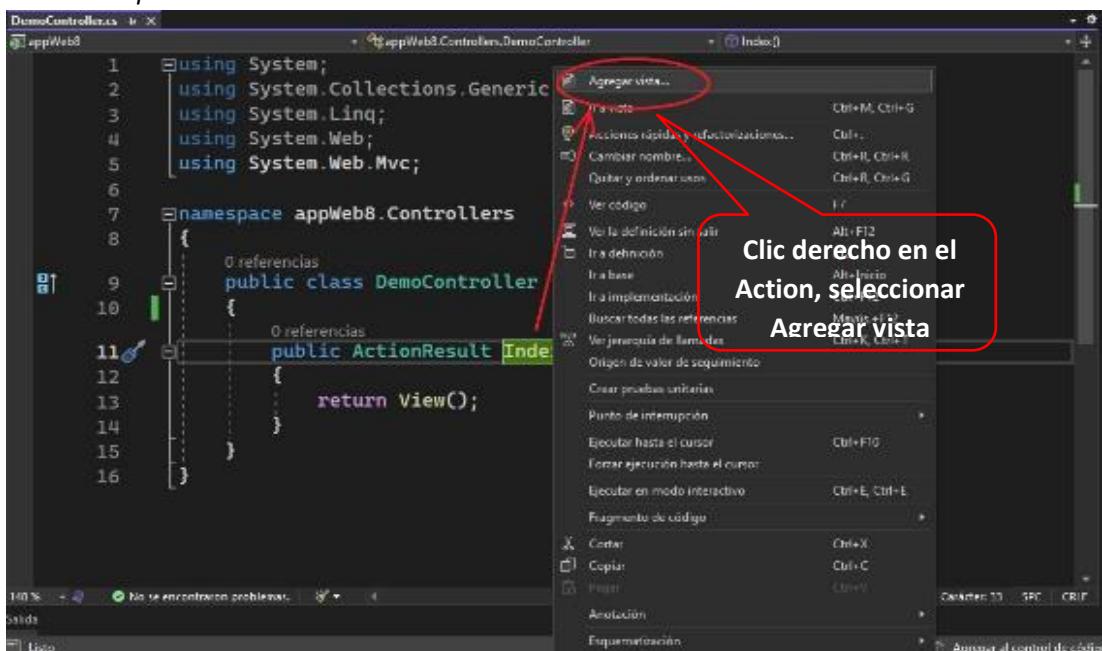


Nota. Elaboración propia.

Agregando la Vista Index

En el controlador DemoController tenemos un ActionResult Index, a continuación, vamos a agregar una vista, tal como se muestra.

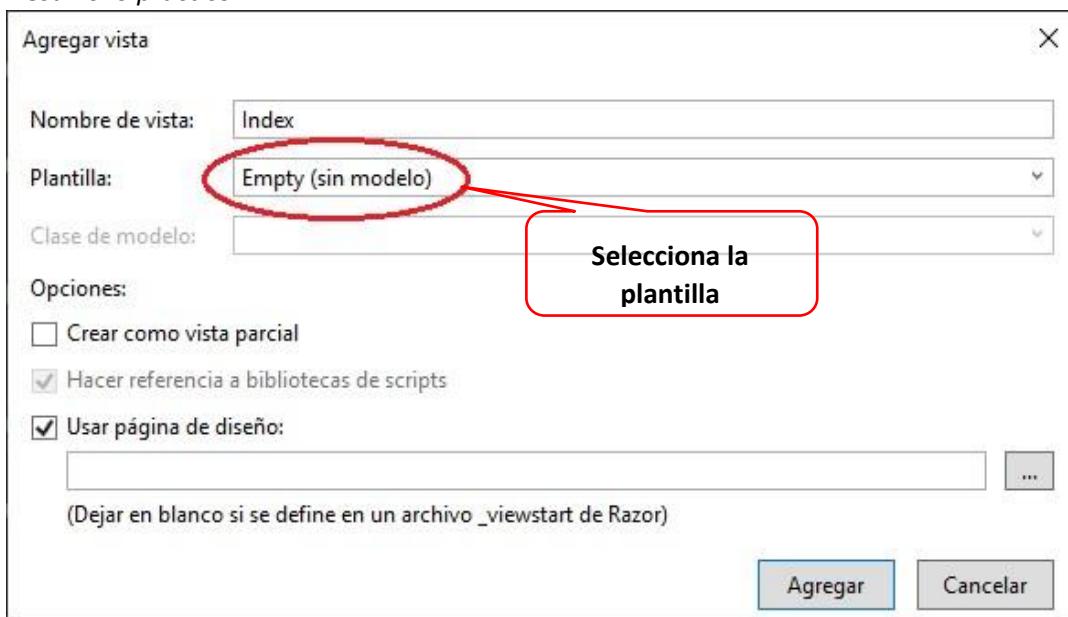
Figura 356
Desarrollo práctico



Nota. Elaboración propia.

Selecciona la Vista de Razor de plantilla Empty (sin modelo) presione el botón Agregar.

Figura 357
Desarrollo práctico



Nota. Elaboración propia.

Ahora, en su archivo Index.cshtml agregue una etiqueta div que contendrá la directiva ngcontroller y luego agregue una referencia a las variables “tutorialName” y “mensaje”. Agregar las librerías angular.min.js y el archivo app.js

Figura 358
Desarrollo práctico

```

@{
    ViewBag.Title = "Index";
}



## Index



# {{tutorialName}}



## {{mensaje}}

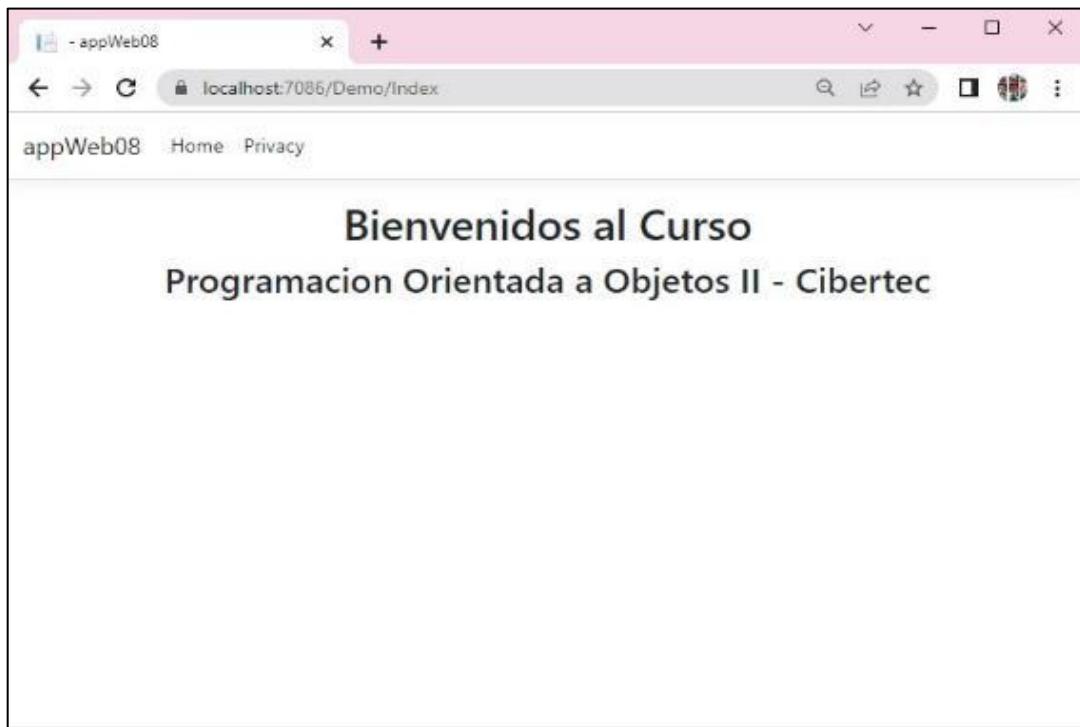


<script src("~/Scripts/angular.min.js")></script>
<script src "~/Scripts/App/app.js"></script>

```

Nota. Elaboración propia.

Presiona la tecla F5 para ejecutar el Action (Index) del controlador (Demo), visualizando los objetos definidos en app.js

Figura 359*Desarrollo práctico*

Nota. Elaboración propia.

LABORATORIO 8.2.:

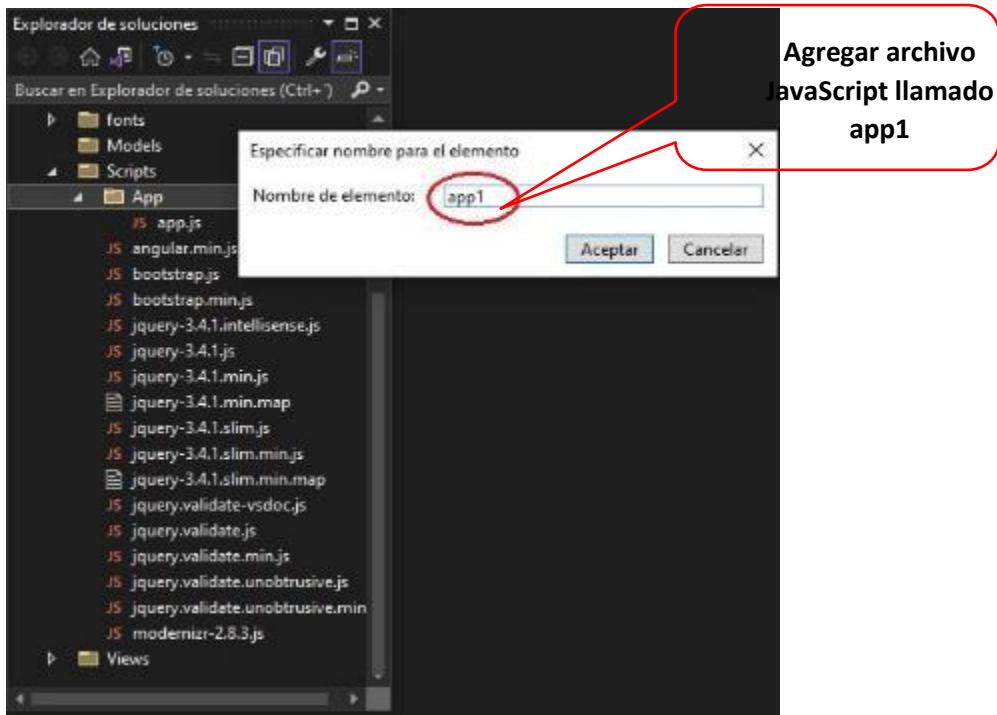
Ingreso de datos en una aplicación ASP.NET MVC

Implemente una aplicación en ASP.NET MVC donde ingrese los datos y visualice los valores en un mensaje utilizando Angular

Agregando un archivo JavaScript

Trabajando en el proyecto anterior, agregar en la carpeta angular un archivo JavaScript llamado app1.js, tal como se muestra.

Figura 360
Desarrollo práctico



Nota. Elaboración propia.

En el archivo app1.js, defina dentro del controller los elementos nombre y apellido y el método mostrar donde visualizamos el nombre y apellido ingresado.

Figura 361
Desarrollo práctico

```
app1.js
var app = angular.module('DemoApp', []);

app.controller('DemoController', function ($scope) {
    $scope.nombre = "";
    $scope.apellido = "";
    $scope.mostrar = function () {
        alert($scope.nombre + ', ' + $scope.apellido);
    };
});
```

Programe el archivo

Nota. Elaboración propia.

Agregando un ActionResult al Controlador DemoController

A continuación, agregamos al controlador un ActionResult llamado Ingreso().

Figura 362*Desarrollo práctico*

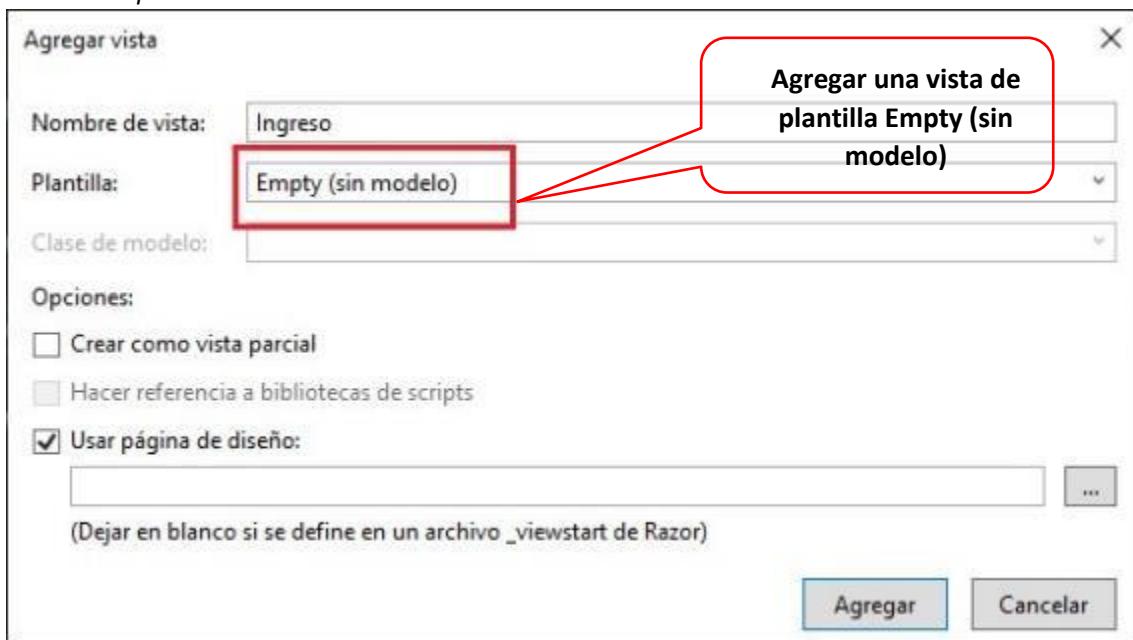
```

1  using ...
2
3  namespace appWeb8.Controllers
4  {
5      public class DemoController : Controller
6      {
7          public ActionResult Index()
8          {
9              return View();
10         }
11     }
12 }
13
14
15
16
17
18
19
20

```

Nota. Elaboración propia.

Agregar una vista de plantilla Empty (sin modelo).

Figura 363*Desarrollo práctico**Nota.* Elaboración propia.

Defina un bloque div con las etiquetas ng-app y ng-controller; dentro del bloque agregar dos input cada uno con su ng-model y un button con su etiqueta ng-click ejecutando el metodo mostrar(). Al final agregar las referencias a los archivos angular.min.js y app1.js.

Figura 364*Desarrollo práctico*

The screenshot shows an IDE interface with two tabs: 'Ingreso00.html' and 'DemoController.js'. The 'Ingreso00.html' tab contains the following HTML code:

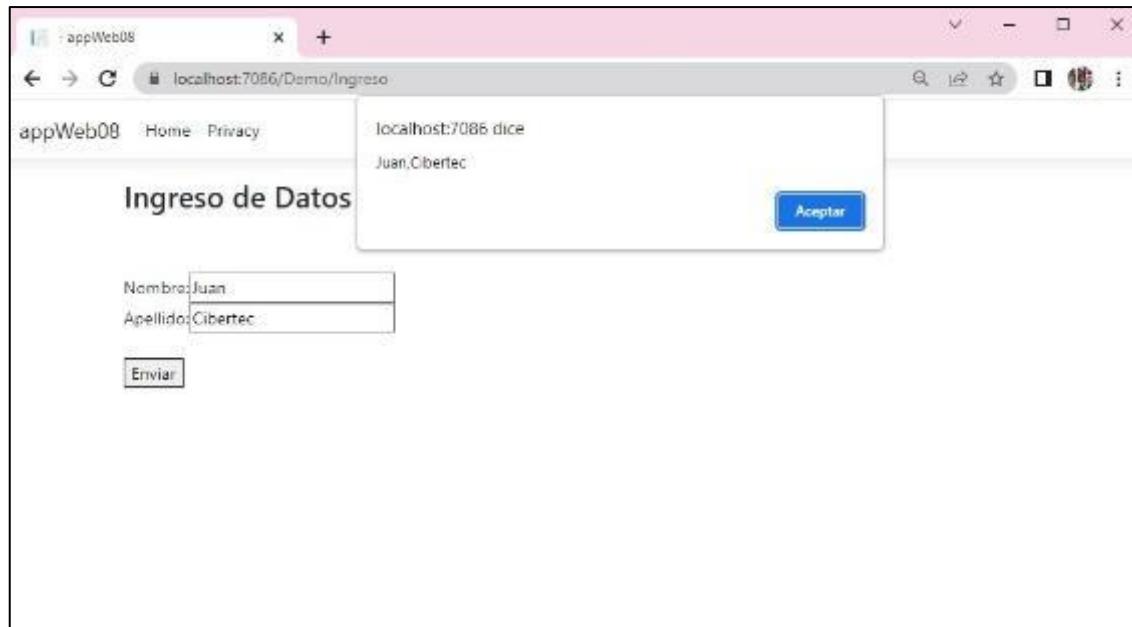
```
<div ng-app="DemoApp" ng-controller="DemoController">
  <h3>Ingreso de Datos</h3>
  <br /><br />
  <div>
    Nombre:<input type="text" ng-model="nombre" />
    <br />
    Apellido:<input type="text" ng-model="apellido" />
    <br /><br />
    <button ng-click="mostrar()">Enviar</button>
  </div>
</div>

<script src("~/Scripts/angular.min.js")></script>
<script src "~/Scripts/App/app1.js"></script>
```

A red box highlights the main HTML structure, and another red box highlights the script tags at the bottom. A callout bubble labeled 'Programando angular' points to the highlighted HTML area, and another callout bubble labeled 'Referencias a los scripts' points to the highlighted script tags.

Nota. Elaboración propia.

Ejecuta el proyecto, ingresa los datos en los inputs, al presionar el botón Enviar visualizamos el nombre y apellido ingresado.

Figura 365*Desarrollo práctico*

Nota. Elaboración propia.

LABORATORIO 8.3.:

Listar una colección de datos en una aplicación ASP.NET MVC

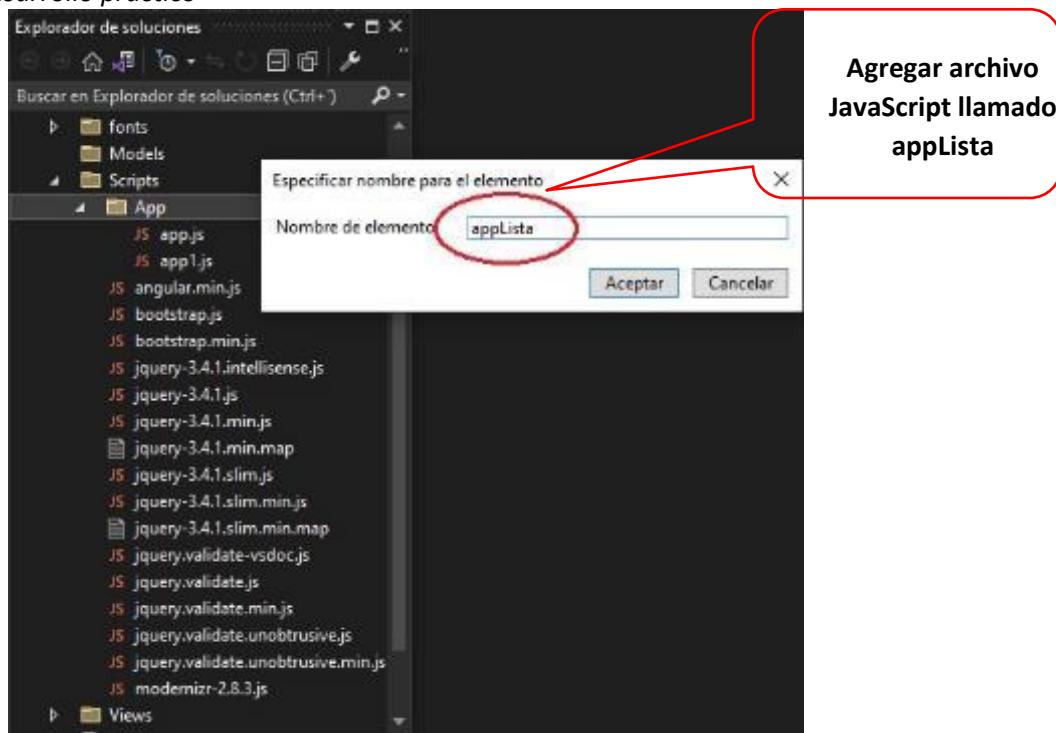
Implemente una aplicación en ASP.NET MVC donde liste los datos de los alumnos en una vista utilizando Angular.

Agregando un archivo JavaScript

Trabajando en el proyecto anterior, agregar en la carpeta app el archivo appLista.js

Figura 366

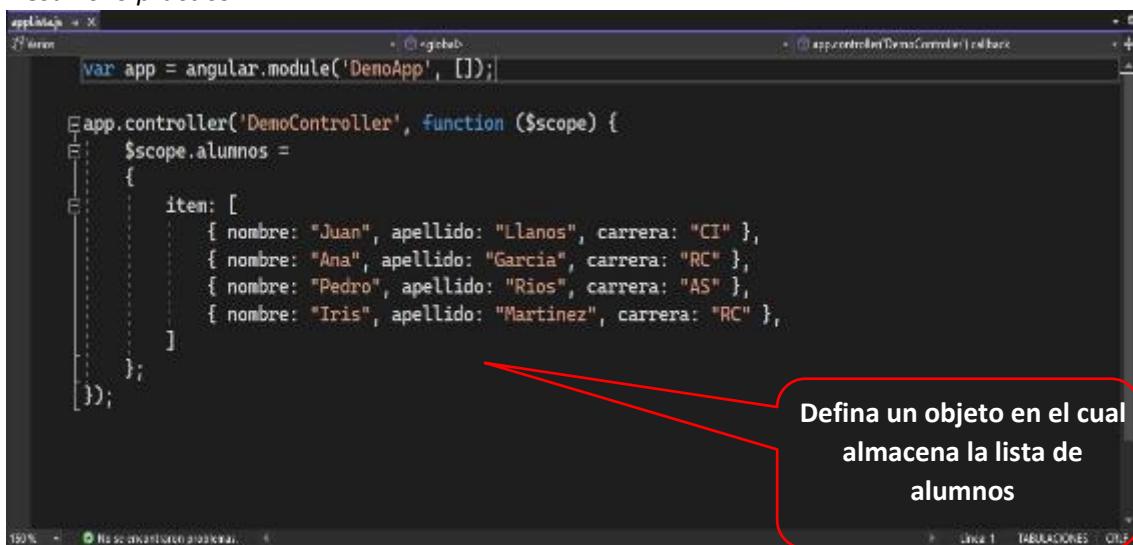
Desarrollo práctico



Nota. Elaboración propia.

En el archivo appLista.js, defina dentro del controller la lista de alumnos llamado item almacenando nombre, apellido y carrera.

Figura 367
Desarrollo práctico



```

var app = angular.module('DemoApp', []);

app.controller('DemoController', function ($scope) {
  $scope.alumnos =
  {
    item: [
      { nombre: "Juan", apellido: "Llanos", carrera: "CI" },
      { nombre: "Ana", apellido: "Garcia", carrera: "RC" },
      { nombre: "Pedro", apellido: "Rios", carrera: "AS" },
      { nombre: "Iris", apellido: "Martinez", carrera: "RC" }
    ]
});

```

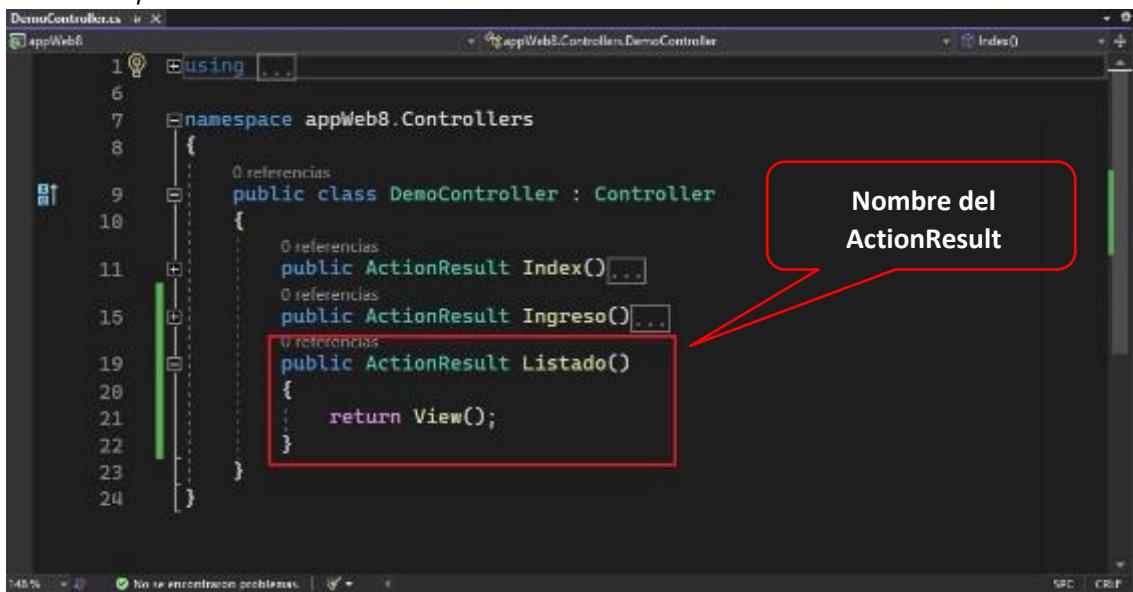
Defina un objeto en el cual almacena la lista de alumnos

Nota. Elaboración propia.

Agregando un ActionResult al Controlador

A continuación, agregamos al controlador un ActionResult llamado Listado(), tal como se muestra.

Figura 368
Desarrollo práctico



```

using ...
namespace appWeb8.Controllers
{
  public class DemoController : Controller
  {
    public ActionResult Index()
    public ActionResult Ingreso()
    public ActionResult Listado()
    {
      return View();
    }
  }
}

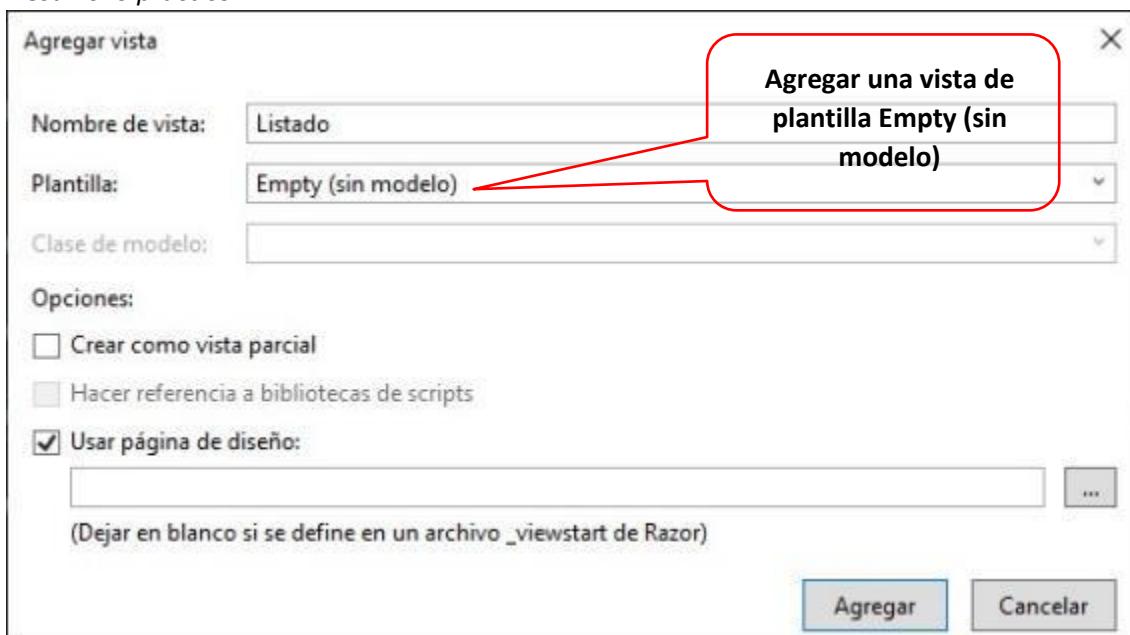
```

Nombre del ActionResult

Nota. Elaboración propia.

Agregar una vista vacía al ActionResult llamado Listado.cshtml.

Figura 369
Desarrollo práctico



Nota. Elaboración propia.

En el Listado.cshtml, defina un bloque div con las etiquetas ng-app y ng-controller; defina un table para listar los elementos de item utilizando la etiqueta ng-repeat. Al final agregar las referencias a los archivos angular.min.js y appLista.js.

Figura 370
Desarrollo práctico

```

<div ng-app="DemoApp" ng-controller="DemoController">
    <div>
        <table class="table">
            <tr>
                <td colspan="3"><h2>Listado de Alumnos</h2></td>
            </tr>
            <tr>
                <th>Nombre</th>
                <th>Apellido</th>
                <th>Carrera</th>
            </tr>
            <tr ng-repeat="it in alumnos.item">
                <td>{{it.nombre}}</td>
                <td>{{it.apellido}}</td>
                <td>{{it.carrera}}</td>
            </tr>
        </table>
    </div>
</div>

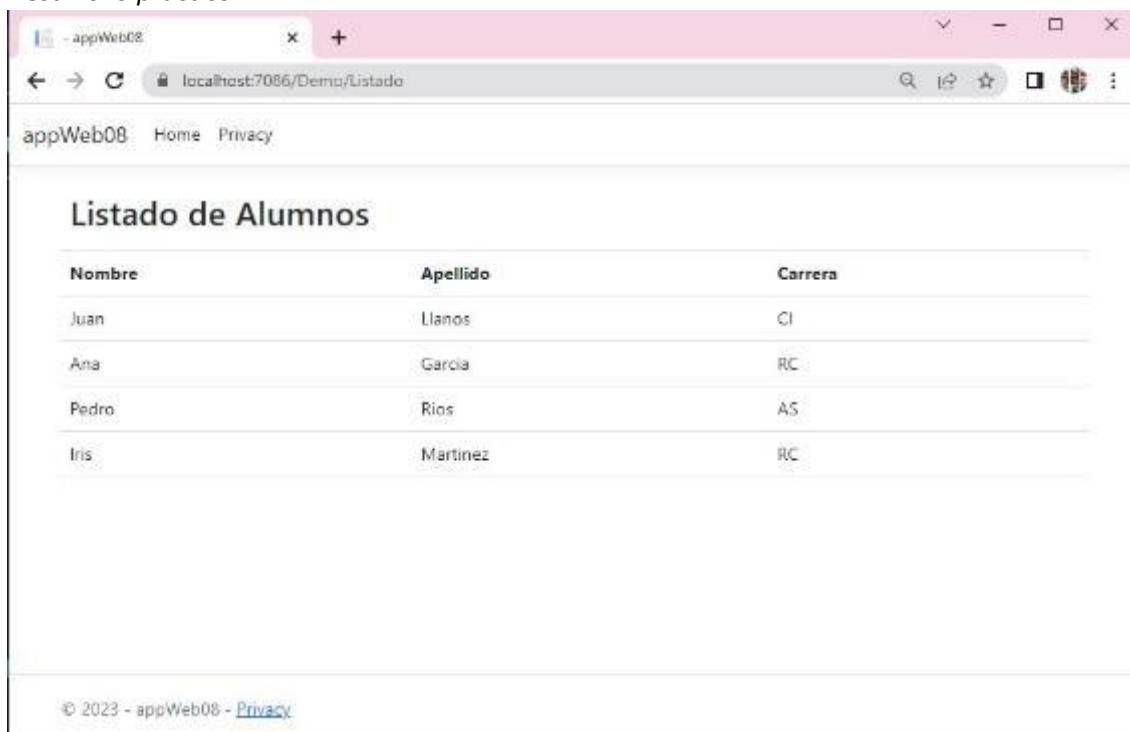
<script src="~/Scripts/angular.min.js"></script>
<script src="~/Scripts/App/appLista.js"></script>

```

Nota. Elaboración propia.

.Ejecuta el proyecto, donde lista en la vista los registros de los alumnos almacenados en @scope.alumnos.

Figura 371
Desarrollo práctico



Nota. Elaboración propia.

LABORATORIO 8.4.:

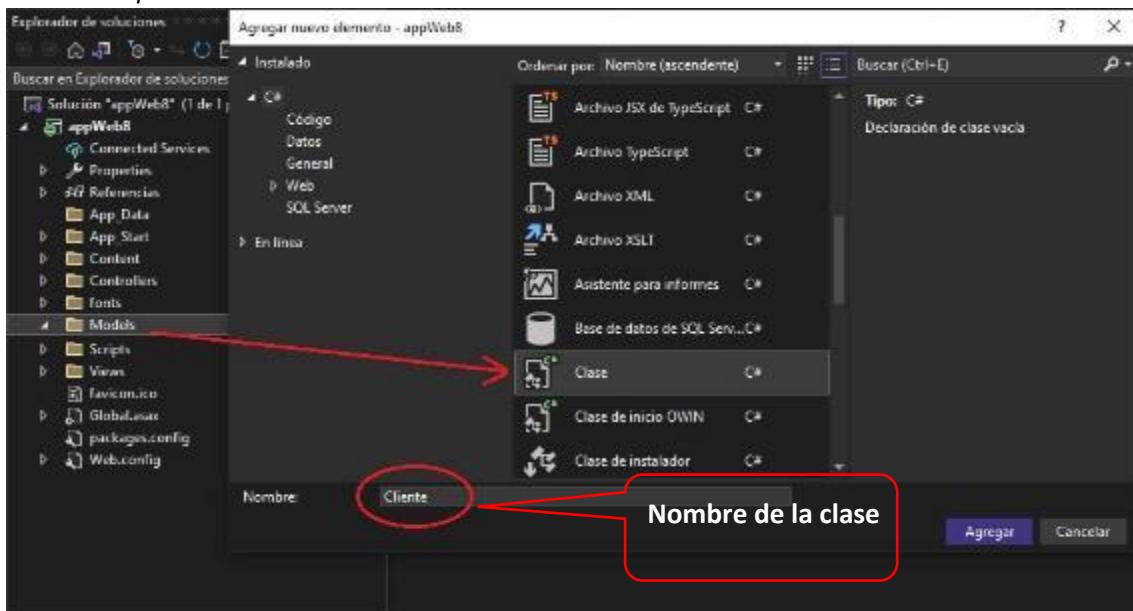
Listar los registros de la tabla tb_clientes en una aplicación ASP.NET MVC

Implemente una aplicación en ASP.NET MVC donde liste los registros de la tabla tb_clientes almacenado en la base de datos Negocios2022 utilizando Angular.

Trabajando con el Modelo de Datos

En la carpeta Models, agregar la clase Cliente, tal como se muestra.

Figura 372
Desarrollo práctico



Nota. Elaboración propia.

Defina los atributos y propiedades de la clase Cliente, tal como se muestra.

Figura 373
Desarrollo práctico

```

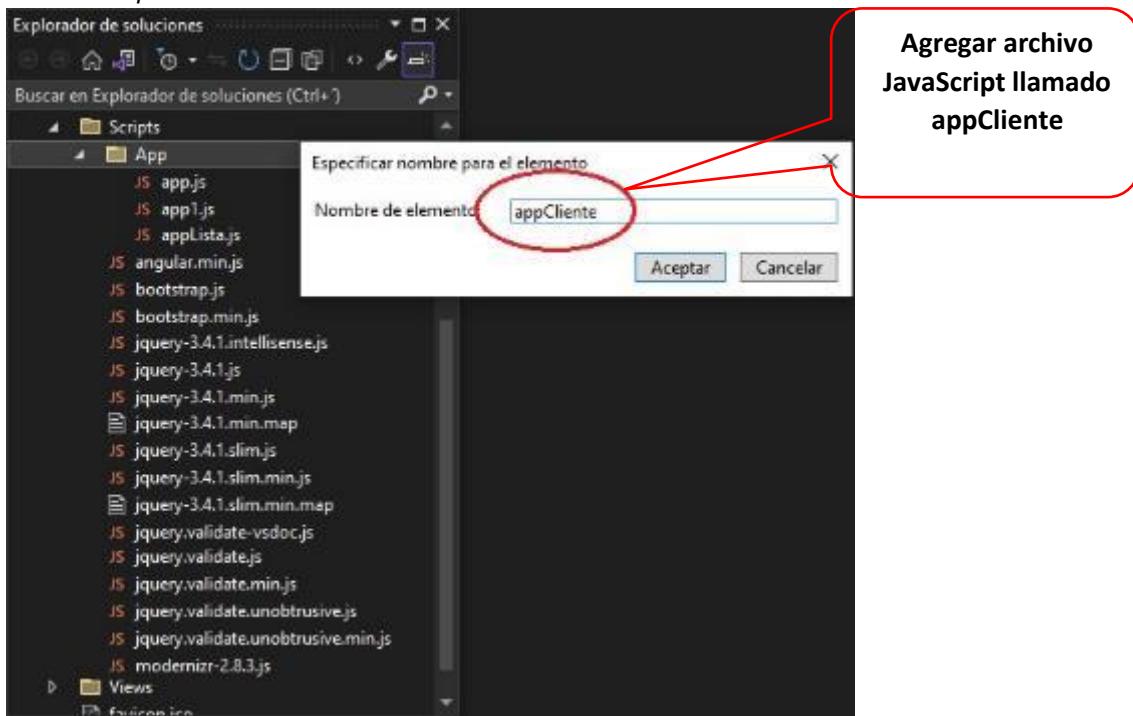
1 using ...
5
6 namespace appWeb8.Models
7 {
8     public class Cliente
9     {
10         public string idcliente { get; set; }
11         public string nombrecia { get; set; }
12         public string direccion { get; set; }
13         public string idpais { get; set; }
14         public string telefono { get; set; }
15     }
16 }

```

Nota. Elaboración propia.

Agregando un archivo JavaScript

Trabajando en el proyecto anterior, agregar en la carpeta app el archivo appCliente.js.

Figura 374*Desarrollo práctico**Nota. Elaboración propia.*

En el archivo appClientes.js, recuperamos el resultado en formato json, del método LoadClientes almacenando en \$scope.Clientes.

Figura 375*Desarrollo práctico*

```

appCliente.js ✎ X
Venos <global> app
var app = angular.module('DemoApp', []);
app.controller('DemoController', function ($scope, $http) {
    $http.get('/Demo/LoadClientes').then(function (d) {
        $scope.Cientes = d.data;
    }, function (error) {
        alert("No se puede listar");
    });
});

```

The screenshot shows the 'appCliente.js' file in the code editor. The code defines an Angular module 'DemoApp' and a controller 'DemoController'. The controller uses the \$http service to make a GET request to '/Demo/LoadClientes'. If the request is successful, it sets the \$scope.Cientes variable to the received data. If there is an error, it displays an alert message. A red box highlights the line '\$scope.Cientes = d.data;' with the annotation 'Recupero el resultado de LoadClientes almacenando en \$scope.Cientes'.

Nota. Elaboración propia.

Trabajando con el Controlador DemoController

Para comenzar, agregar las librerías para recuperar los datos de tb_clientes en SQL Server.

Figura 376*Desarrollo práctico*

```

1 @> using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Data.SqlClient;
6 using System.Web.Mvc;
7 using appWeb8.Models;
8 namespace appWeb8.Controllers
9 {
10     public class DemoController : Controller
11     {
12         public ActionResult Index()
13         {
14             public ActionResult Ingreso()
15             {
16                 public ActionResult Listado()
17             }
18         }
19     }
20 }
21 
```

Nota. Elaboración propia.

Agregar el método JsonResult el cual retorna, la lista de los registros de tb_clientes en formato Json, tal como se muestra.

Figura 377*Desarrollo práctico*

```

24 public JsonResult LoadClientes()
25 {
26     List<Cliente> temporal = new List<Cliente>();
27     using (var cn = new SqlConnection(
28         @"server=LAPTOP-OBNAIDIG9\SQLEXPRESS;database=Negocios2022;Integrated security=true"))
29     {
30         cn.Open();
31         SqlCommand cmd = new SqlCommand("Select * from tb_clientes", cn);
32         SqlDataReader dr = cmd.ExecuteReader();
33         while (dr.Read())
34         {
35             temporal.Add(new Cliente()
36             {
37                 idcliente = dr.GetString(0),
38                 nombrecia = dr.GetString(1),
39                 direccion = dr.GetString(2),
40                 idpais = dr.GetString(3),
41                 telefono = dr.GetString(4)
42             });
43         }
44         dr.Close();
45     }
46     return Json(temporal, JsonRequestBehavior.AllowGet);
47 }
48 
```

Nota. Elaboración propia.

A continuación, agregamos al controlador un ActionResult llamado ListadoCliente(), tal como se muestra.

Figura 378
Desarrollo práctico

```

10    public class DemoController : Controller
11    {
12        public ActionResult Index()
13        {
14            return View();
15        }
16        public ActionResult Ingreso()
17        {
18            return View();
19        }
20        public ActionResult Listado()
21        {
22            return View();
23        }
24        public JsonResult LoadClientes()
25        {
26            return Json(new List<Cliente>());
27        }
28        public ActionResult ListadoCliente()
29        {
30            return View();
31        }
32    }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

```

Nota. Elaboración propia.

Agregar una vista vacía al ActionResult llamado ListadoCliente.cshtml.

Figura 379
Desarrollo práctico



Nota. Elaboración propia.

En ListadoClientes.cshtml, defina un bloque div con las etiquetas ng-app y ng-controller; defina un table para listar los elementos de Clientes utilizando la etiqueta ng-repeat. Al final agregar las referencias a los archivos angular.min.js y appCliente.js.

Figura 380*Desarrollo práctico*

```

<div ng-app="DemoApp" ng-controller="DemoController">
  <div>
    <table class="table">
      <tr>
        <td colspan="5"><h2>Listado de Clientes</h2></td>
      </tr>
      <tr>
        <th>Id Cliente</th>
        <th>Nombre</th>
        <th>Direccion</th>
        <th>Id Pais</th>
        <th>Telefono</th>
      </tr>
      <tbody>
        <tr ng-repeat="d in Clientes">
          <td>{{d.idcliente}}</td>
          <td>{{d.nombrecia}}</td>
          <td>{{d.direccion}}</td>
          <td>{{d.idpais}}</td>
          <td>{{d.telefono}}</td>
        </tr>
      </tbody>
    </table>
  </div>
<script src="/Scripts/angular.min.js"></script>
<script src="/Scripts/App/appCliente.js"></script>

```

Nota. Elaboración propia.

Ejecuta el proyecto, donde lista los registros de los clientes almacenados en @scope.Clientes

Figura 381*Desarrollo práctico*

Id Cliente	Nombre	Direccion	Id Pais	Telefono
ALFKI	Alfreds Futterkiste	Obere Str. 57	002	030-0074321
ANATR	Ana Trujillo Emparedados y helados	Avda. de la Constitucion 2222	005	(5) 555-4729
ANTON	Antonio Moreno Taqueria	Mataderos 2312	007	(5) 555-3932
AROUT	Around the Horn	120 Hanover Sq	004	(71) 555-7788
BERGS	Berglunds snabbköp	Berguvsvagen 8	006	0921-12 34 65
BLAUS	Blauer See Delikatessen	Forststr. 57	001	0621-08460
BLONP	Blondel père et fils	24, place Kleber Estrasburgo	008	88 60 15 31
BOLID	Bolido Comidas preparadas	C/ Araquil, 67	009	(91) 555 91 99
BONAP	Bon app	12, rue des Bouchers	001	91 24 45 41
BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	003	(604) 555-3745
BSBEV	Bis Beverages	Faunlerooy Circus	009	(71) 555-1212
CACTU	Cactus Comidas para llevar	Cerito 333	002	(1) 135-4892
CENTC	Centro comercial Mocedezma	Sierras de Granada 9993	008	(5) 555-7293

Nota. Elaboración propia.

LABORATORIO 8.5.

Insertar los registros a la tabla tb_clientes en una aplicación ASP.NET MVC

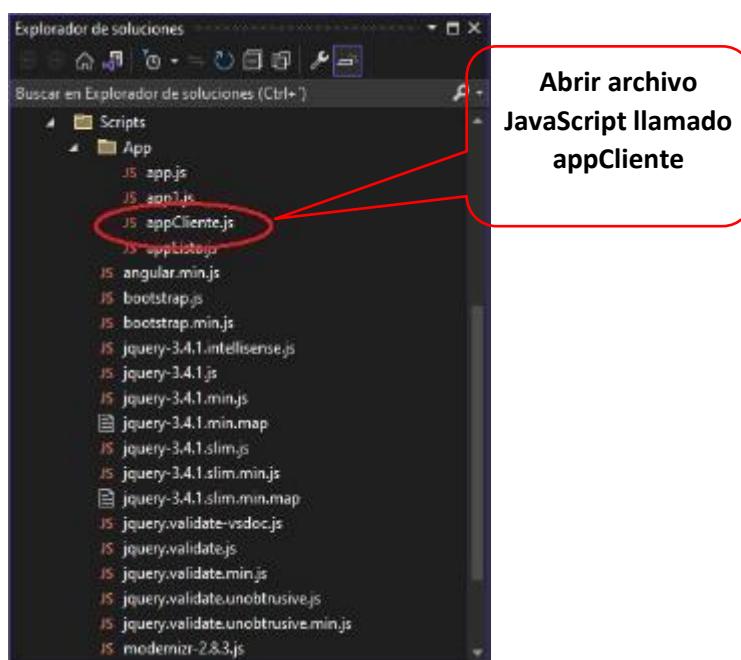
Implemente una aplicación en ASP.NET MVC donde inserte registros a la tabla tb_clientes almacenado en la base de datos Negocios2022 utilizando Angular.

Trabajando con el archivo appCliente de JavaScript

Utilizando el script del Laboratorio anterior, abrir el archivo appCliente.js, tal como se muestra.

Figura 382

Desarrollo práctico



Nota. Elaboración propia.

En el archivo appClientes.js, defina el metodo SaveData, para ejecuta el metodo SaveCliente.

Figura 383*Desarrollo práctico*

```

var app = angular.module('DemoApp', []);
app.controller('DemoController', function ($scope, $http) {
    $http.get('/Demo/LoadClientes').then(...);

    $scope.SaveData = function () {
        $http({
            method: 'POST',
            url: '/Demo/SaveCliente',
            data: $scope.Cliente
        }).success(function (a) {
            $scope.btnSaveCliente = "Save";
            $scope.Cliente = null;
            alert(a);
        }).error(function () {
            alert("Faild");
        });
    };
});

```

Nota. Elaboración propia.

Trabajando con el Controlador DemoController

Agregar el método JsonResult el cual ejecuta el insert into a tb_clientes retornando el mensaje del proceso en formato Json, tal como se muestra.

Figura 384*Desarrollo práctico*

```

public JsonResult SaveCliente(Cliente reg)
{
    string mensaje = "";
    using (var cn = new SqlConnection(
        @"server=LAPTOP-OBKADIG9\SQLEXPRESS;database=Negocios2022;Integrated security=true"))
    {
        try
        {
            cn.Open();
            SqlCommand cmd = new SqlCommand(
                "Insert tb_clientes Values(@idcli,@nom,@dir,@idpais,@fono)", cn);
            cmd.Parameters.AddWithValue("@idcli", reg.idcliente);
            cmd.Parameters.AddWithValue("@nom", reg.nombrecia);
            cmd.Parameters.AddWithValue("@dir", reg.direccion);
            cmd.Parameters.AddWithValue("@idpais", reg.idpais);
            cmd.Parameters.AddWithValue("@fono", reg.telefono);
            int c=cmd.ExecuteNonQuery();

            mensaje = $"Se ha registrado {c} cliente";
        }
        catch (SqlException ex) { mensaje = ex.Message; }
        finally { cn.Close(); }
    }
    return Json(mensaje, JsonRequestBehavior.AllowGet);
}

```

Nota. Elaboración propia.

A continuación, agregar al controlador un ActionResult llamado Create(), tal como se muestra.

Figura 385
Desarrollo práctico

```

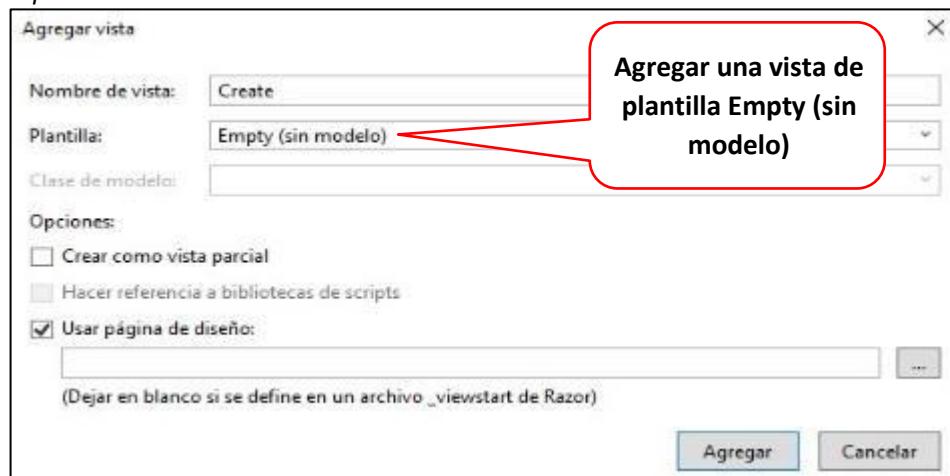
12     public class DemoController : Controller
13     {
14         public ActionResult Index()
15         {
16             return View();
17         }
18         public ActionResult Ingreso()
19         {
20             return View();
21         }
22         public ActionResult Listado()
23         {
24             return View();
25         }
26         public JsonResult LoadClientes()
27         {
28             return Json(new List<Cliente>());
29         }
30         public ActionResult ListadoCliente()
31         {
32             return View();
33         }
34         public JsonResult SaveCliente(Cliente reg)
35         {
36             return Json("OK");
37         }
38         public ActionResult Create()
39         {
40             return View();
41         }
42     }

```

Nota. Elaboración propia.

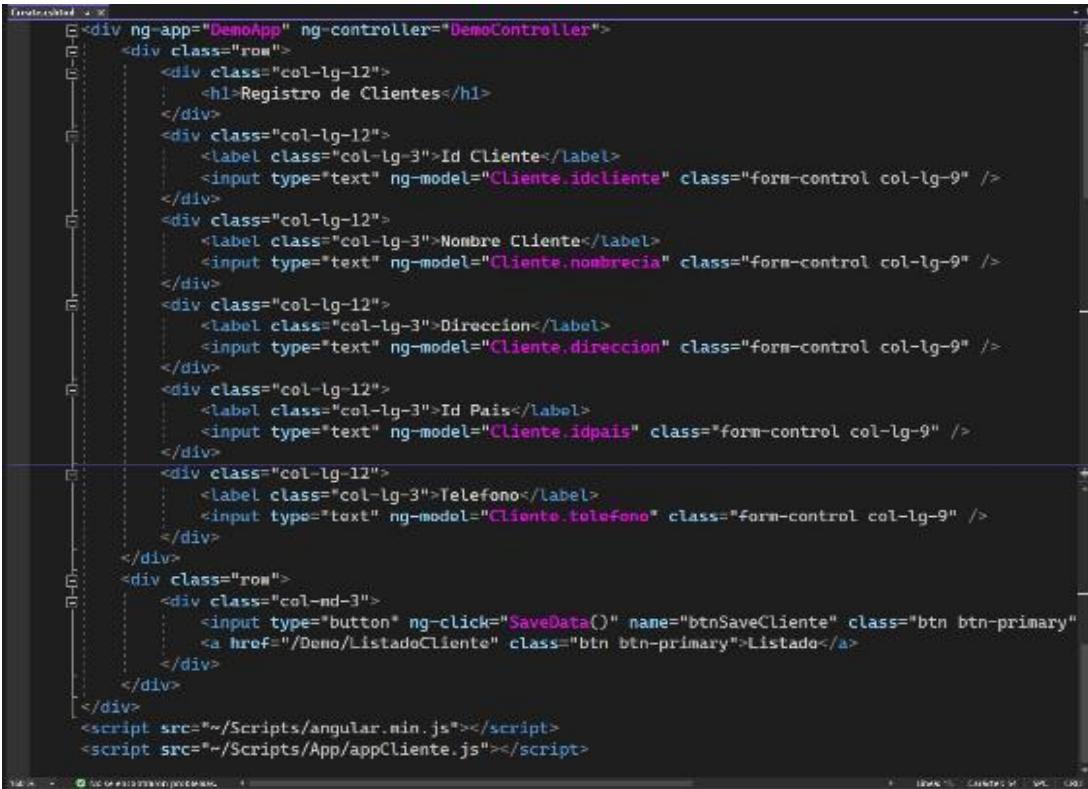
Agregar una vista vacía al ActionResult llamado Create.cshtml.

Figura 386
Desarrollo práctico



Nota. Elaboración propia.

En Create.cshtml, defina un bloque div con las etiquetas ng-app y ng-controller; defina los input para ingresar los datos, asigne el atributo ng-model. Al final agregar las referencias a los archivos angular.min.js y appCliente.js.

Figura 387*Desarrollo práctico*


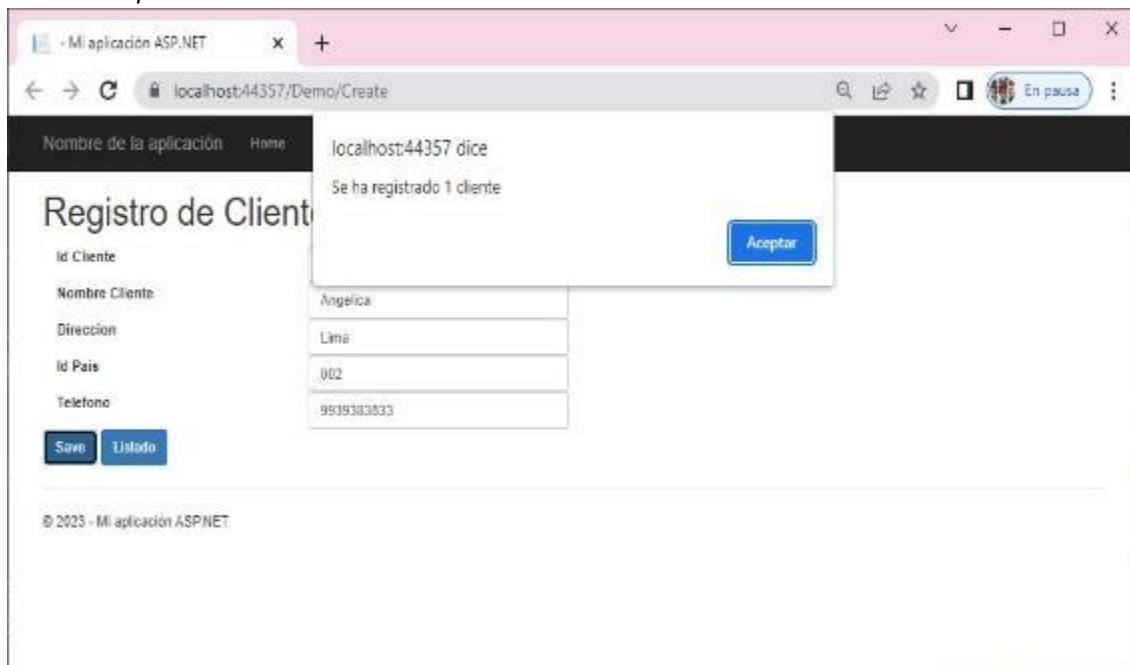
```

<div ng-app="DemoApp" ng-controller="DemoController">
  <div class="row">
    <div class="col-lg-12">
      <h1>Registro de Clientes</h1>
    </div>
    <div class="col-lg-12">
      <label class="col-lg-3">Id Cliente</label>
      <input type="text" ng-model="Cliente.idcliente" class="form-control col-lg-9" />
    </div>
    <div class="col-lg-12">
      <label class="col-lg-3">Nombre Cliente</label>
      <input type="text" ng-model="Cliente.nombrecia" class="form-control col-lg-9" />
    </div>
    <div class="col-lg-12">
      <label class="col-lg-3">Direccion</label>
      <input type="text" ng-model="Cliente.direccion" class="form-control col-lg-9" />
    </div>
    <div class="col-lg-12">
      <label class="col-lg-3">Id Pais</label>
      <input type="text" ng-model="Cliente.idpais" class="form-control col-lg-9" />
    </div>
    <div class="col-lg-12">
      <label class="col-lg-3">Telefono</label>
      <input type="text" ng-model="Cliente.telefono" class="form-control col-lg-9" />
    </div>
  </div>
  <div class="row">
    <div class="col-md-3">
      <input type="button" ng-click="SaveData()" name="btnSaveCliente" class="btn btn-primary" value="Guardar" />
      <a href="/Demo/ListaDeCliente" class="btn btn-primary">Listado</a>
    </div>
  </div>
</div>
<script src="~/Scripts/angular.min.js"></script>
<script src="~/Scripts/App/appCliente.js"></script>

```

Nota. Elaboración propia.

Ejecuta el proyecto, ingrese los datos en los controles, al presionar el botón Save ejecutamos el proceso y visualizamos el mensaje de la operación.

Figura 388*Desarrollo práctico**Nota.* Elaboración propia.

Resumen

1. Desde su creación, Angular ha sido el framework preferido por la mayoría de los desarrolladores JavaScript. Este éxito ha provocado que los desarrolladores quieran usar el framework para más y más cosas.
2. De ser una plataforma para la creación de Web Apps, ha evolucionado como motor de una enorme cantidad de proyectos del ámbito empresarial y de ahí para aplicaciones en la Web Mobile Híbrida, llevando la tecnología al límite de sus posibilidades.
3. El inyector de dependencias de Angular no necesita que estén en memoria todas las clases o código de todos los elementos que conforman una aplicación. Ahora con Lazy SPA el framework puede funcionar sin conocer todo el código de la aplicación, ofreciendo la posibilidad de cargar más adelante aquellas piezas que no necesitan todavía.
4. AngularJS es un framework de JavaScript que nos permite crear aplicaciones de “página única” o SPA. AngularJS también nos permite implementar el patrón de diseño MVVM.
5. Arrancamos el Visual Studio y creamos un nuevo proyecto de tipo Aplicación web ASP.NET .NET FrameWork con C#.
6. Una estrategia para trabajar con Angular en una aplicación ASP.NET MVC es definir un archivo js para la ejecución de sus métodos en las vistas.
7. A continuación, mostramos las siguientes directivas de AngularJS:
 - a) ng-controller: define el controlador
 - b) ng-repeat - repite la plantilla un cierto número de veces
 - c) ng-click: agrega una función al evento de clic del controlador AngularJS
 - d) ng-model: define un modelo de datos vinculado a nuestra vista
 - e) ng-disabled: deshabilita / habilita el botón

Recursos

Puede revisar los siguientes enlaces para ampliar los conceptos vistos en esta:

- <https://www.webnethelper.com/2021/06/grpc-services-using-net-5-using-entity.html>
- <https://www.csharp.com/article/grpc-service-create-using-net-core-6-0-entity-frameworkfor-crud-operation/>
- https://www-telerik-com.translate.goog/blogs/introduction-to-grpc-dotnet-core-and-dotnet-5?x_tr_sl=en&x_tr_tl=es&x_tr_hl=es&x_tr_pto=sc
- <https://www.enmilocalfunciona.io/net5-implementacion-de-servicios-grpc/>
- https://learn-microsoft-com.translate.goog/en-us/aspnet/core/grpc/aspnetcore?view=aspnetcore-6.0&tabs=visual-studio&x_tr_sl=en&x_tr_tl=es&x_tr_hl=es&x_tr_pto=sc

Bibliografía

- Aguilar, J. M. (s.f.). *Desarrollo Web con ASP.NET MVC 5.* https://www.campusmvp.es/catalogo/Product-Desarrollo-Web-con-ASP-NET-MVC-5_92.aspx?
- Alberca Rojas, V. W., y Cuello Moron, Y. I. (5 de octubre de 2017). *ADO.NET.* <http://desarrollolossoftwarevicente.blogspot.com/2017/10/adonet.html>
- Alvarez, M. A. (20 de septiembre de 2023). Qué es MVC. Desarrollo Web. <https://desarrolloweb.com/articulos/que-es-mvc.html>
- Alvarez, R. (24 de julio de 2021). Sesiones en PHP. <https://desarrolloweb.com/articulos/sesiones-en-php>
- Basalo, A. (19 de junio de 2019). *Angular para grandes aplicaciones.* Medium. <https://medium.com/@albertobasalo71/angular-para-grandes-aplicaciones-b66786fd3032>
- Basalo, A. (9 de junio de 2016). *Introducción a Angular.* Desarrollo Web. <https://desarrolloweb.com/articulos/introduccion-angular2.html>
- Gabillaud, J. (2017). *SQL Server 2016. Aprender a administrar una base de datos transaccional con SQL Server Management Studio).* Ediciones ENI.
- Guérin, B. A. (2018). *ASP.NET con C# en Visual Studio 2017. Diseño y desarrollo de aplicaciones Web.* Ediciones ENI.
- Hugon, J. (2018). *C# 7: Desarrolle aplicaciones Windows con Visual Studio 2017.* Ediciones ENI.
- IHMC Public Cmaps. (s.f.). *E-commerce.* <https://skat.ihmc.us/rid=1H6DK5SPJ-20YTF1T-TP0/e-commerce.docx>
- Improdex Desarrollo Empresarial. (s.f.). *El Comercio Electrónico: Conceptos Generales.* Creación de empresas. <https://www.creaciondempresas.es/gestion-y-actualidad/tecnologia/comercio-electronico-conceptos-generales/>
- Ingeniería Systems. (s.f.). *MVC - MVA - Microsoft Virtual Academy - Desarrollo en Microsoft Visual Studio - Módulo 35 de 44 - Avanzado.* <https://www.ingenieriasystems.com/2013/05/mvc-mva-microsoft-virtual-academy.html>
- Knowpia. (2009). *ADO.NET.* <https://es.knowpia.com/pages/ADO.NET>
- Microsoft. (2022). *Información general sobre ASP.NET MVC.* <https://learn.microsoft.com/es-es/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview>
- Mozilla Foundation. (13 de noviembre de 2023). *MVC.* <https://developer.mozilla.org/es/docs/Glossary/MVC>
- Tiendas Virtuales. (s.f.). *¿Qué es un carrito de venta?.* <https://www.tiendasvirtualesycomercioweb.com/preguntas-frecuentes/656-que-es-un-carrito-de-venta>
- Torres Remón, M. A. (2016). *Programación Orientada a Objetos con Visual C# (2015) y ADO.NET 4.6.* Macro.