



TI-Capital Humano

Desarrollador .Net

Manual del Curso



Contenido

1	Arquitectura de la Aplicación	3
1.1.1	Aplicación monolítica	3
1.1.2	Arquitectura Cliente - Servidor.....	4
1.1.3	Arquitectura en capas.....	5
1.1.4	Configuración de Solución en 4 Capas.....	8
1.2	Estructura de una Biblioteca de Clases.....	16
1.2.1	Propiedades de un Proyecto	16
1.2.2	Referencias	19
1.3	Capa de Entidades.....	20
1.3.1	Clases POCO y DTO.....	20
1.3.2	Propiedades	21
1.3.3	Crear Clases.....	23
1.4	Capa de Datos	25
1.5	Capa de Negocio	25
1.6	Capa de Presentación.....	26
1.6.1	Estructura de una Aplicación Web Forms.....	26
Los proyectos que usan PackageReference no usan packages.config.		¡Error!
Marcador no definido.		

1 Arquitectura de la Aplicación

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema.

- Una arquitectura de software, también denominada *arquitectura lógica*, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software.
- Una arquitectura de software se selecciona y diseña con base en objetivos (requisitos) y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como el mantenimiento, la auditoria, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas. Por ejemplo, no es viable emplear una arquitectura de software de tres capas para implementar sistemas en tiempo real.
- La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada cada tarea.

Generalmente, no es necesario inventar una nueva arquitectura de software para cada sistema de información. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Así, las arquitecturas más universales son:

- Aplicación Monolítica. [Cliente-servidor](#). Donde el software reparte su carga de cómputo en dos partes independientes pero sin reparto claro de funciones.
- Arquitectura en capas. Especialización de la arquitectura cliente-servidor donde la carga se divide en tres partes (o capas) con un reparto claro de funciones: una capa para la presentación (interfaz de usuario), otra para el cálculo (donde se encuentra modelado el negocio) y otra para el almacenamiento (persistencia). Una capa solamente tiene relación con la siguiente.

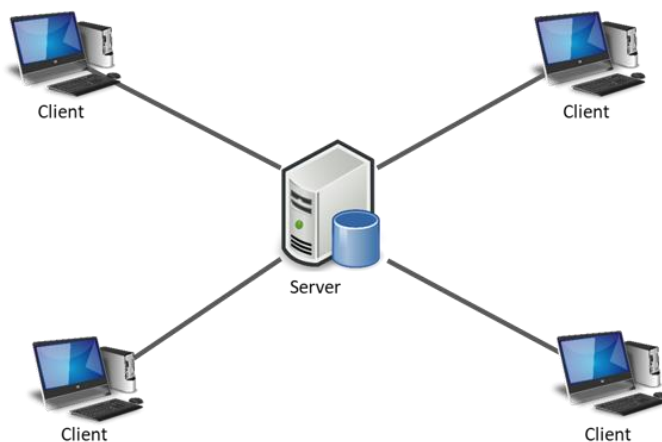
1.1.1 Aplicación monolítica

En ingeniería de software, una aplicación monolítica hace referencia a una aplicación software en la que la capa de interfaz de usuario y la capa de acceso a datos están combinadas en un mismo programa y sobre una misma plataforma.

Una aplicación monolítica es autónoma, e independiente de otras aplicaciones. La filosofía de diseño es que la aplicación es responsable no sólo de una única tarea, si no que es capaz de realizar todos los pasos o tareas necesarias para completar una determinada función. Hoy en día, algunas aplicaciones de finanzas son monolíticas en el sentido que ayudan al usuario a realizar una tarea por completo, y son "silos de datos" privados en lugar de partes de un sistema de aplicaciones que trabajan conjuntamente. Algunos procesadores de texto son aplicaciones monolíticas. Estas aplicaciones a veces son asociadas con unidades centrales.

En su uso original, el término "monolítico" hacía referencia grandes aplicaciones que no podían ser usadas de forma modular. Esto junto con el rápido aumento en capacidad de cómputo y por tanto el rápido aumento en la complejidad de los problemas que podían ser resueltos mediante software dieron como resultado sistemas inmantenibles y la "crisis del software".

1.1.2 Arquitectura Cliente - Servidor



La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

Algunos ejemplos de aplicaciones que usen el modelo cliente-servidor son el Correo electrónico, un Servidor de impresión y la World Wide Web.

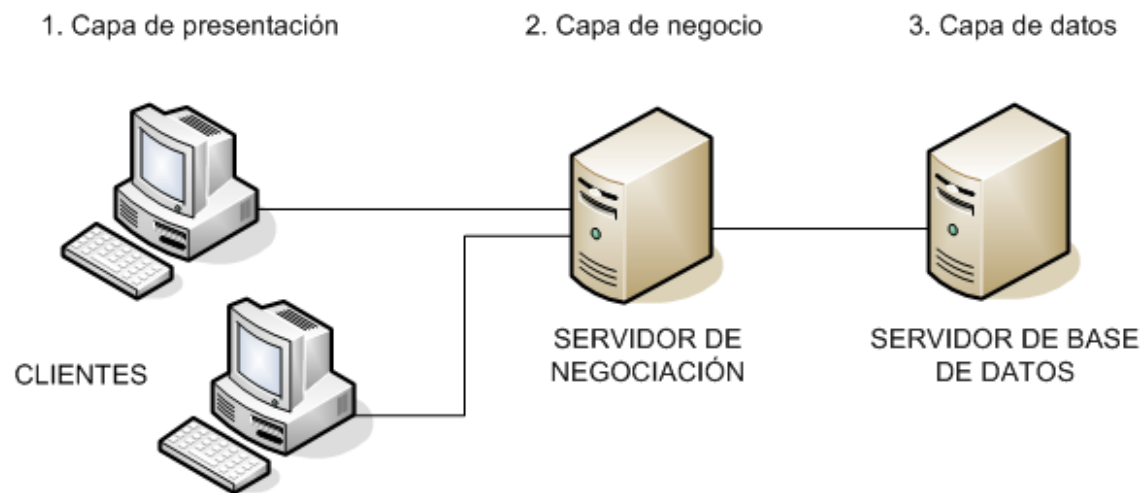
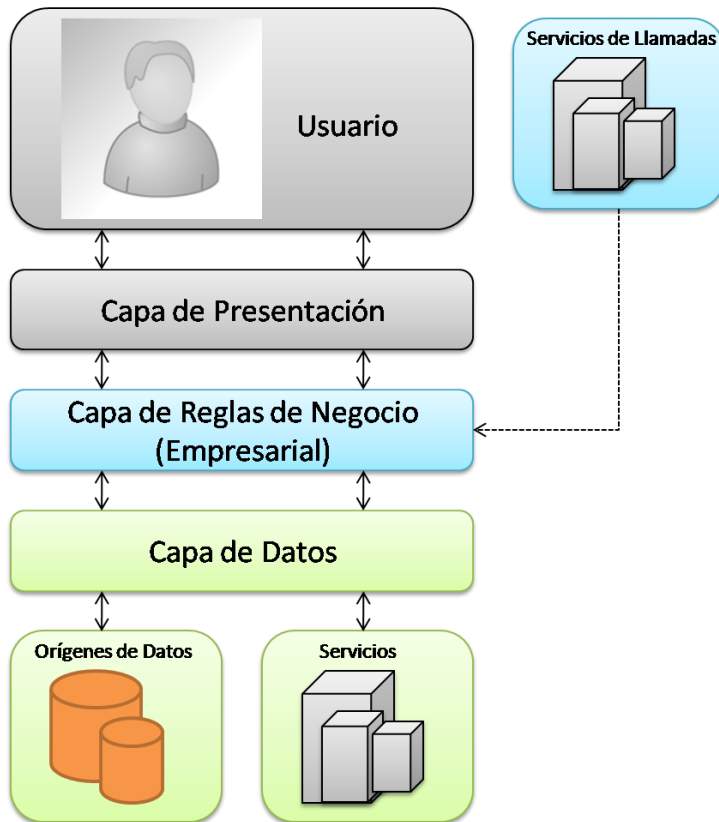
En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un solo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

La red cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta; y que los pone a disposición de los clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se realizan se concentran en el servidor, de manera que en él se disponen los requerimientos provenientes de los clientes que tienen prioridad, los archivos que son de uso público y los que son de uso restringido, los archivos que son de sólo lectura y los que, por el contrario, pueden ser modificados, etc. Este tipo de red puede utilizarse conjuntamente en caso de que se esté utilizando en una red mixta.

1.1.3 Arquitectura en capas



La arquitectura por capas es un modelo de desarrollo software en el que el objetivo primordial es la separación (desacoplamiento) de las partes que componen un sistema software o también una arquitectura cliente-servidor: lógica de negocios, capa de presentación y capa de datos. De esta forma, por ejemplo,

es sencillo y mantenible crear diferentes interfaces sobre un mismo sistema sin requerirse cambio alguno en la capa de datos o lógica.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo afectará al nivel requerido sin tener que revisar entre el código fuente de otros módulos, dado que se habrá reducido el Acoplamiento informático hasta una interfaz de paso de mensajes.

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suelen usar las arquitecturas multinivel o programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

El más utilizado actualmente es el diseño en tres niveles (o en tres capas).

Capas y niveles

Capa de presentación: la que ve el usuario (también se la denomina «capa de usuario»), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser «amigable» (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.

Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

Capa de datos: es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Todas estas capas pueden residir en un único ordenador, si bien lo más usual es que haya una multitud de ordenadores en donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las

necesidades lo aconseja se pueden separar en dos o más ordenadores. Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.

Si, por el contrario, fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos se llega a tener una serie de ordenadores sobre los cuales corre la capa de negocio, y otra serie de ordenadores sobre los cuales corre la base de datos.

En una arquitectura de tres niveles, los términos «capas» y «niveles» no significan lo mismo ni son similares.

El término «capa» hace referencia a la forma como una solución es segmentada desde el punto de vista lógico:

Presentación. (Conocida como capa Web en aplicaciones Web o como capa de usuario en Aplicaciones Nativas)

Lógica de Negocio. (Conocida como capa Aplicativa)

Datos. (Conocida como capa de Base de Datos)

En cambio, el término «nivel» corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física. Por ejemplo:

Una solución de tres capas (presentación, lógica del negocio, datos) que residen en un solo ordenador (Presentación+lógica+datos). Se dice que la arquitectura de la solución es de tres capas y un nivel.

Una solución de tres capas (presentación, lógica del negocio, datos) que residen en dos ordenadores (Presentación+lógica por un lado; lógica+datos por el otro lado). Se dice que la arquitectura de la solución es de tres capas y dos niveles.

1.1.4 Configuración de Solución en 4 Capas

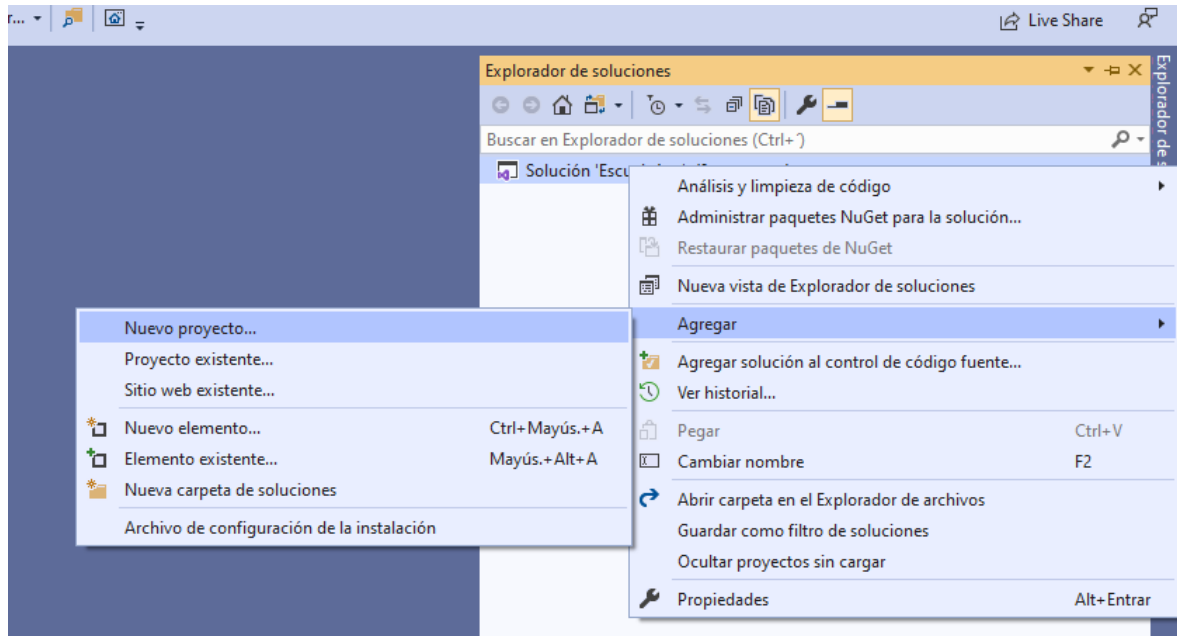
Después de crear solución, puede usar el Explorador de para agregar los proyectos:

Crear ProyectosBiblioteca de Clases

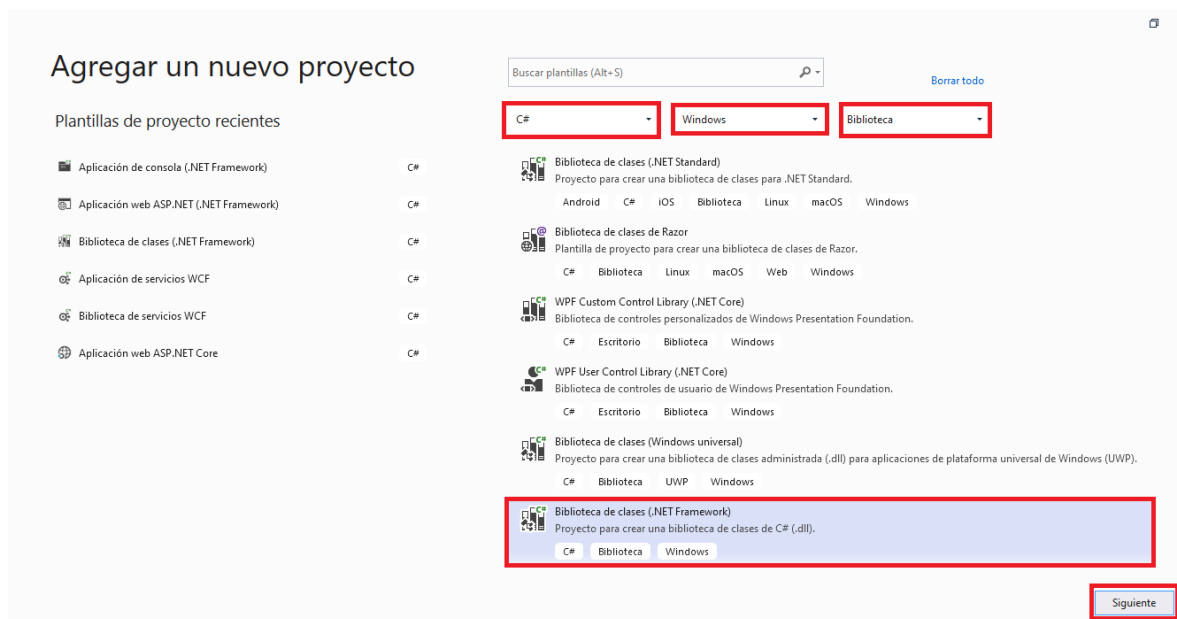
Datos
Negocio
Entidades

Bajo el siguientes procedimiento

Seleccionar la solución, dar click derecho, seleccionar agregar y finalmente Nuevo Proyecto...

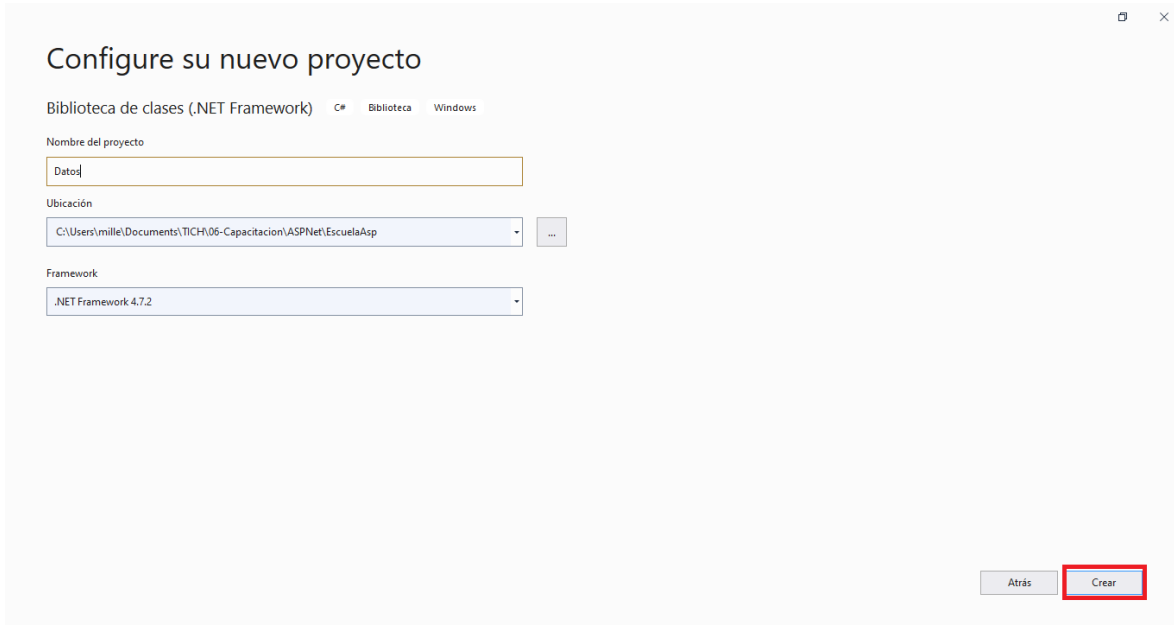


Hecho lo anterior, aparecerá la siguiente Pantalla



Para estos proyectos utilizaremos la plantilla Biblioteca de clases (.Net Framework), para ello, en los filtros seleccionar el lenguaje con el cual

trabajaremos (c#), La plataforma(sistema operativo en nuestro caso Windows), y el tipo de proyecto Biblioteca



Configure su nuevo proyecto

Biblioteca de clases (.NET Framework) C# Biblioteca Windows

Nombre del proyecto

Datos

Ubicación

C:\Users\mille\Documents\TICH\06-Capacitacion\ASPNet\EscuelaAsp

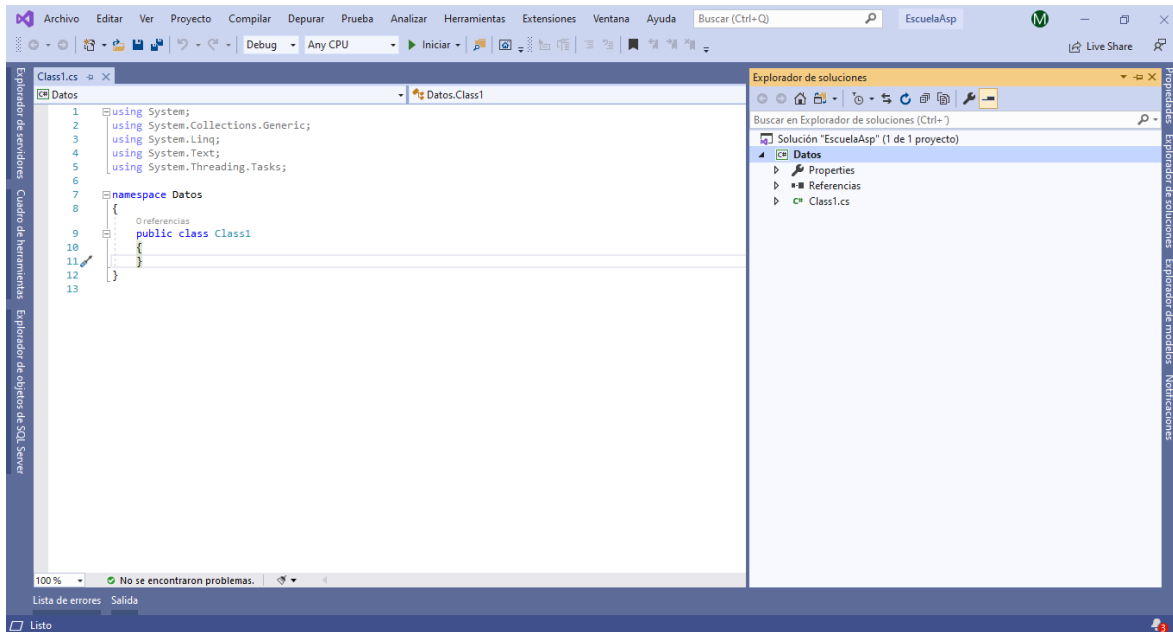
Framework

.NET Framework 4.7.2

Atrás Crear

La página **Configurar el nuevo proyecto** incluye opciones para asignar nombre al proyecto (y la solución), elegir una ubicación de disco y seleccionar una versión de Framework (si se aplica a la plantilla elegida), cuando esto suceda utilice la misma versión del framework para todos los proyectos dentro de la misma solución.

Al oprimir crear, en breve, debería ver algo como lo siguiente:

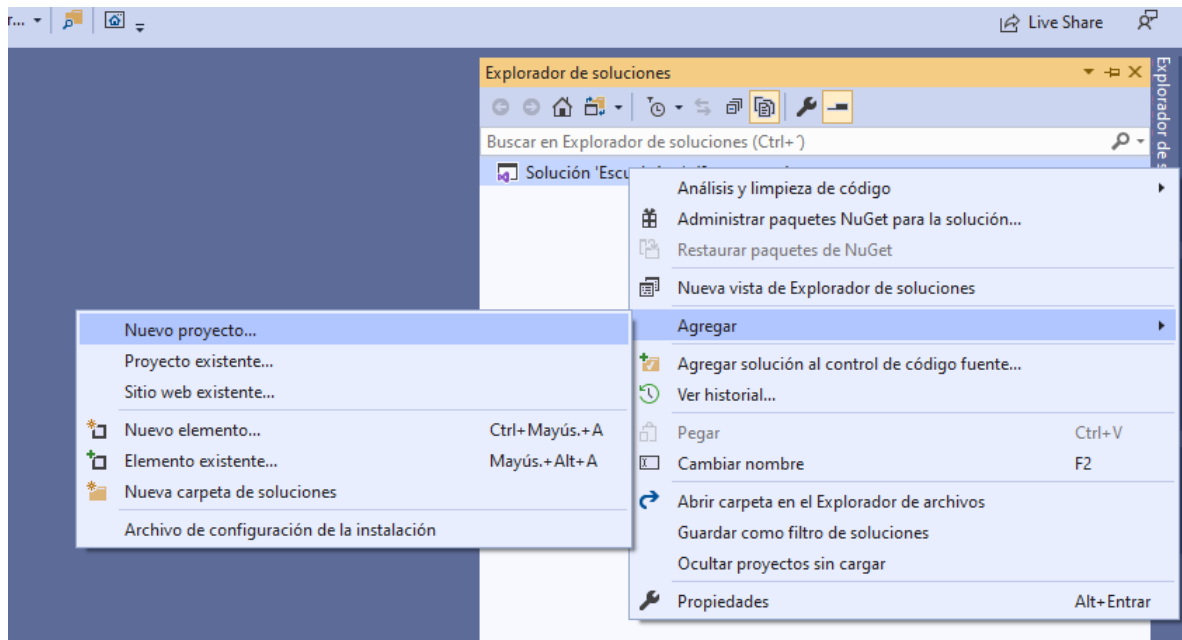


El código C # de su aplicación se muestra en la ventana del editor, que ocupa la mayor parte del espacio. Observe que el texto se colorea automáticamente para indicar diferentes partes del código, como palabras clave y tipos. Además, las líneas discontinuas verticales pequeñas en el código indican qué llaves coinciden, y los números de línea lo ayudan a ubicar el código más adelante. Puede elegir los pequeños signos menos en caja para contraer o expandir bloques de código. Esta función de descripción de código le permite ocultar el código que no necesita, lo que ayuda a minimizar el desorden en pantalla. Los archivos del proyecto se enumeran en el lado derecho en una ventana llamada Explorador de soluciones.

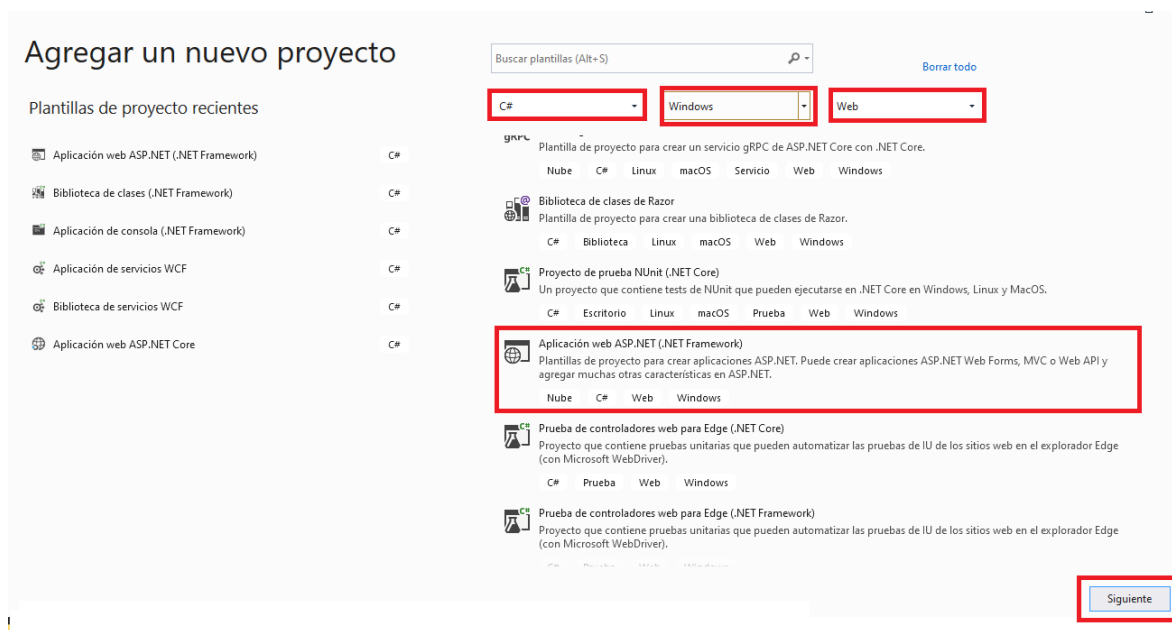
Crear Proyecto Asp .Net

En este caso, el proyecto que hará las funciones de la capa de presentación es un Proyecto con Plantilla “Aplicación Web Asp .Net (.Net Framework), en especial con Web Forms. ASP .Net Web Forms permite crear sitios web dinámicos con un modelo familiar controlado por eventos para arrastrar y colocar. Una superficie de diseño y cientos de controles y componentes permiten crear rápidamente sofisticados y eficaces sitios controlados por la interfaz del usuario y con acceso a datos.

Seleccionar la solución, dar click derecho, seleccionar agregar y finalmente Nuevo Proyecto...

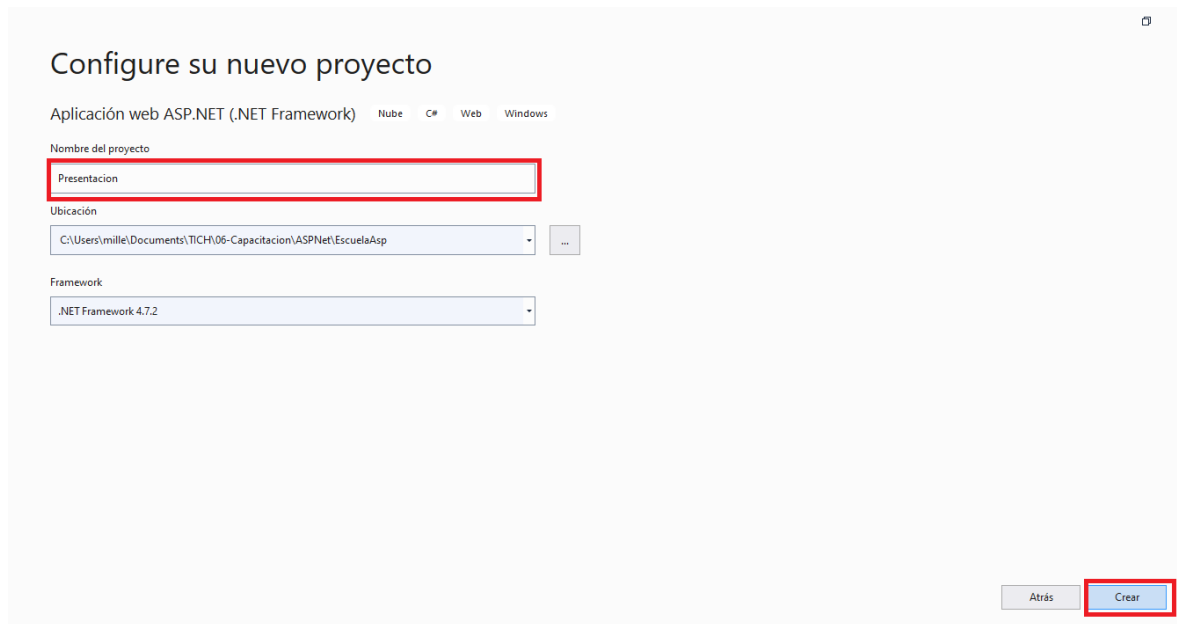


Con lo cual se mostrará la siguiente pantalla:



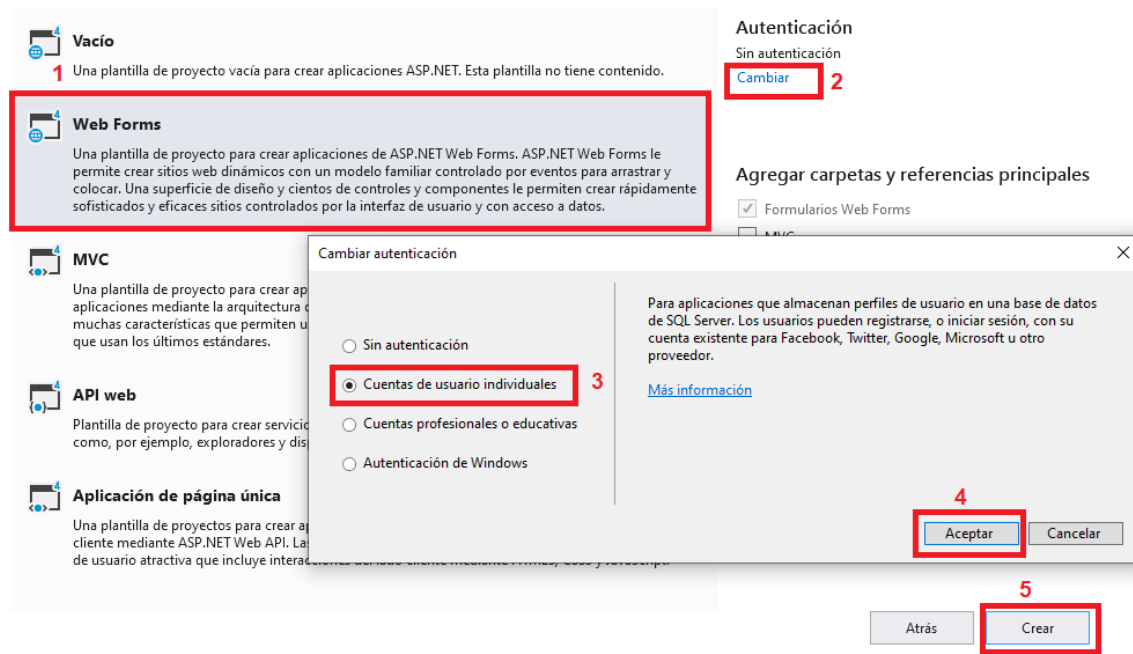
En los filtros, seleccionar el lenguaje **C#**, plataforma **Windows** (Sistema Operativo), Tipo de Proyecto **Web**

Seleccione la plantilla “Aplicación web ASP .NET(.Net Framework), y oprimir siguiente.



Configure el Proyecto, asignándole el nombre del Proyecto (en este caso le daremos el nombre de Presentación), la ubicación, y el framework, elija el mismo framework que se eligió para los demás proyectos de la solución. Y oprima **Crear**.

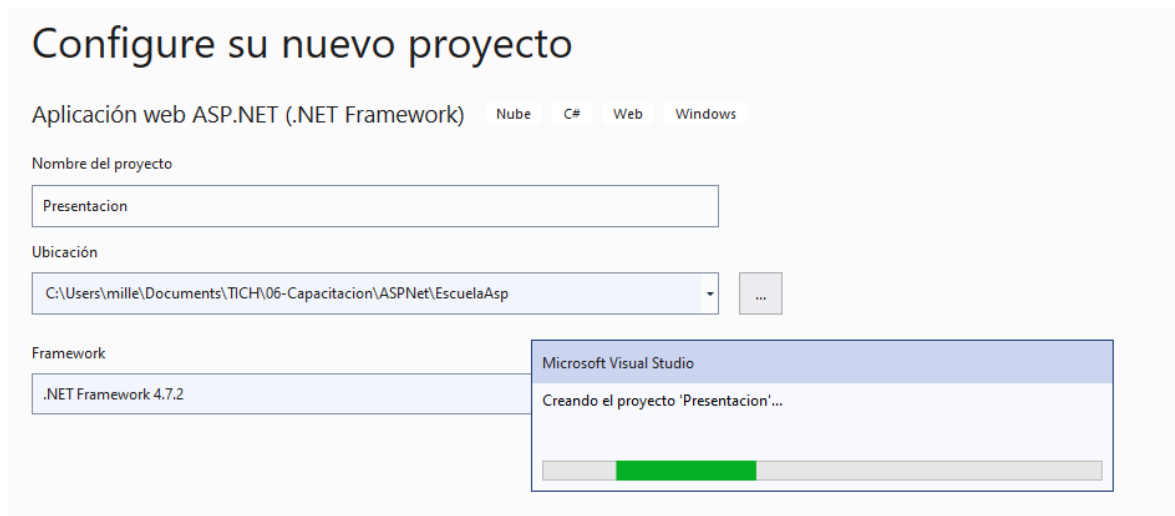
Crear una aplicación web ASP.NET



Para este caso, seleccionamos la opción de Web Forms.

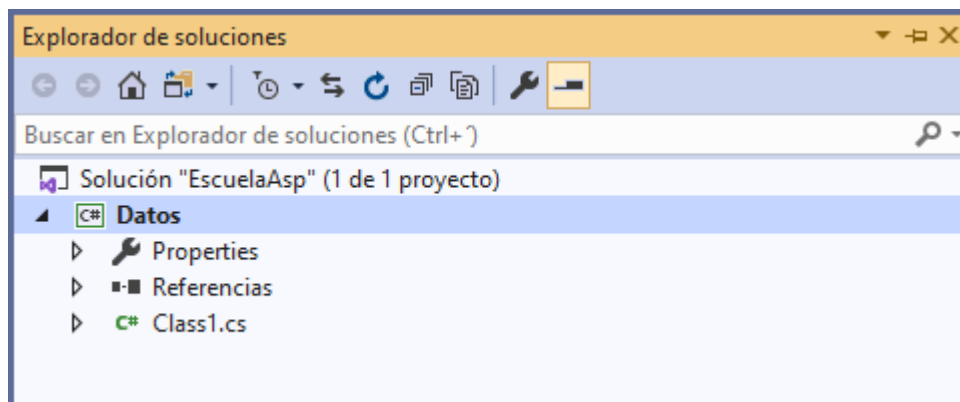
Este tipo de plantilla tiene la opción de crearla con diferentes opciones de autenticación, es decir con o sin la posibilidad de acceder a la aplicación mediante una forma de autenticarse (usuario y contraseña). Para nuestro caso, seleccionamos “Cuentas de usuario individuales” la cual nos proporcionará una plantilla incluyendo toda una infraestructura para crear usuarios que puedan acceder a la aplicación.

Para realizar lo anterior oprimir “Cambiar” y en la pantalla que nos aparece, seleccionar “Cuentas de usuario individuales” y aceptar, para finalmente oprimir “Crear”.



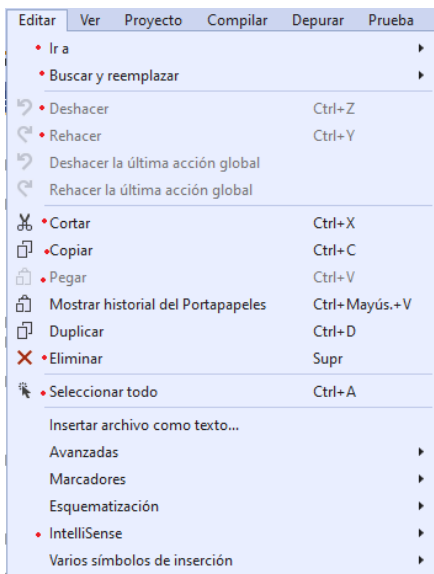
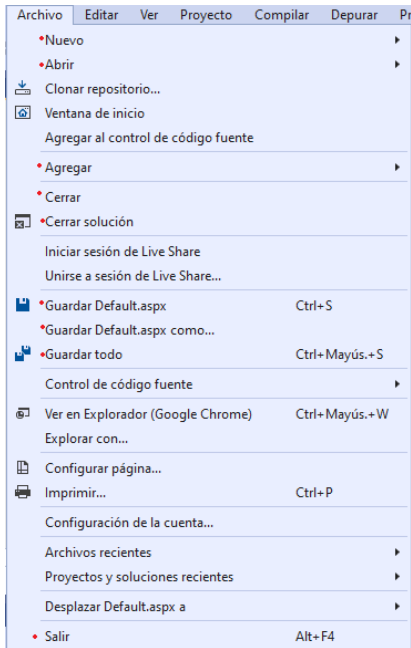
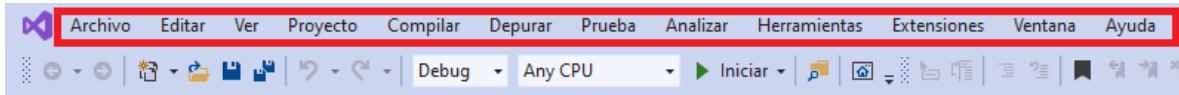
Explorador de la solución

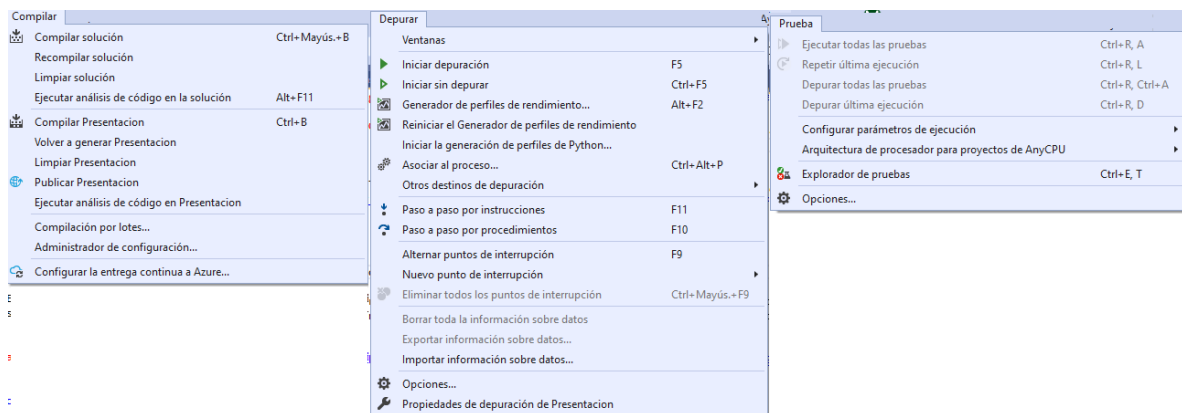
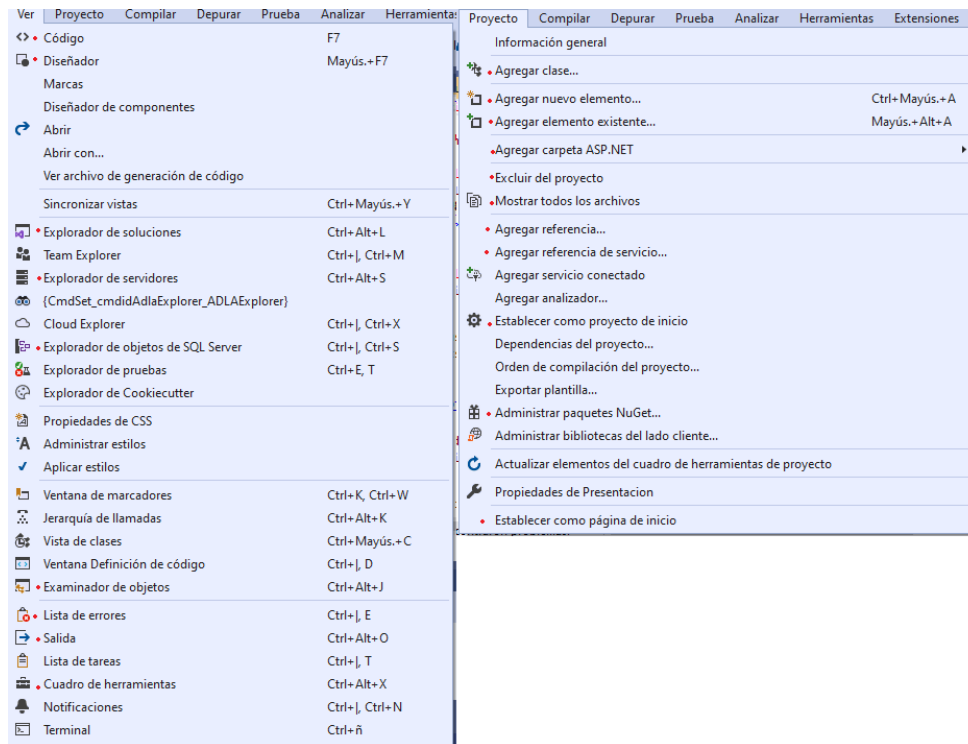
El Explorador de soluciones, que normalmente se encuentra en el lado derecho de Visual Studio, le muestra una representación gráfica de la jerarquía de archivos y carpetas en su proyecto, solución o carpeta de código. Puede examinar la jerarquía y navegar hasta un archivo en el Explorador de soluciones.



Menús

La barra de menú en la parte superior de Visual Studio agrupa los comandos en categorías. Por ejemplo, el menú Proyecto contiene comandos relacionados con el proyecto en el que está trabajando. En el menú Herramientas, puede personalizar cómo se comporta Visual Studio seleccionando Opciones o agregar características a su instalación seleccionando Obtener herramientas y características.

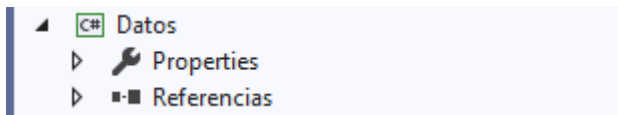




1.2 Estructura de una Biblioteca de Clases

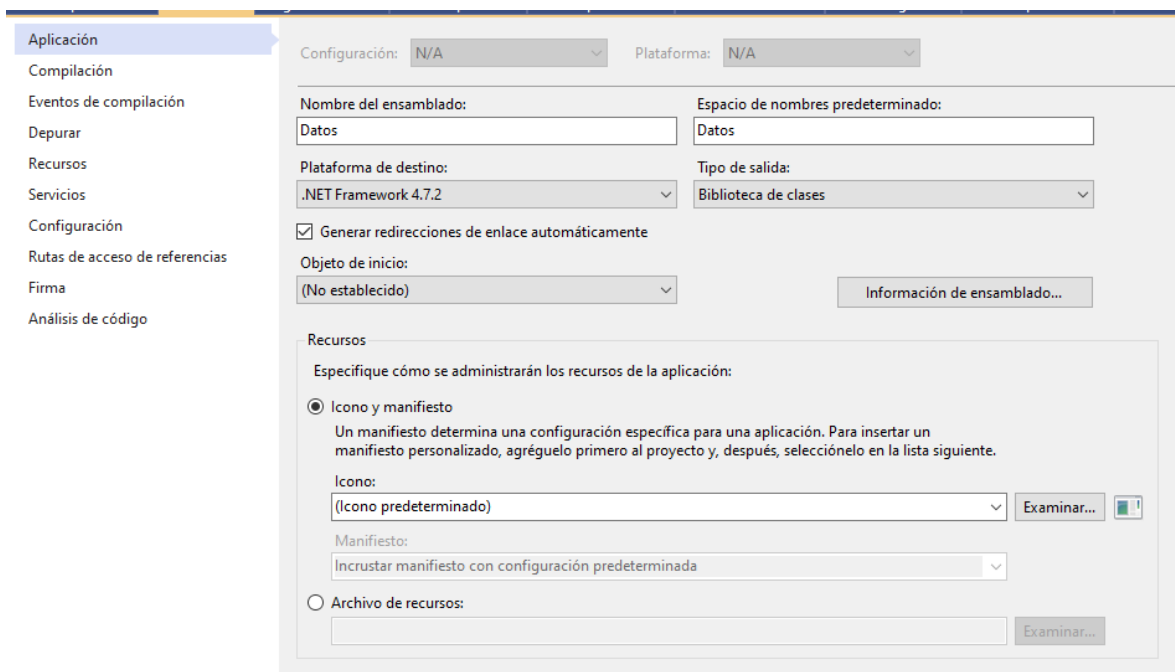
La plantilla **Biblioteca de clases** se utiliza para crear clases y componentes reutilizables que pueden compartirse con otros proyectos. La Plantilla creada es muy sencilla, ya que solo consta con su archivo de proyecto y sus referencias, como se pueden ver en el Explorador de soluciones.

1.2.1 Propiedades de un Proyecto



Los proyectos tienen propiedades que controlan muchos aspectos de la compilación, la depuración, las pruebas y la implementación. Algunas propiedades están presentes en todos los tipos de proyecto, mientras que otras son exclusivas de plataformas o idiomas específicos. Para acceder a las propiedades del proyecto, seleccione el nodo de propiedades dependiente del proyecto y dé doble click, o bien haga clic con el botón derecho en el nodo del proyecto en el Explorador de soluciones y elija Propiedades o escriba propiedades en el cuadro de búsqueda de la barra de menús y elija Ventana de propiedades en los resultados.

En los proyectos de C#, Visual Basic y F#, las propiedades se exponen en el Diseñador de proyectos. En la siguiente ilustración se muestra la página Propiedad de un Proyecto Biblioteca de clases



En la cejilla de Aplicación podemos observar varios parámetros:

Nombre del ensamblado: Por default el nombre del ensamblado es el nombre del Proyecto, aunque éste puede cambiarse. Cuando se compila un proyecto biblioteca de clase, se generará una dll con el nombre proporcionado en esta casilla.

Espacio de nombres predeterminado: Por default el Espacio de nombres predeterminado es el nombre del Proyecto, aunque éste puede cambiarse. Esto significa que todas las clases que se creen bajo el Proyecto se les creará bajo el espacio de nombres predeterminado indicado en esta casilla.

Los espacios de nombres organizan los objetos definidos en un ensamblado. Los ensamblados pueden contener varios espacios de nombres, que a su vez pueden contener otros espacios de nombres. Los espacios de nombres evitan las ambigüedades y simplifican las referencias cuando se usan grupos de objetos grandes, como las bibliotecas de clases.

Por ejemplo, el .NET Framework define la clase `ListBox` en el `System.Windows.Forms` espacio de nombres. En el siguiente fragmento de código se muestra cómo declarar una variable con el nombre completo de esta clase:

`System.Web.UI.WebControls.TextBox` txtWebNombre

En este ejemplo, tenemos toda una estructura en el espacio de nombres empezando por el de más alto nivel `System` el cual contiene uno llamado `Web`, y así sucesivamente hasta llegar al nombre de la clase `TextBox`, de tal forma que la clase `TextBox` pertenece al espacio de nombres `System.Web.UI.WebControls`

Con el uso de los espacios de nombres podemos evitar confusiones ya que la misma clase `TextBox` existe para un Proyecto de aplicación `Windows`

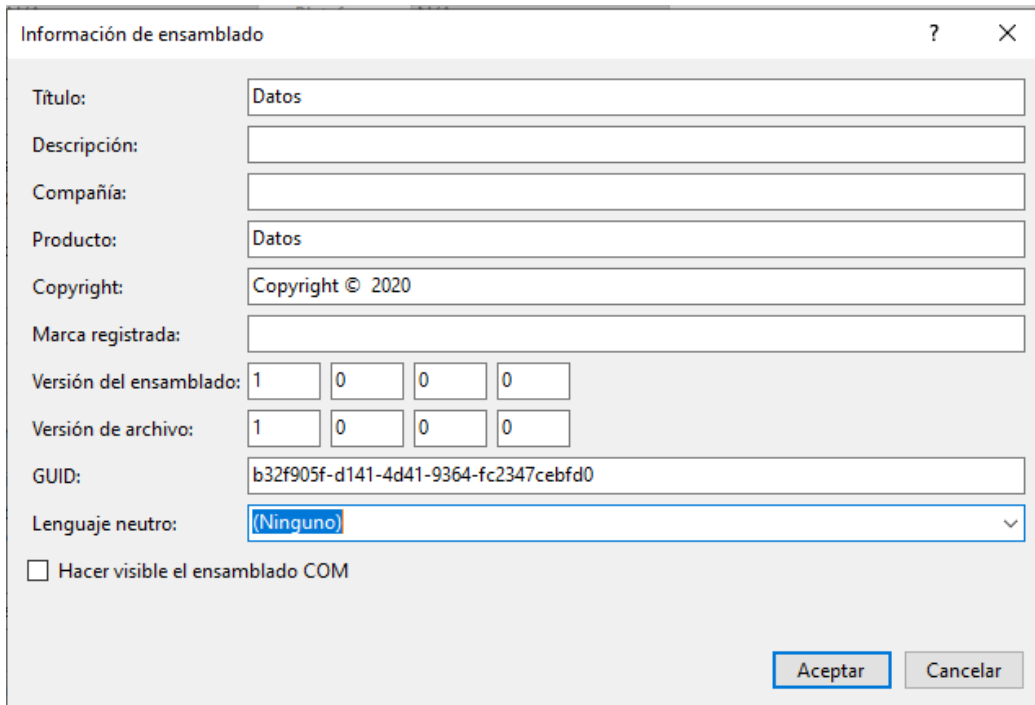
`System.Windows.Forms.TextBox` txtWinNombre

Plataforma de destino

En esta casilla también podemos encontrar y en su caso modificar la plataforma bajo la cual se creará el proyecto es decir, la versión del framework que se estará utilizando.

Información de ensamblado

Con la opción de información del ensamblado podemos documentar nuestro proyecto, sobre todo podemos ir registrando, para una mejor administración, la versión de nuestro ensamblado.

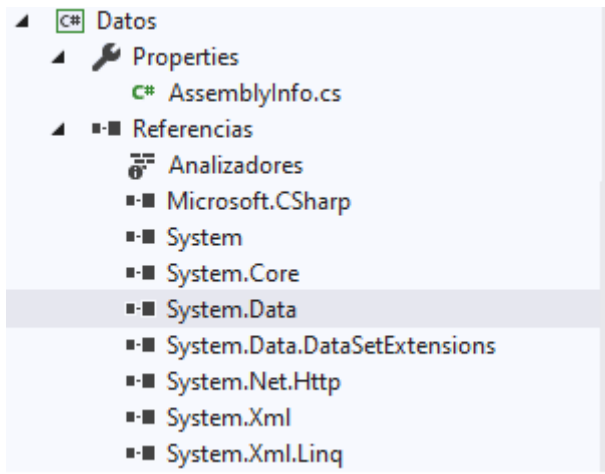
The screenshot shows the 'Información de ensamblado' (Assembly Information) dialog box. It contains several text boxes for metadata: 'Título' (Title) with 'Datos', 'Descripción' (Description) empty, 'Compañía' (Company) empty, 'Producto' (Product) with 'Datos', 'Copyright' with 'Copyright © 2020', 'Marca registrada' (Registered trademark) empty, 'Versión del ensamblado' (Assembly version) with four input boxes containing '1', '0', '0', '0', 'Versión de archivo' (File version) with four input boxes containing '1', '0', '0', '0', 'GUID' with 'b32f905f-d141-4d41-9364-fc2347cebfd0', and 'Lenguaje neutro' (Neutral language) with a dropdown menu showing '(Ninguno)'. At the bottom, there is a checkbox 'Hacer visible el ensamblado COM' (Make assembly COM visible) which is unchecked, and two buttons 'Aceptar' (OK) and 'Cancelar' (Cancel).

1.2.2 Referencias

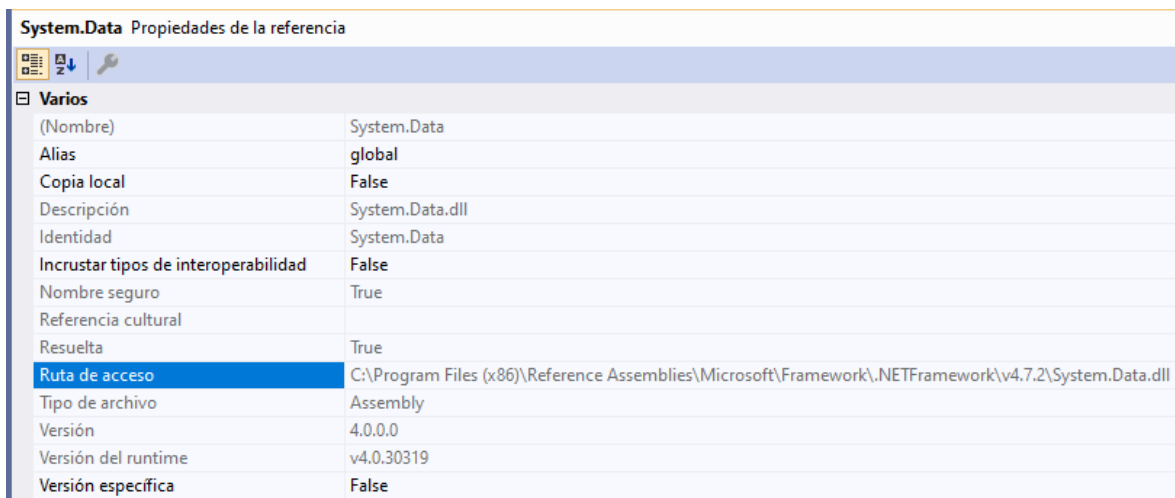
Una referencia es básicamente una entrada de un archivo de proyecto que contiene la información que Visual Studio necesita para localizar el componente o el servicio.

Antes de escribir código en un componente externo o en un servicio conectado, el proyecto debe contener primero una referencia a él.

La Plantilla de una Biblioteca de clases crea automáticamente las referencias a los componentes más comunes que se utilizan cuando se crea una biblioteca de clases.



Como se puede ver en las propiedades de una referencia, entre otras cosas nos indica la ruta del archivo de ensamblado donde se encuentra el componente que deseamos utilizar, así como la versión del mismo.



1.3Capa de Entidades

1.3.1 Clases POCO y DTO

La capa de Entidades estará compuesta por clases tipo POCO que entyre otras cosas las utilizaremos como DTO.

Las clases tipo POCO, por las siglas de Plain Old C# Object, se refieren a clases simples que no dependen de ninguna framework. Es un término derivado del concepto del mundo Java: POJO.

Mientras que DTO, se refiere a Data Transfer Objects y es un objeto que por definición se envía y recibe dentro de un servicio.

Nuestras clases POCO serán clases sencillas que constan únicamente del conjunto de propiedades del objeto que queremos representar.

1.3.2 Propiedades

Una propiedad es un miembro que proporciona un mecanismo flexible para leer, escribir o calcular el valor de un campo privado. Las propiedades se pueden usar como si fueran miembros de datos públicos, pero en realidad son métodos especiales denominados descriptores de acceso. Esto permite acceder fácilmente a los datos a la vez que proporciona la seguridad y la flexibilidad de los métodos.

Información general sobre propiedades

- Las propiedades permiten que una clase exponga una manera pública de obtener y establecer valores, a la vez que se oculta el código de implementación o verificación.
- Para devolver el valor de la propiedad se usa un descriptor de acceso de propiedad get, mientras que para asignar un nuevo valor se emplea un descriptor de acceso de propiedad set. Estos descriptores de acceso pueden tener diferentes niveles de acceso. Para más información, vea Restringir la accesibilidad del descriptor de acceso.
- La palabra clave value se usa para definir el valor que va a asignar el descriptor de acceso set.
- Las propiedades pueden ser de lectura y escritura (en ambos casos tienen un descriptor de acceso get y set), de solo lectura (tienen un descriptor de acceso get, pero no set) o de solo escritura (tienen un descriptor de acceso set, pero no get). Las propiedades de solo escritura son poco frecuentes y se suelen usar para restringir el acceso a datos confidenciales.
- Las propiedades simples que no necesitan ningún código de descriptor de acceso personalizado se pueden implementar como definiciones de cuerpos de expresión o como propiedades implementadas automáticamente.

Propiedades con campos de respaldo

Un patrón básico para implementar una propiedad conlleva el uso de un campo de respaldo privado para establecer y recuperar el valor de la propiedad. El descriptor de acceso get devuelve el valor del campo privado y el descriptor de acceso set puede realizar alguna validación de datos antes de asignar un valor al campo privado. Ambos descriptores de acceso además pueden realizar algún cálculo o conversión en los datos antes de que se almacenen o se devuelvan.

En el ejemplo siguiente se muestra este patrón. En este ejemplo, la clase TimePeriod representa un intervalo de tiempo. Internamente, la clase almacena el intervalo de tiempo en segundos en un campo privado denominado `_seconds`. Una

propiedad de lectura y escritura denominada Hours permite al cliente especificar el intervalo de tiempo en horas. Los descriptores de acceso get y set realizan la conversión necesaria entre horas y segundos. Además, el descriptor de acceso set valida los datos e inicia una excepción ArgumentOutOfRangeException si el número de horas no es válido.

```
using System;

class TimePeriod
{
    private double _seconds;

    public double Hours
    {
        get { return _seconds / 3600; }
        set {
            if (value < 0 || value > 24)
                throw new ArgumentOutOfRangeException(
                    $"{nameof(value)} must be between 0 and 24.");
            _seconds = value * 3600;
        }
    }
}
```

Definiciones de cuerpos de expresión

Los descriptores de acceso de propiedad suelen constar de instrucciones de una sola línea que simplemente asignan o devuelven el resultado de una expresión. Puede implementar estas propiedades como miembros con forma de expresión. Las definiciones de cuerpos de expresión constan del símbolo => seguido de la expresión que se va a asignar a la propiedad o a recuperar de ella.

Las propiedades de solo lectura pueden implementar el descriptor de acceso get como miembro con forma de expresión. En este caso, no se usan ni la palabra clave del descriptor de acceso get ni la palabra clave return. En el ejemplo siguiente se implementa la propiedad de solo lectura Name como miembro con forma de expresión.

```
public class Persona
{
    private string _nombre;
    private string _apellido;

    public Persona(string nombre, string apellido)
    {
        _nombre = nombre;
        _apellido = apellido;
    }

    public string nombreCompleto => $"{_nombre} {_apellido}";
}
```

```
public string nombre
{
    get => _nombre;
    set => _nombre = value;
}
public string apellido
{
    get => _apellido;
    set => _apellido = value;
}
}
```

Propiedades implementadas automáticamente

En algunos casos, los descriptores de acceso de propiedad `get` y `set` simplemente asignan un valor a un campo de respaldo o recuperan un valor de él sin incluir ninguna lógica adicional. Mediante las propiedades implementadas automáticamente, puede simplificar el código y conseguir que el compilador de C# le proporcione el campo de respaldo de forma transparente.

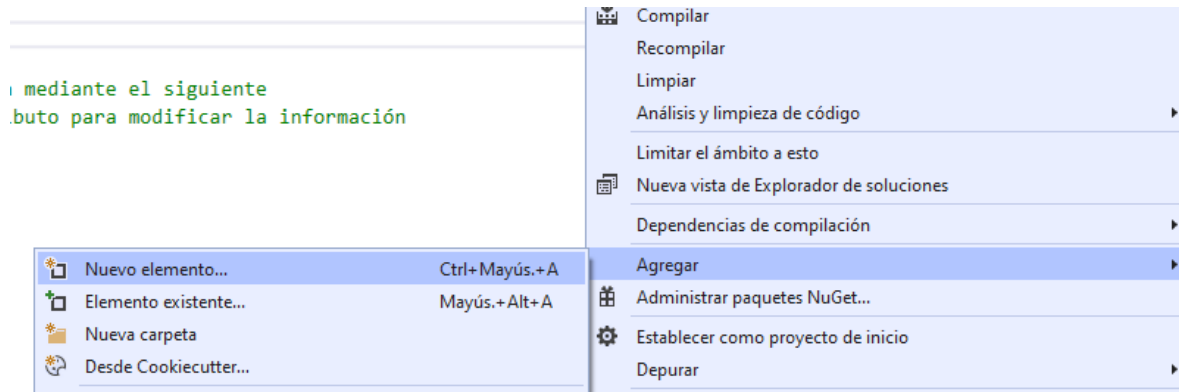
Si una propiedad tiene un descriptor de acceso `get` y `set`, ambos deben ser implementados automáticamente. Una propiedad implementada automáticamente se define mediante las palabras clave `get` y `set` sin proporcionar ninguna implementación. El ejemplo siguiente repite el anterior, salvo que `nombre` y `apellido` son propiedades implementadas automáticamente. Tenga en cuenta que en el ejemplo también se quita el constructor parametrizado, por lo que los objetos `Persona` ahora se inicializan con una llamada al constructor sin parámetros y un inicializador de objeto.

```
Public class Persona
{
    Public string nombre {get;set; }
    Public string apellido {get;set; }
}
```

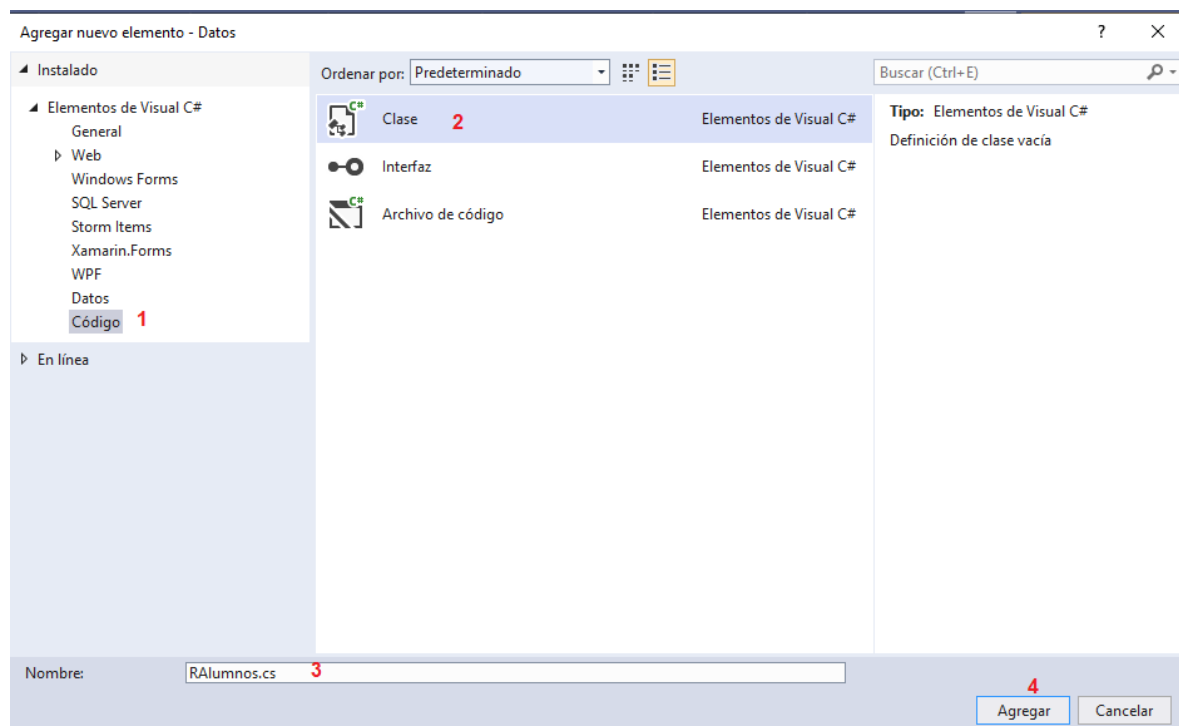
Nuestras clases POCO de la capa de Entidades las definiremos como Propiedades implementadas automáticamente;

1.3.3 Crear Clases

Para crear una clase, seleccionar el nodo del Proyecto o de una carpeta, debajo del cual se quiere crear la nueva clase, y con dar click con el botón derecho y seleccionar “agregar nuevo elemento”.



Seleccionar código, seguido de Clase y proporcionar el nombre, y oprimir Agregar



Con lo anterior se crea la plantilla de una clase.

A partir de ahí agregaremos las propiedades o métodos correspondientes


```
public class Alumno
{
    6 referencias
    public int Id { get; set; }
    8 referencias
    public string Nombre { get; set; }
    8 referencias
    public string PrimerApellido { get; set; }
    8 referencias
    public string SegundoApellido { get; set; }
    8 referencias
    public string Correo { get; set; }
    8 referencias
    public string Telefono { get; set; }
    8 referencias
    public DateTime FechaNacimiento { get; set; }
    8 referencias
    public string Curp { get; set; }
    8 referencias
    public int IdEstadoOrigen { get; set; }
    8 referencias
    public int IdEstatus { get; set; }
}
```

1.4Capa de Datos

Empezaremos a desarrollar nuestra aplicación con la capa de datos, y para ello implementaremos las clases DALumnos, DEstados y DEstatusAlumnos con métodos CRUD utilizando ADO .NET para el accesos a la Base de Datos.

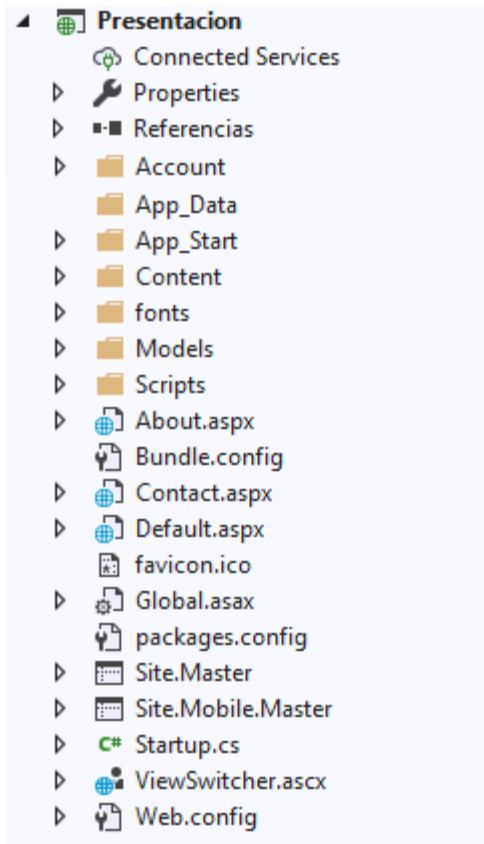
1.5Capa de Negocio

Como se menciona en el apartado [Arquitectura en capas](#) en esta capa residen las clases que reciben las peticiones del usuario provenientes de la capa de Presentación, genéricamente se realizan validaciones, cálculos, en general se aplican las reglas del negocio. Si es necesario almacenar de manera permanente o recuperar de ese medio la información, estas clases llaman a las correspondientes de la capa de datos. Finalmente envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse.

Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él.

1.6Capa de Presentación

1.6.1 Estructura de una Aplicación Web Forms

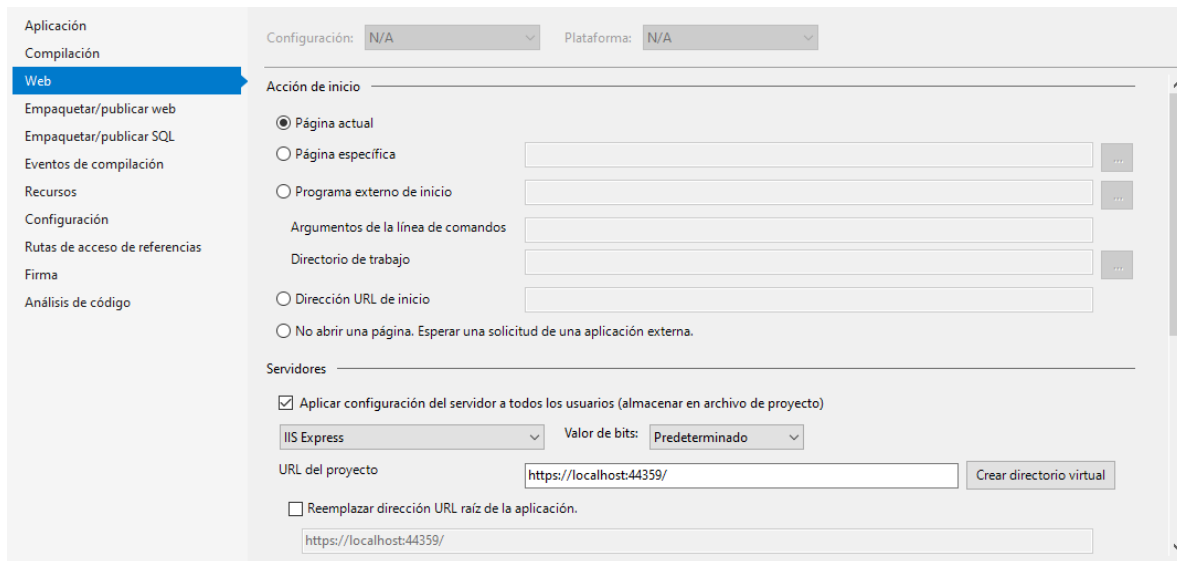


1. Properties

En cuanto a las propiedad del Proyecto, una diferencia importante con respecto a los Proyectos anteriormente vistos, es la pestaña Web.

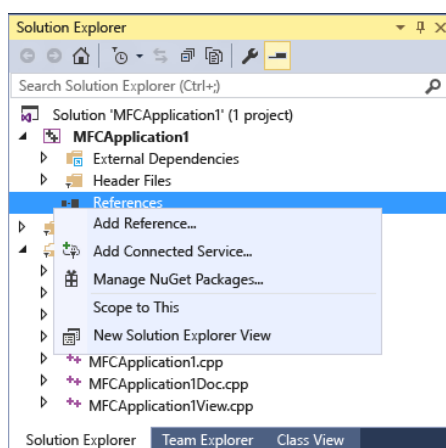
En esta pestaña podemos establecer con cuál página se iniciará nuestra aplicación al momento de ejecutarla:

- Pagina actual .- Indicaré que la página que tenemos en la ventana principal de nuestro IDE será la que se muestre en primera instancia.
- Página específica.- Indicaré que la página que será siempre la que se muestre en primera instancia.
- En momento de desarrollo se recomienda tenerla en Página Actual, para tener flexibilidad de ir construyendo y probando cada una de las páginas que compondrán nuestra aplicación; pero cuando se pase a producción deberá establecerse la página inicial de la aplicación.
-



2- References.

Al igual que eAntes de escribir código en un componente externo o en un servicio conectado, el proyecto debe contener primero una referencia a él. Una referencia es básicamente una entrada de un archivo de proyecto que contiene la información que Visual Studio necesita para localizar el componente o el servicio. Para agregar una referencia, haga clic con el botón derecho en el nodo **Referencias** o **Dependencias** del **Explorador de soluciones** y elija **Agregar referencia**. También puede hacer clic con el botón derecho en el nodo del proyecto y seleccionar **Agregar > Referencia**. Para obtener más información, vea [Adición o eliminación de referencias](#).



Puede agregar una referencia a los siguientes tipos de componentes y servicios:

- Bibliotecas de clases o ensamblados de .NET Framework
- Aplicaciones para UWP

- componentes COM
- Otros ensamblados o bibliotecas de clases de proyectos de la misma solución
- servicios Web XML
-

3- BundleConfig.cs

Un Bundle es una agrupación de enlaces a archivos externos desde el HTML. Es algo habitual. Para evitar una tarea tan repetitiva se han creado estos Bundles, que son accesos directos a agrupaciones de este tipo de archivos. Por ejemplo, este conjunto de llamadas a scripts de javascript:

```
<scriptsrc="~/Scripts/jquery-2.1.1.min.js"></script>  
<scriptsrc="~/Scripts/jquery-ui-1.11.1.js"></script>  
<scriptsrc="~/Scripts/jquery.validate.min.js"></script>  
<scriptsrc="~/Scripts/jquery.validate.unobtrusive.min.js"></script>
```

Puede simplificarse a un Bundle como este:

```
<%: Scripts.Render("~/bundles/modernizr") %>
```

La ventaja simplificando el código es clara. En este caso solo hay cuatro scripts, pero podría haber docenas. Además, no solo se pueden agrupar scripts, sino también Hojas de estilo.

¿Dónde se configuran?

Puedes crear muchos Bundles, conteniendo numerosos scripts u hojas de estilo. Todos ellos se construyen dentro del fichero **BundleConfig.cs** en el **App_Start**.

4- RouteConfig.cs

Primeramente es necesario explicar lo que es el enrutamiento.

El enrutamiento permite a la aplicación web usar direcciones URL que son fáciles de recordar y que son más compatibles con los motores de búsqueda.

El enrutamiento de direcciones URL permite configurar una aplicación para que acepte direcciones URL de solicitud que no se asignan a archivos físicos. Una dirección URL de solicitud es simplemente la dirección URL que un usuario escribe en el explorador para buscar una página en el sitio Web. El enrutamiento se usa para definir direcciones URL semánticamente significativas para los usuarios y que pueden ayudar con la optimización del motor de búsqueda (SEO).

De forma predeterminada, la plantilla de formularios Web Forms incluye direcciones URL descriptivas de ASP.net. Gran parte del trabajo de enrutamiento básico se implementará mediante el uso de direcciones URL descriptivas, donde prácticamente la ruta de la página es la dirección física de la página o recurso, como sigue:

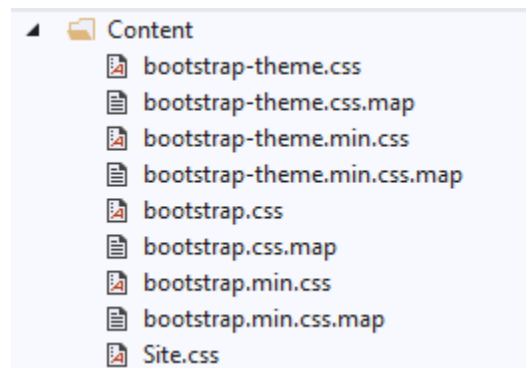
<https://localhost:44300/Alumnos.aspx?IdAlumno=2>

Rutas

Una ruta es un patrón de dirección URL que se asigna a un controlador. El controlador puede ser un archivo físico, como un archivo .aspx, en una aplicación de formularios Web Forms. Un controlador también puede ser una clase que procesa la solicitud. Para definir una ruta, cree una instancia de la clase Route especificando el patrón de la dirección URL, el controlador y, opcionalmente, un nombre para la ruta.

5- Content.

El directorio Content está pensado para el contenido estático de la aplicación, especialmente útil para archivos css e imágenes asociadas.



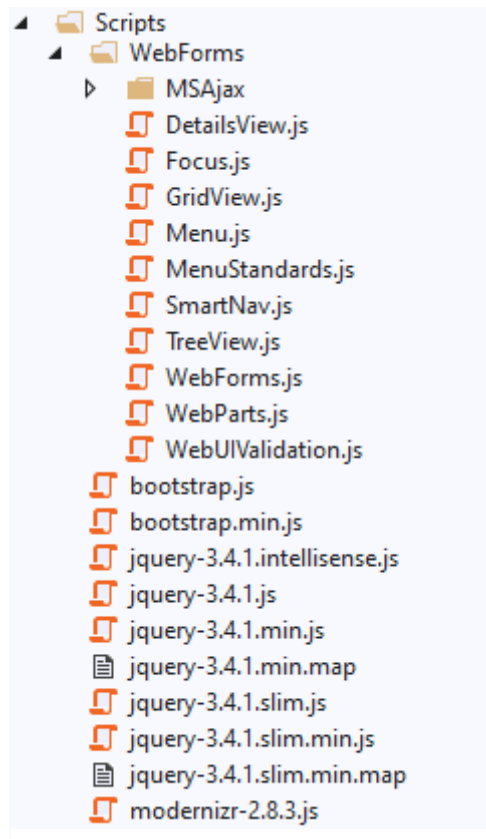
ASP.NET Web Form nos ofrece por defecto las clases definidas en bootstrap

6- Fonts

El directorio fonts está pensado para ubicar los archivos utilizados para las fuentes e iconos que utilizarías en una aplicación, por default la plantilla viene cargada con los fonts de bootstrap.

7- Scripts

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



ASP.NET Web Forms incluye varias librerías de javascript por defecto:

- **jquery.js.** Esta popular y súper útil librería nos va a permitir gestionar fácilmente peticiones [AJAX](#), manipular el DOM en cliente, etc ... Esta librería se ha convertido en un framework muy popular, y es la base de para la creación de [pluggins](#) que nos van a permitir dotar a nuestro sitio web de efecto sorprendentes sin apenas esfuerzo. <http://jquery.com/>
- **modernizr.js.** Esta librería nos permite validar fácilmente si el navegador que esta ejecutando la página web es compatible con HTML5, en en caso de que no sea así proporcionar un mecanismo alternativo (*polyfill*). <http://modernizr.com/>. Por ejemplo, si nuestro navegador es compatible con HTML5 interpretará sin problema la etiqueta *VIDEO*, pero en el caso de que estemos navegando con un navegador antiguo deberemos ofrecer al usuario un mecanismo alternativo para el video (un flash por ejemplo).

Nota: El nombre de los archivos puede varias dependiendo de la versión instalada, por ejemplo el archivo jQuery.js se llama jquery-1.7.1.js, ya que corresponde con la versión 1.7.1 de la librería.

8- Web.Config

El archivo web.config es el archivo principal de configuración de ASP.NET. Se trata de un archivo XML donde se define la configuración de la aplicación. Veremos poco a poco el

contenido de este fichero, aunque vamos a ver aquí algunas características generales que es necesario conocer.



Para simplificar el despliegue de las aplicaciones, el archivo de configuración se presenta en diferentes versiones o “sabores”, de forma que podemos modificar el archivo de configuración dependiendo de nuestra configuración de despliegue. Si observamos estos archivos observaremos que contienen transformaciones XML que se aplican sobre el archivo web.config al desplegar la aplicación.

De este modo cuando desplegamos la aplicación en modo *Debug* se aplicarán las transformaciones XML definidas en *Web.Debug.config*.

El archivo de configuración aplica un mecanismo de jerarquía a nivel de los archivos de configuración, en la que un archivo de mayor profundidad dentro de la jerarquía sobrescribe al de menor, en el contexto del recurso solicitado.

9- Favicon.ico

Archivo que representa el icono de la aplicación web, es agregado automáticamente al crear el Proyecto plantilla de Visual Studio.

10- Global.asax

Toda aplicación ASP.NET Web Form es una instancia de una clase derivada de *System.Web.HttpApplication*. Esta clase es el punto de entrada de nuestra aplicación – el *Main* de la aplicación web por decirlo de alguna manera.

11-packages.config

El *packages.config* archivo se usa en algunos tipos de proyectos para mantener la lista de paquetes referenciados por el proyecto. Esto permite que NuGet restaure fácilmente las dependencias del proyecto cuando el proyecto se transporte a una máquina diferente, como un servidor de compilación, sin todos esos paquetes.

Si se usa, *packages.config* normalmente se encuentra en una raíz de proyecto. Se crea automáticamente cuando se ejecuta la primera operación NuGet, pero también puede crearse manualmente antes de ejecutar cualquier comando como *nuget restore*.