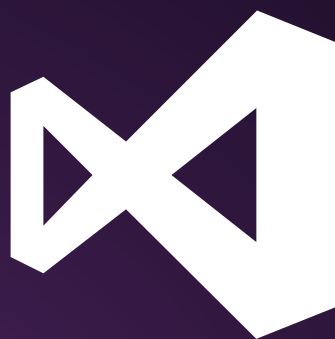


# DESARROLLO DE APLICACIONES WINDOWS EN **C#** USANDO



Visual Studio



## Estructura de contenidos

	Pág.
Introducción .....	3
Mapa de contenido .....	4
1. Proceso de instalación de Visual Studio .Net.....	5
2. Características principales de Visual Studio .Net.....	8
3. Arquitectura de tres capas.....	11
4. Acceso a datos .....	13
5. Controles principales de C# .....	16
5.1 Controles generales principales .....	16
5.2 Manejo de propiedades y eventos de los controles .....	18
6. Detalles del lenguaje C#.....	19
6.1 Variables.....	19
6.2 Constantes .....	20
6.3 Conversiones.....	21
6.4 Comentarios .....	21
6.5 Operadores principales .....	22
7. Creación de aplicaciones en Windows Forms.....	23
7.1 Arquitectura de la aplicación.....	26
8. Creación de instalador en aplicación Windows Forms.....	33
Glosario .....	39
Bibliografía.....	40
Control del documento .....	41

## Introducción



Microsoft Visual Studio .Net es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones de escritorio, aplicaciones web ASP.NET, Servicios Web XML y aplicaciones móviles. Todos los lenguajes .Net como C#.Net, Visual Basic.Net y Visual C++.net utilizan el mismo entorno de desarrollo integrado (IDE), que habilita el uso compartido de herramientas y facilita la creación de soluciones en varios lenguajes.

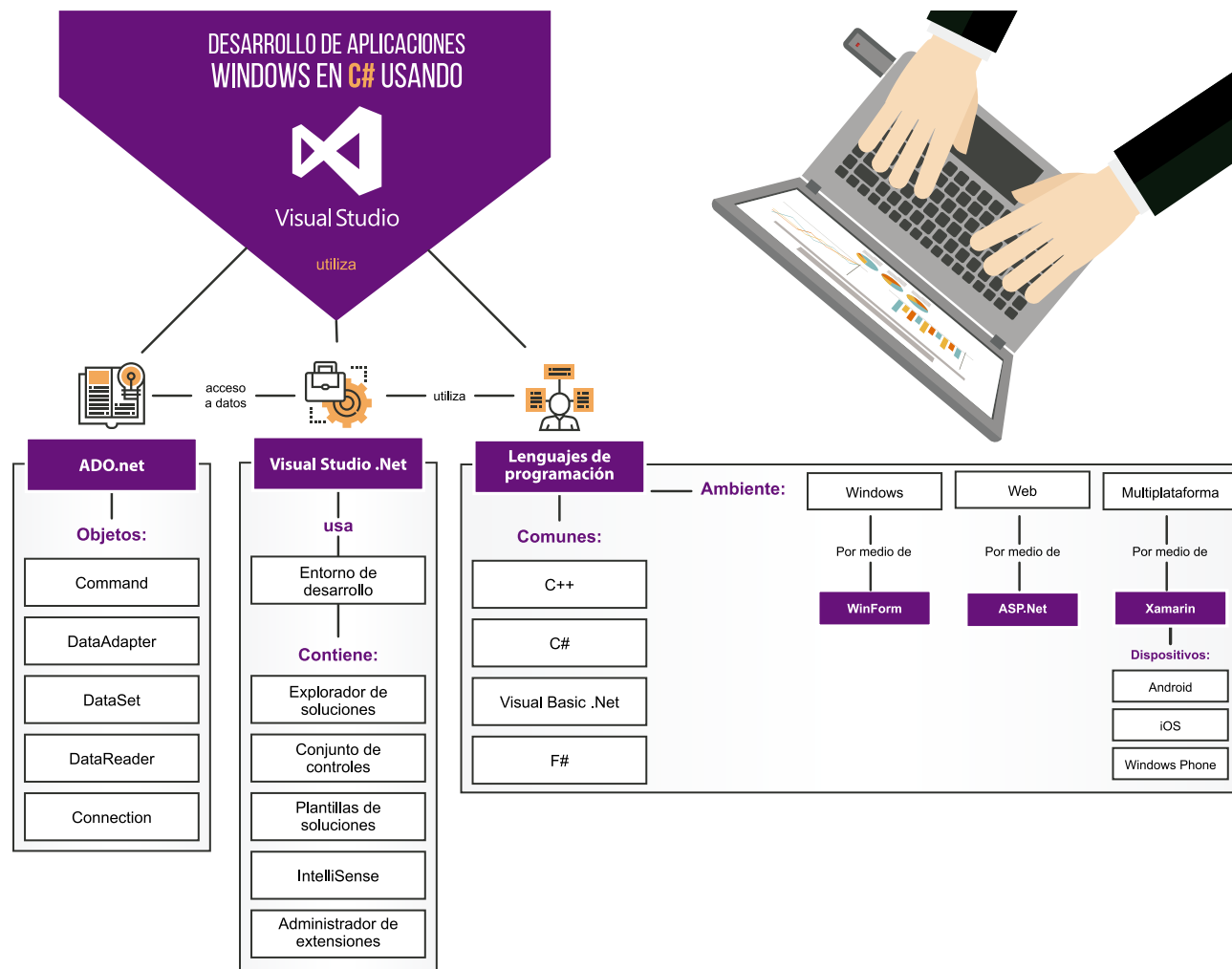
Así mismo, dichos lenguajes utilizan las funciones de .NET Framework, lo cual permite al desarrollador utilizar funciones ya probadas en sus propias aplicaciones, simplificando su construcción o desarrollo.

El propósito de este recurso didáctico es que el aprendiz se familiarice con los elementos del entorno de desarrollo integrado (IDE), conocer los componentes principales del lenguaje C#, la programación en arquitectura de tres capas, el acceso a datos utilizando las herramientas de ADO.Net y por último se realiza un ejercicio práctico del desarrollo de una aplicación Windows Forms utilizando las herramientas anteriormente expuestas y realizando conexión a una base de datos en SQL Server.

Para una mejor comprensión de este recurso didáctico se requiere que el aprendiz haya revisado el objeto de aprendizaje “Introducción base datos SQLServer” y el objeto de aprendizaje “Lenguaje transaccional SQL”.



## Mapa de contenido



## Desarrollo de contenidos



### 1. Proceso de instalación de Visual Studio .Net

La versión más reciente a la fecha de este objeto de aprendizaje es la de Microsoft Visual Studio .Net 2017, la cuál puede ser instalada en versiones de servidor como Windows Server 2012 en adelante y en versión de sistema operativo cliente como Windows 7 Professional en adelante.

Existen 3 versiones de Visual Studio .Net a saber:

- a. Visual Studio Community Edition.
- b. Visual Studio Professional.
- c. Visual Studio Enterprise.

Restricciones de uso de la versión Community:

- Empresas pequeñas (PYMES) para uso comercial siempre y cuando el número de desarrolladores sean 1 a 5 personas.
- Desarrollador independiente para uso personal como para uso comercial, sin ninguna restricción.
- Organización, empresa o persona que desarrolle un software de código abierto (Open Source).
- Los docentes y formadores sin restricciones.
- Para la investigación académica.

Esta versión Community se asemeja a la versión Professional con la única diferencia de la licencia. La gran ventaja de esta versión de Visual Studio es que es gratuita.

Si no se cumple con alguna de las restricciones anteriores se debe de adquirir una licencia Professional o Enterprise con derecho a soporte MSDN (Microsoft Developer Network).

La versión que se utilizará en este objeto de aprendizaje es la versión gratuita Microsoft Visual Studio .Net 2017 Community Edition en español.

Prerrequisitos de instalación:

Visual Studio 2017 requiere .Net Framework 4.6 o posterior, el cual puede ser descargado del sitio web: <https://www.microsoft.com/es-ES/download>

Link de descarga de las versiones de Visual Studio 2017 que se encuentra al final de la página web: <https://www.visualstudio.com/vs/>

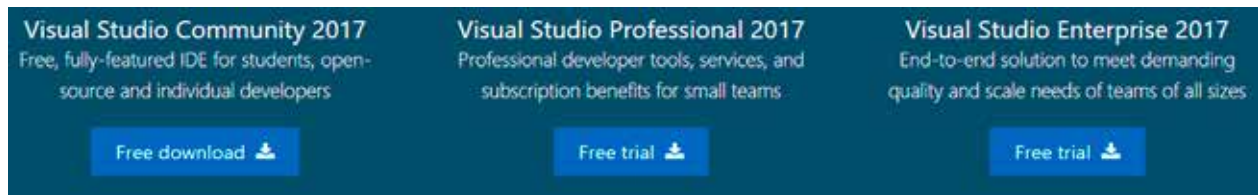


Figura 1. Versiones de descarga de Visual Studio 2017.

Después de instalar los prerequisites se procede a realizar la instalación y seleccionando las opciones a instalar (cargas de trabajo) como se ilustra en la siguiente imagen:

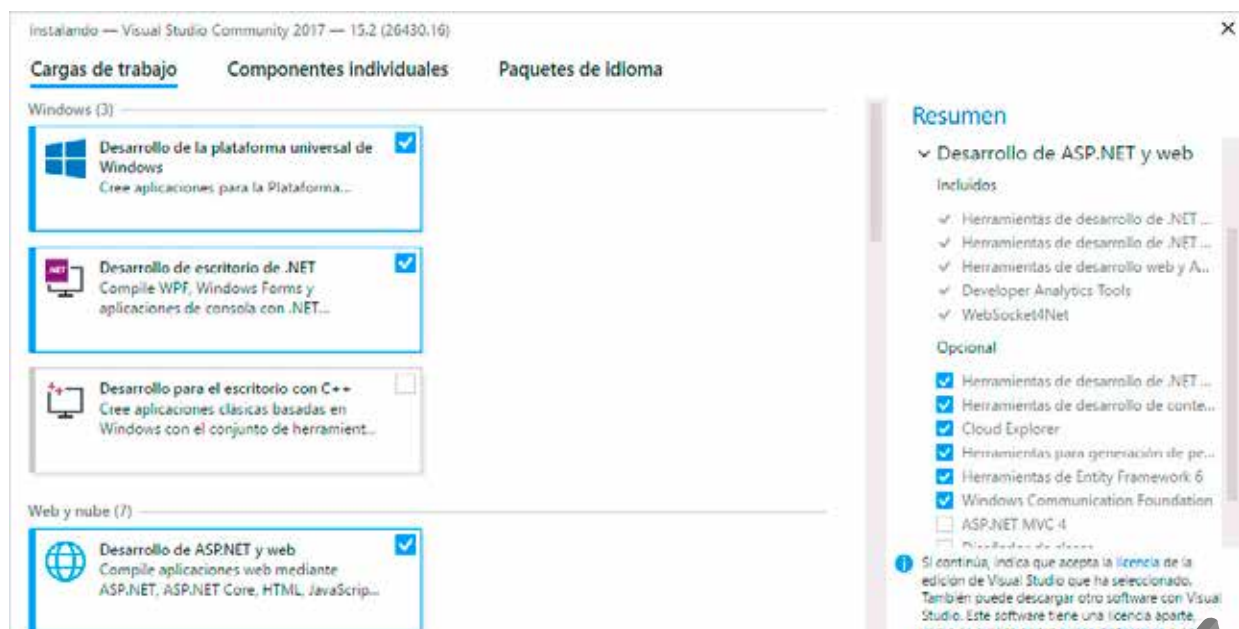
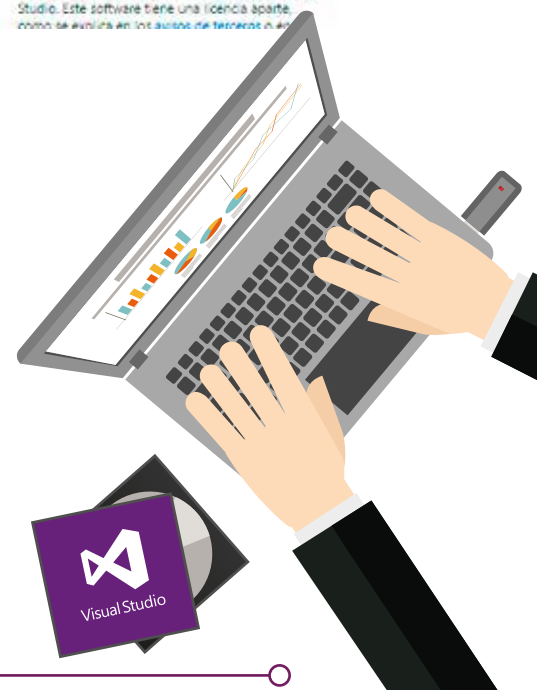


Figura 2. Instalación-Funcionalidades a Instalar.

Se recomienda tener una buena conexión a internet para la instalación de las diferentes características de Visual Studio.



## Visual Studio

### Productos

#### Instalados

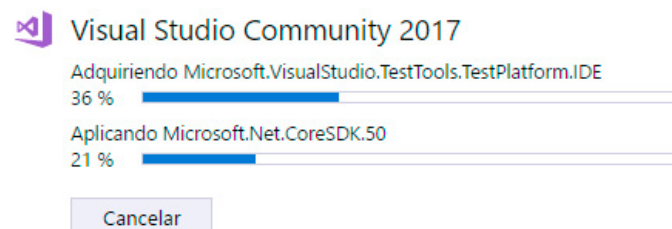


Figura 3. Proceso de Instalación de Visual Studio

Al finalizar la instalación el sistema genera un mensaje de confirmación:

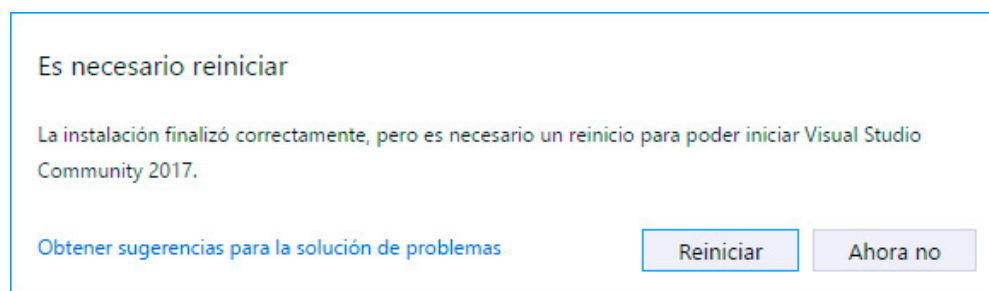


Figura 4. Instalación – Finalización.



## 2. Características principales de Visual Studio .Net

Microsoft Visual Studio .Net es un conjunto de tecnologías diferentes que incluye:

- a. Los lenguajes .Net entre ellos C#, C++ y Visual Basic .Net son lenguajes orientados a objetos que representan la metodología más reciente y exitosa en programación.
- b. Utilizan un marco de trabajo o framework que contiene una biblioteca de clases o componentes ya programados que pueden ser utilizados en las aplicaciones. De esta manera ayuda a los programadores a reutilizar código. Estas bibliotecas de clases están disponibles para cualquier lenguaje compatible con .Net y se organizan por NameSpaces o espacios de nombres:
  - System.Data accesos a bases de datos.
  - System.Windows.Forms desarrollo de aplicaciones de escritorio.
  - System.Web desarrollo de aplicaciones Web y “Web Services” Servicios Web.
  - System Biblioteca de clases generales.
  - System.Net Acceso a servicios de red.
- c. Entorno de ejecución común o CLR (Common Language Runtime): es una máquina virtual interna donde se ejecutan todos los programas desarrollados con tecnología .Net, proporcionando administración automática de seguridad, memoria y desempeño. El código compilado se llama Portable Executable (PE) y puede ser .exe o .dll. El CLR también permite a los lenguajes .Net interactuar entre ellos, por ejemplo, la memoria puede asignarse mediante código escrito en un lenguaje (Visual Basic .Net) y puede ser liberada con código escrito en otro lenguaje (C#). De esta misma manera, los errores pueden ser detectados en un lenguaje y procesados en otro. El proceso de ejecución de cualquier aplicación incluye los pasos siguientes:
  - 1. Diseñar y escribir el código fuente.
  - 2. Compilar el código fuente a código intermedio (MSIL Microsoft Intermediate Language o simplemente IL).
  - 3. Compilar el código intermedio a código nativo.
  - 4. Ejecutar el código nativo.



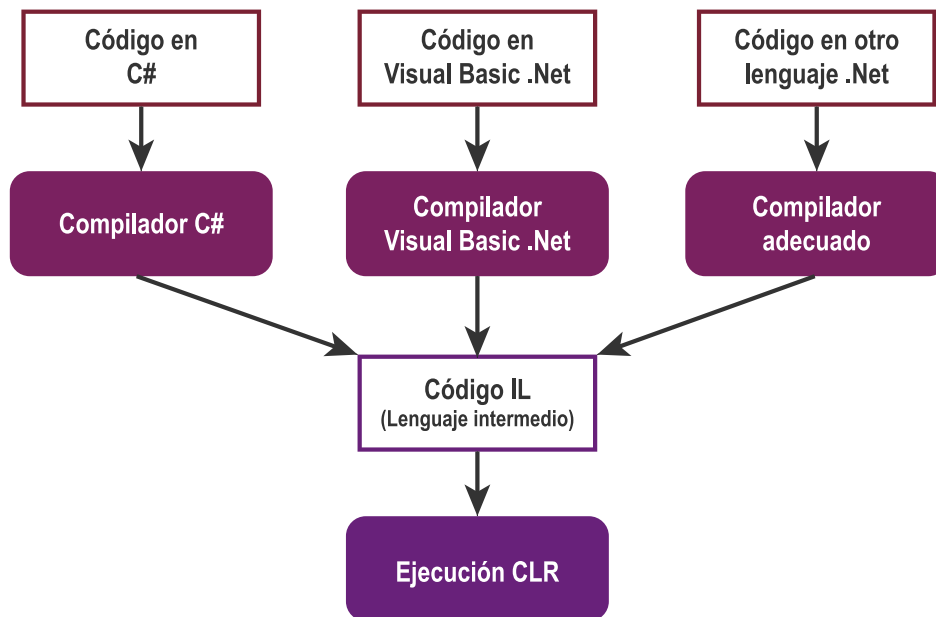


Figura 5. Compilación de lenguajes en .Net

**d.** Entorno de desarrollo integrado “IDE” (Integrated Development Environment) contiene herramientas integradas que facilitan la creación o el desarrollo de programas sin tener que utilizar programas externos, proporcionando un entorno de programación completo. El IDE de .Net contiene:

- Editor de código con IntelliSense que permite detectar errores antes de ejecutar la aplicación. Los errores potenciales se subrayan mientras se escriben ayudando al desarrollador al escribir código .Net.
- Depurador de código que permite colocar puntos de interrupción y ver la evolución de la aplicación, así como el valor de las variables.
- Compilador de código que permite detectar errores de sintaxis antes de ejecutar la aplicación.
- Diseño de formularios, haciendo posible crear aplicaciones Windows con la facilidad de arrastrar y soltar controles en el diseñador de formularios Windows de .Net.
- Módulos de ayuda local o ayuda en línea.

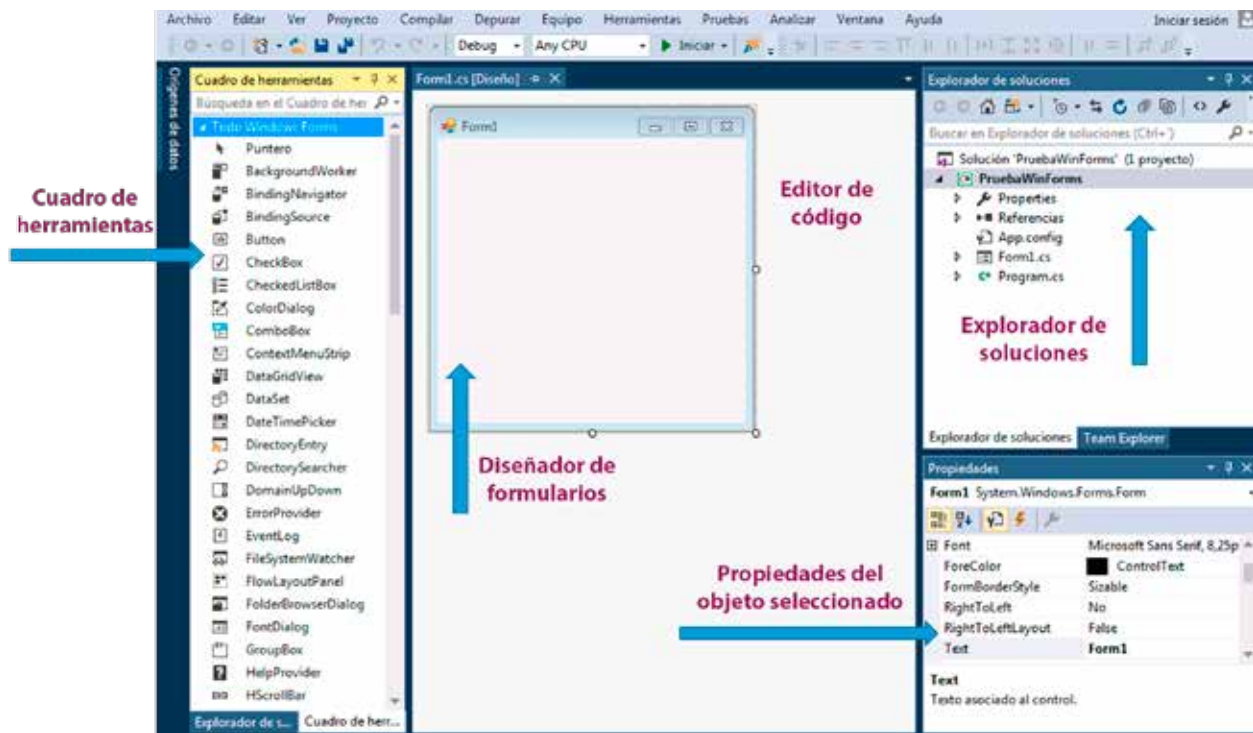


Figura 6. IDE de desarrollo .Net

## Tipos de aplicaciones y servicios .Net:

- Aplicaciones de consola.
- Aplicaciones de Windows Forms o formularios Windows.
- Aplicaciones WPF (aplicaciones que muestran una interfaz gráfica enriquecida).
- Aplicaciones de ASP.NET (aplicaciones para la Web). ASP.NET es una plataforma web que proporciona todos los servicios necesarios para compilar y ejecutar aplicaciones web.
- Servicios Windows para la automatización de tareas a nivel de servicios.
- Servicios web o Web Services para integración entre plataformas web.
- Aplicaciones multiplataforma Xamarin para el desarrollo de aplicaciones móviles para sistemas operativos Androide, iOS y Windows Phone.

### 3. Arquitectura de tres capas



Es una arquitectura de desarrollo que permite separar componentes de una aplicación con los siguientes beneficios:

- Separar funcionalidades donde cada capa tiene una función específica, permitiendo su propia compilación.
- Reutilizar código. Al utilizar programación por capas como proyectos independientes es posible reutilizar su código.
- Hace más fácil reemplazar o modificar una capa, sin afectar a las capas restantes.
- Capacidad de migrar a otro motor de Base de Datos sin grandes impactos al resto del proyecto.
- Poder cambiar la capa de presentación de la aplicación sin afectar la lógica de la aplicación ni la base de datos.

Los principales componentes de la arquitectura de tres capas son:

**Interfaz de usuario o UI (User interface):** es la capa encargada de interactuar con el usuario, es decir, son aquellas ventanas, cuadros de diálogos, mensajes o páginas web (en el caso del desarrollo web), que el usuario final utiliza para comunicarse con la aplicación, por medio de esta capa el usuario solicita que se ejecuten las tareas proporcionando parámetros de entrada y recibiendo datos como respuesta.



Esta capa se comunica con la capa de lógica de negocio, enviando y solicitando información y con la capa de entidades usando sus objetos para enviar y recibir esta información.

**Lógica de negocio o BL (Business Logic):** es la capa encargada de implementar la lógica del negocio, es decir, esta capa recibe de la capa de *Presentación las solicitudes*, valida que las condiciones que establece el negocio se cumplan antes de realizar dicha acción o de hacer la respectiva solicitud a la capa de *Acceso a datos*. Por ejemplo, validaciones de campos, de tipos de datos o reglas de negocio de la aplicación.



**Acceso a datos o DA (Data Access):** es la capa encargada de la comunicación con la base de datos y las operaciones sobre esta, por ejemplo: acciones CRUD (Create, Read, Update y Delete), ejecución de procedimientos almacenados o funciones. Se encarga de recibir las peticiones de la capa de *Lógica de Negocio*, ejecutar las acciones y devolver el resultado a esta capa.



**Capa de entidades o EL (Entity Layer):** es la capa encargada de contener todos aquellos objetos que representan al negocio, y es la única que puede ser instanciada en las 3 capas anteriores, es decir, sólo ella puede tener comunicación con el resto, pero su función se limita a únicamente ser un puente de transporte de datos.

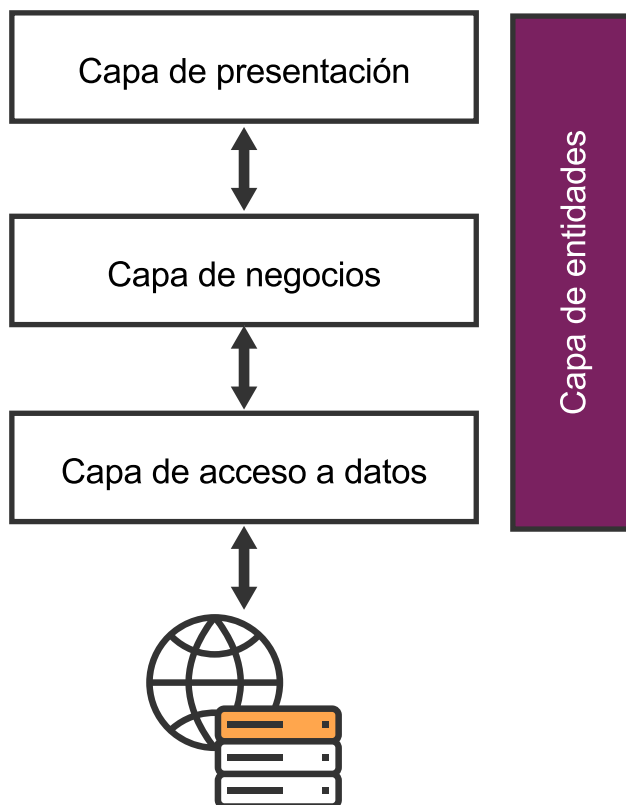


Figura 7. Arquitectura de 3 Capas



#### 4. Acceso a datos

ADO.NET es la tecnología que las aplicaciones Windows en .Net utilizan para comunicarse con una base de datos. ADO.Net es un conjunto de clases que exponen servicios de acceso a datos para desarrolladores .Net. Hace parte integral del .NET Framework y proporciona acceso a bases de datos relacionales, XML, archivos.

ADO.NET incluye proveedores de datos para conectarse a una base de datos, ejecutar comandos y recuperar resultados.

ADO.Net tiene clases que realizan tareas habituales en la gestión de bases de datos:

- Establecer una conexión con la base de datos.
- Solicitar al servidor de bases de datos, datos específicos (por ejemplo, consultas).
- El servidor retorna los datos solicitados.
- El usuario modifica los datos, y envía las actualizaciones al servidor.
- Cerrar la conexión.

Dentro de ADO.Net existen los siguientes espacios de nombres principales dentro de la biblioteca de clases .Net:

- **System.Data:** contiene las clases fundamentales de ADO.Net incluyendo DataSet y DataRelation que permiten manipular los datos relacionales.
- **System.Data.OleDb:** contiene las clases utilizadas para conectar al proveedor OLE DB, incluyendo OleDbConnection y OleDbCommand.
- **System.Data.SqlClient:** contiene las clases utilizadas para conectar a bases de datos Microsoft SQL Server.
- **System.Data.OracleClient:** contiene las clases utilizadas para conectar a bases de datos Oracle.



ADO.Net permite la gestión de datos en ambientes conectados (en línea), o en ambientes desconectados, dependiendo de la necesidad de contar con datos actualizados en línea o la disponibilidad de acceso al servidor.

ADO.NET ofrece un modelo unificado de desarrollo independiente del lenguaje de programación (Visual Basic .NET, C#, etc.) e independiente de la arquitectura de la aplicación a desarrollar (aplicación web, aplicación de escritorio, aplicación de consola, etc.).

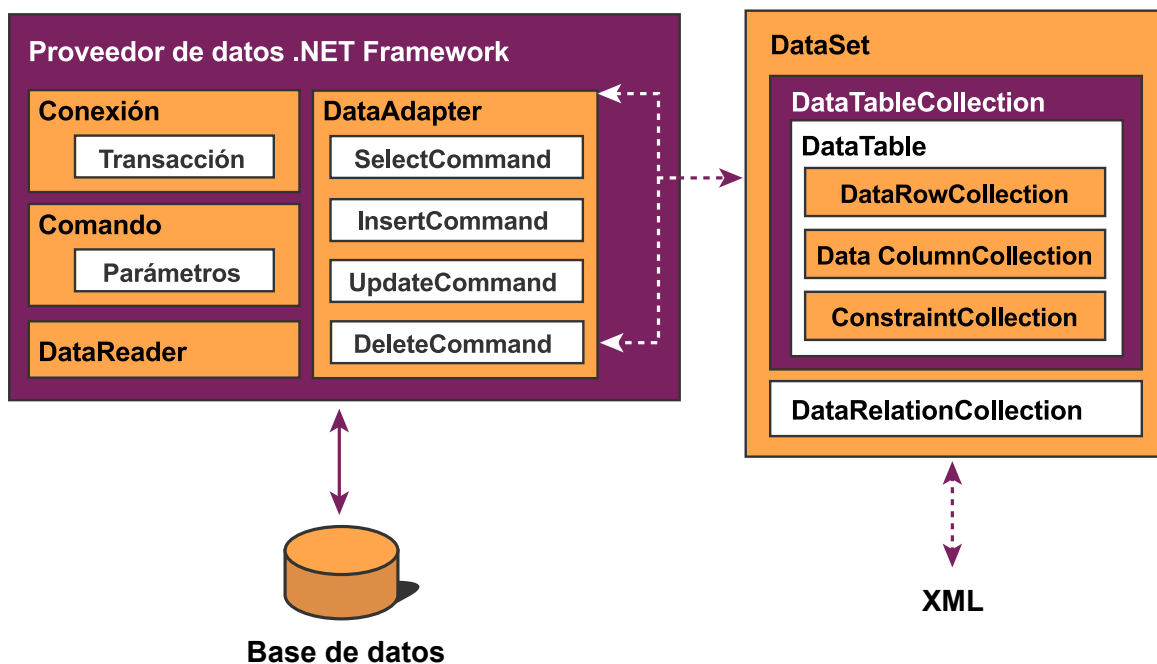


Figura 8. Arquitectura ADO.Net / Fuente: (Microsoft, 2007)

### Objetos ADO.Net:

OBJETO	FUNCIÓN
<b>Connection</b>	Define las características de conexión a una base de datos, tales como el nombre del servidor, nombre de la base de datos, usuario, contraseña, entre otros en un <code>ConnectionString</code> . Administra conexiones con diferentes motores, por ejemplo, <code>SQLClient</code> para bases de datos <code>SQLServer</code> , <code>OracleClient</code> para bases de datos <code>Oracle</code> y <code>OleDb</code> para otros motores de bases de datos.
<b>Command</b>	Permite ejecutar sentencias SQL sobre las bases de datos, como Consultar, Insertar, Actualizar o Eliminar datos.
<b>DataSet</b>	Objeto que almacena información de manera desconectada para hacer operaciones rápidas y que contiene una colección de uno o más objetos <code>DataTable</code> así como sus relaciones <code>DataRelation</code> que se pueden tener en una base de datos.
<b>DataAdapter</b>	Permite alimentar un <code>Dataset</code> con datos a partir de cuatro objetos <code>command</code> ( <code>SelectCommand</code> , <code>InsertCommand</code> , <code>UpdateCommand</code> , <code>DeleteCommand</code> ).

<b>DataTable</b>	Representa la estructura de una tabla de datos y que está contenida en un dataSet. Contiene objetos para manipulación de filas DataRow y manipulación de columnas DataColumn.
<b>DataReader</b>	Objeto de lectura rápida de datos en una sola dirección, no permite modificación de datos.

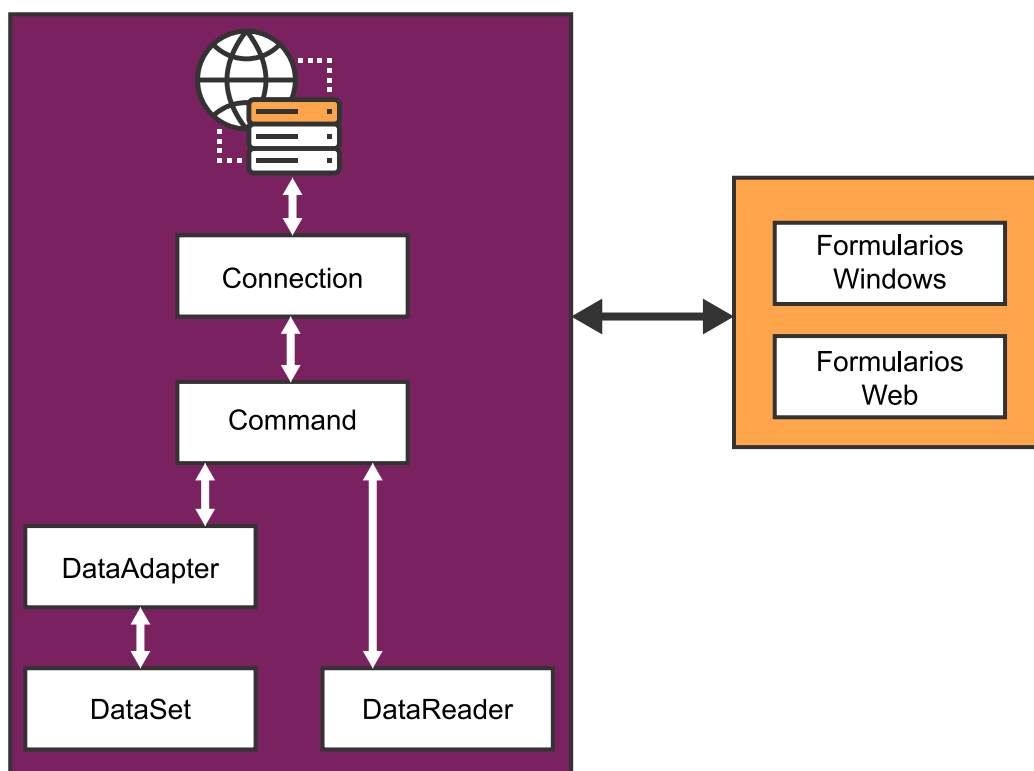


Figura 9. Objetos ADO.Net y Consumidores de datos  
Fuente: elaboración del Autor



## 5. Controles principales de C#











Visual Studio .Net contiene una gran cantidad de controles en cada uno de sus lenguajes de programación. Estos controles se encuentran inmersos dentro de la ventana de ToolBox o caja de controles clasificados por funcionalidad (controles generales, de datos, de validación, entre otros).

En algunos casos cuando se adquieren controles de terceros (de otras casas de software especializadas en desarrollo de controles), estos se ubican por defecto al final del ToolBox.













Los controles de la caja de herramientas se habilitan y están disponibles cuando se está en modo de diseño y se deshabilitan cuando se está en modo de edición de código.

A continuación, los controles principales en C# y su descripción:

### 5.1 Controles generales principales

Imagen	Descripción
 Button	Control que puede contener imagen o texto y responde a eventos cuando es presionado, por ejemplo evento clic.
 CheckBox	Control de usuario que se puede activar o desactivar según la opción seleccionada.
 ComboBox	Control que muestra un cuadro de texto editable o una lista despegable de valores.
 DateTimePicker	Control que permite al usuario seleccionar de una lista despegable la fecha.
 Label	Control que proporciona una forma de establecer un texto mediante programación.
 ListBox	Control que permite seleccionar uno o varios elementos de una lista predefinida.
 MaskedTextBox	Control que permite el ingreso de texto según una máscara preestablecida. Por ejemplo, 00/00/0000 para fechas o \$999,999.00 para valores de moneda.
 NumericUpDown	Control que muestra y establece un valor numérico único el cual se puede incrementar o disminuir usando los botones hacia arriba y hacia abajo.
 PictureBox	Control que se utiliza para mostrar gráficos en formato de mapa de bits, .gif, .jpeg, metarchivo o icono.
 ProgressBar	Control que indica el progreso de una operación. Está compuesto de una ventana que se llena gradualmente a medida que una operación progresa.



 RadioButton	Control que permite seleccionar una única opción dentro de un grupo de varias opciones disponibles.
 TextBox	Control que permite la entrada de texto.
 TreeView	Control que se utiliza para mostrar datos jerárquicos, como una tabla de contenido o un directorio de archivo, en una estructura de árbol.
 WebBrowser	Control que permite al usuario explorar páginas Web o utilizar el control como visor simple de documentos HTML.
<b>Controles de menú principales:</b>	
 MenuStrip	Control que permite crear menús personalizados con características de diseño, como la alineación, orden de texto e imágenes.
 ToolStrip	Control que permite crear opciones de menú en la sección del ToolBar (barra de herramientas).
 ContextMenuStrip	Control que permite crear opciones de menú contextual (botón derecho del mouse).
<b>Controles contenedores principales:</b>	
 GroupBox	Control que muestra un marco alrededor de un grupo de controles y un título de manera opcional.
 Panel	Control utilizado para agrupar colecciones de controles.
 TabControl	Control que muestra múltiples fichas, similares a las etiquetas de un conjunto de carpetas de un archivador. Las fichas pueden contener imágenes y otros controles.
<b>Controles de datos principales:</b>	
 Chart	Control que permite generar gráficos y que responde a eventos.
 DataGridView	Control que muestra datos en formato de tabla (filas y columnas) personalizable.



**Eventos:** son las acciones que el control puede realizar, cada control proporciona múltiples eventos, donde el evento más utilizado es el evento Clic. Para escribir el código o instrucciones del evento que se desea simplemente seleccionar el evento de la lista de eventos y dar doble clic sobre el mismo. En este momento el IDE agrega automáticamente un método y habilita el editor de código para escribir sus instrucciones.

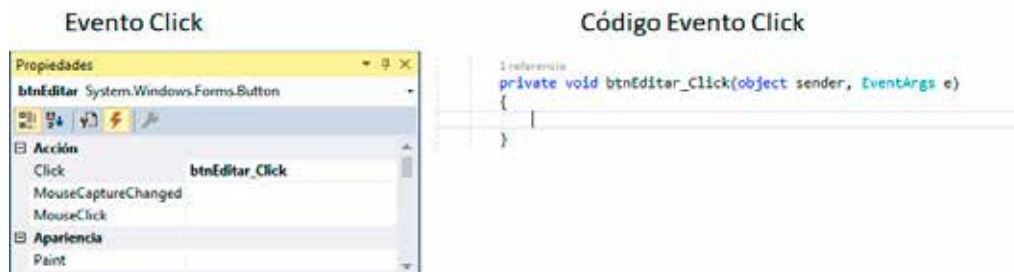


Figura 11. Evento clic.

## 6. Detalles del lenguaje C#



### 6.1 Variables

Representan una cadena de texto o un valor numérico o un objeto de una clase. El valor almacenado en la variable puede cambiar, pero el nombre sigue siendo el mismo. Una variable se declara con un tipo de datos y una etiqueta, es decir, que antes de que un valor se pueda almacenar en una variable, se debe especificar el tipo de la variable.

#### Ejemplo de una variable.

```
int x = 10; //variable x tiene el valor de 10 y está definida como de tipo entero.
```

```
x = 20; //ahora el valor de la variable x es de 20.
```

El tipo especifica, entre otras cosas, la cantidad de memoria exacta que se debe asignar para almacenar el valor cuando la aplicación está en ejecución. En C# se debe de cumplir ciertas reglas al convertir una variable de un tipo en otro.



## Tipo de datos comunes en C#

Tipo de datos	Intervalo
byte	0 ... 255
sbyte	-128 ... 127
short	-32,768 ... 32,767
int	-2,147,483,648 ... 2,147,483,647
long	-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807
float	-3.402823e38 .. 3.402823e38
double	-1.79769313486232e308 ... 1.79769313486232e308
decimal	-79228162514264337593543950335 ... 79228162514264337593543950335
char	Un carácter Unicode.
string	Una cadena de caracteres.
bool	True o False.
Object	Un objeto.

## Ejemplos de tipos de variables

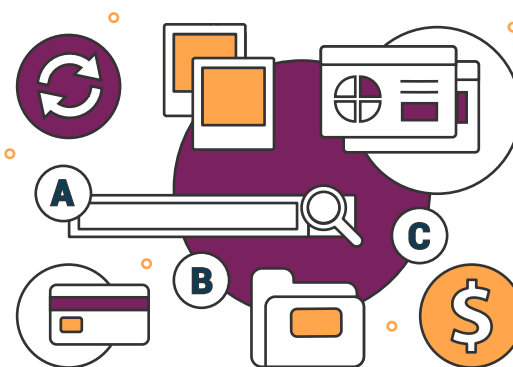
```
string Apellidos;
int Modelo_vehiculo;
bool Mayor_de_Edad;
float Promedio;
```

## 6.2 Constantes

Contiene un valor que es asignado cuando se compila el programa y nunca cambia después. Las constantes se declaran con la palabra clave **const**; son útiles para que el código sea más legible.

## Ejemplos de constantes

```
const double pi = 3.1415926;
const int limite_velocidad = 80;
```



Una variable con el identificador **readonly** es como una constante, donde su valor es asignado al iniciar el programa. Esto le permite establecer el valor basándose en alguna otra condición que no conoce hasta que se ejecuta el programa. Pero después de esa primera asignación, el valor no puede cambiar de nuevo mientras el programa se está ejecutando.

### 6.3 Conversiones

Los tipos de datos se pueden convertir implícitamente, en cuyo caso el compilador hace automáticamente la conversión, o explícitamente, utilizando un operador de conversión explícita, donde el programador fuerza la conversión.

La clase **System.Convert** proporciona métodos para las conversiones compatibles y está disponible para todos los lenguajes .Net cuyo destino es Common Language Runtime. Esta clase realiza conversiones de ampliación y conversiones de restricción, así como conversiones en tipos de datos no relacionados. Por ejemplo, se admiten las conversiones de los tipos String a los tipos numéricos, de los tipos DateTime a los tipos String y de los tipos String a los tipos Boolean. Si no se admite la conversión se produce una excepción.

#### Ejemplos

```
string myString = "true";  
bool myBool = Convert.ToBoolean(myString); //variable myBool tiene valor de true
```

```
string newString = "123456789";  
int myInt = Convert.ToInt32(newString); //variable myInt tiene valor de 123456789
```

En algunos casos si se realiza una conversión de restricción con la clase Convert, cambia el valor del elemento que se está convirtiendo. En el siguiente ejemplo se convierte un valor Double en un valor Int32. En este caso, el valor se redondea de 42.72 a 43 para la conversión.

```
Double myDouble = 42.72;  
int myInt = Convert.ToInt32(myDouble); //variable myInt tiene un valor de 43
```

Existe otra forma de conversión explícita y es colocando un prefijo del tipo de dato que define el tipo de conversión que se desea realizar.

```
double x = 1234.7;  
int a;  
a = (int)x; //convierte tipo de dato double a entero, valor de a=1234
```

Para mayor información de tipos de datos y conversiones, favor consultar la siguiente dirección web: [https://msdn.microsoft.com/es-es/library/ms228360\(v=vs.90\).aspx](https://msdn.microsoft.com/es-es/library/ms228360(v=vs.90).aspx)

### 6.4 Comentarios

Es muy buena práctica documentar el código, de esta manera es fácil recordar el objetivo del conjunto de instrucciones y más sencillo hacerle mantenimiento al código por parte de otros programadores.

Existen dos formas de realizar comentarios:

- El símbolo // coloca en comentario todo lo que esté a su derecha y en la misma línea.
- El texto que esté entre los símbolos /\* y \*/ es considerado como comentarios y pueden estar en varias líneas.

### Por ejemplo

```
/****** COMENTARIO *****/
```

```
*estas líneas de comentarios son ignoradas al compilar
```

```
*Función ABC que realiza el proceso XYZ
```

```
*****/
```

Los comentarios quedan identificados con un color verde en el IDE de Visual Studio.

**IMPORTANTE:** Microsoft C# es sensible a las mayúsculas, es decir, unas variables que tengan el mismo nombre, pero escritas de manera diferente, serán para C# variables diferentes.

```
string Apellidos
```

```
string apellidos
```

Para C# son variables diferentes y por consiguiente tendrán valores diferentes.

## 6.5 Operadores principales

<b>Operador de Igualdad</b>	x == y: igualdad x != y: distinto
<b>Operador lógico</b>	x & y: corresponde al AND lógico x   y: corresponde al OR lógico
<b>Operador condicional lógico</b>	x && y: corresponde al AND lógico condicional. Si el primer operando es false, C# no evalúa el siguiente operando. x    y: corresponde al OR lógico condicional. Si el primer operando es true, C# no evalúa el segundo operando.
<b>Operador de asignación</b>	x = y: asignación x += y: agrega el valor de y al valor de x, es similar a la instrucción x = x + y. De la misma forma para resta, multiplicación y división.

Para detallar sobre los operadores de C#, favor visitar el sitio web: <https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/index>



## 7. Creación de aplicaciones en Windows Forms

En general, para crear una aplicación Windows se deben de seguir los siguientes pasos:

- I. Crear un nuevo proyecto de Windows Forms. Menú Archivo > Nuevo > Proyecto. Dentro de las plantillas mostradas seleccionar el lenguaje Visual C# - Aplicación de Windows Forms.

**Nota:** un proyecto puede ser una sola aplicación.

Una solución puede contener varios proyectos, por ejemplo, para una aplicación de tres capas se tendría una solución con diferentes tipos de proyectos (proyecto de entidades, proyecto de acceso a datos, proyecto de reglas de negocio) donde cada uno de estos proyectos son proyectos de clases que generan .dll independientes como se detalló en el capítulo de arquitectura en 3 capas.

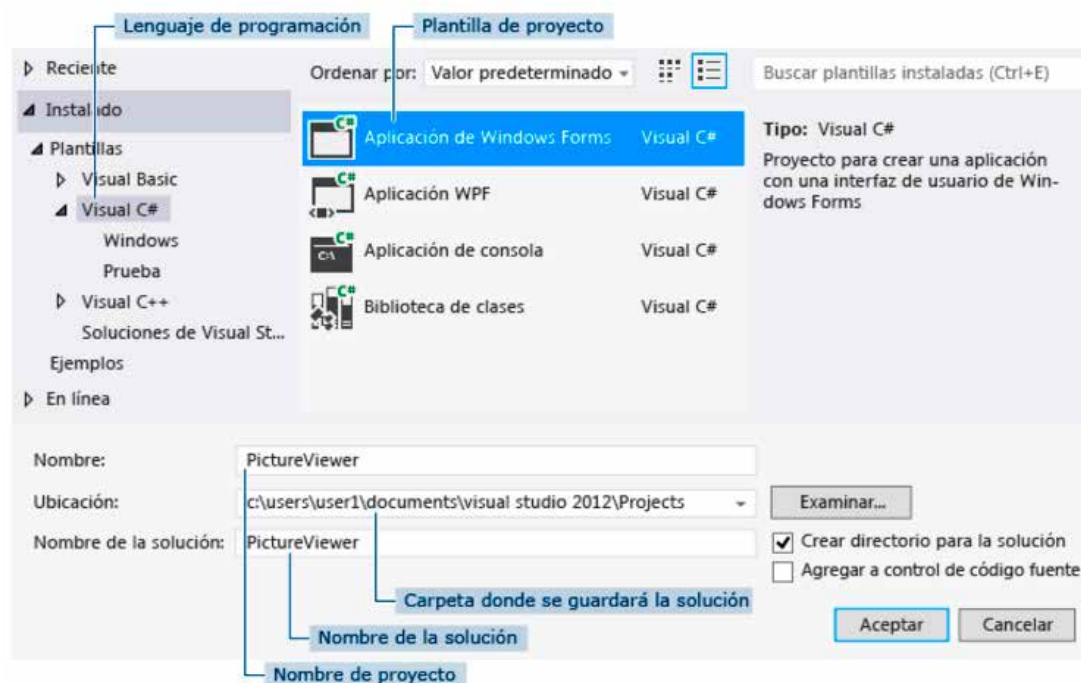


Figura 12 Ventana nuevo proyecto.

- II. Agregar los controles sobre el formulario. Desde el cuadro de herramientas se selecciona el control y este se arrastra al formulario.
- III. Establecer las propiedades del formulario y de los controles.
- IV. Escribir el código para los eventos que se considere en cada uno de los controles.
- V. Guardar – Compilar y ejecutar la aplicación.
- VI. De manera opcional, utilizar el depurador para hacer seguimiento del comportamiento de la aplicación y el valor de sus variables.



Para el ejemplo de Citas y Tratamientos Médicos, se creó un proyecto de aplicación Windows C# utilizando la respectiva plantilla llamado SInTratamientos generando el siguiente entorno de desarrollo:

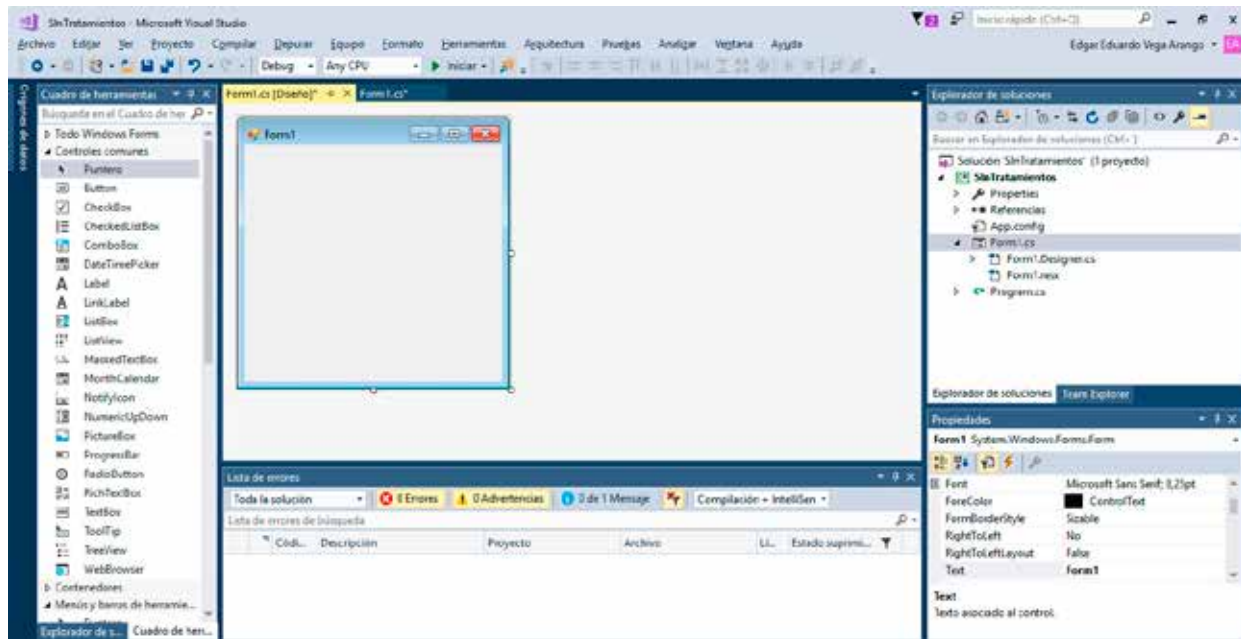


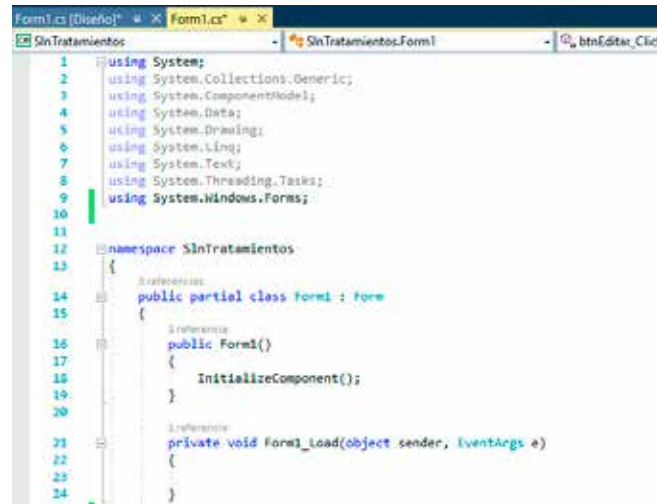
Figura 13. Entorno de desarrollo.

Se ilustra a la izquierda el cuadro de herramientas, a la derecha el *Explorador de soluciones* y en la parte inferior la ventana de Propiedades. En el centro el formulario y la ventana de errores.

Por defecto, se observa un formulario, denominado Form1, que contiene los archivos de código Form1.cs y Form1.Designer.cs; el primero es el utilizado por el programador para escribir el código y el segundo, el utilizado por el diseñador de formularios. También se observa un nodo *References* que agrupa las referencias a las bibliotecas de clases de objetos que utilizará la aplicación; y se pueden añadir nuevas referencias a otras bibliotecas haciendo clic con el botón secundario del ratón sobre ese nodo.

En la barra de herramientas del Explorador de soluciones se muestra una barra de botones que permiten ver el código, mostrar todos los archivos, la ventana de propiedades, entre otros. Por ejemplo, si estamos viendo el diseñador de formularios y hacemos clic en el botón "<>" Ver Código, la página de diseño será reemplazada por el editor de código, como se puede observar en la figura siguiente:





```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11
12  namespace SinTratamientos
13  {
14      public partial class Form1 : Form
15      {
16          public Form1()
17          {
18              InitializeComponent();
19          }
20
21          private void Form1_Load(object sender, EventArgs e)
22          {
23
24          }
25      }

```

Figura 14. Editor de código.

Se procede a crear un formulario MDI que es el formulario principal y el que va a contener el menú con las opciones del sistema. Para esto sobre la solución – botón derecho – nuevo elemento – Windows Forms y seleccionar Formulario primario MDI.

Propiedades del MDI

Text: Sistema de Tratamientos Médicos

Name: MDISistema

WindowState: Maximized

IsMdiContainer: True

A este formulario MDI se le agrega un control MenuStrip que contendrá el menú de opciones del sistema. Este menú se identificará en la propiedad “Name” como menuPrincipal.

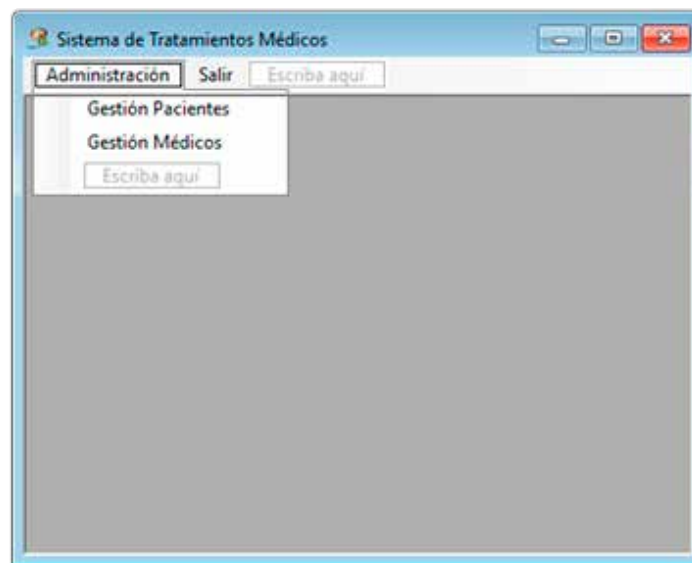


Figura 15. Formulario principal MDI.

El código interno para la opción de Gestión Pacientes y la opción Salir, es el siguiente:

```
1 referencia
private void gestiónPacientesToolStripMenuItem_Click(object sender, EventArgs e)
{
    FrmPacientes objFrmPacientes = new FrmPacientes();
    objFrmPacientes.MdiParent = this.MdiParent; //defino que es formulario hijo
    objFrmPacientes.Show(); //muestra el formulario pacientes
}

1 referencia
private void salirToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit(); //cierra la aplicación
}
}
```

Figura 16. Código del formulario principal MDI.

## 7.1 Arquitectura de la aplicación

La aplicación va a utilizar la arquitectura de tres capas, para lo cual se van a crear las *Capas de negocios*, *Capas de datos* y *Capas de entidades*.

**Nota:** estas capas son proyectos de clase, que son independientes y al momento de compilar cada una genera su propia librería o archivo .dll.

Para adicionar un proyecto de Clase, ubicarse en la parte superior sobre el texto Solución, botón derecho > Agregar > Nuevo proyecto y sobre las plantillas mostradas, seleccionar la plantilla biblioteca de clases.

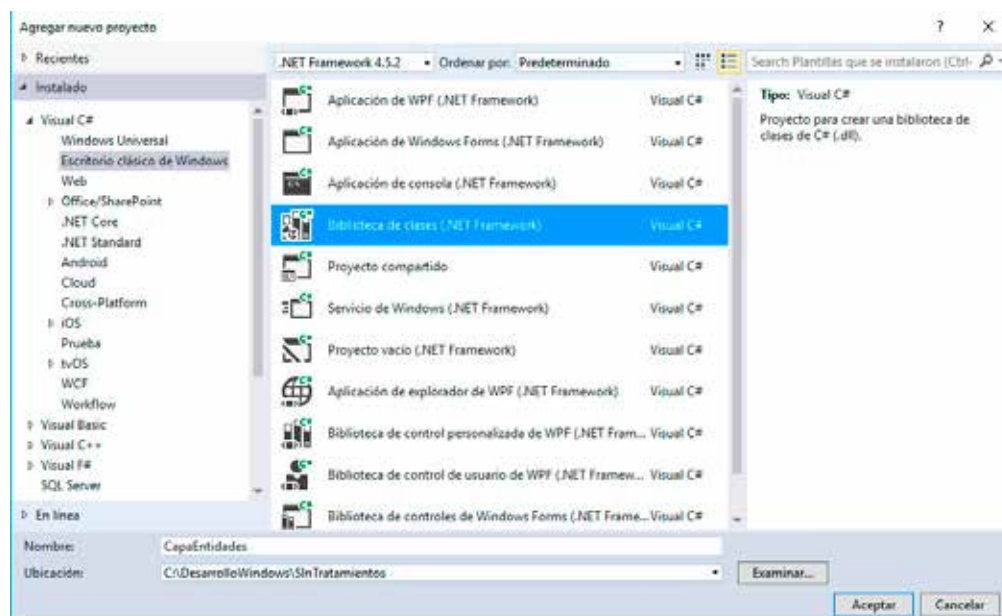
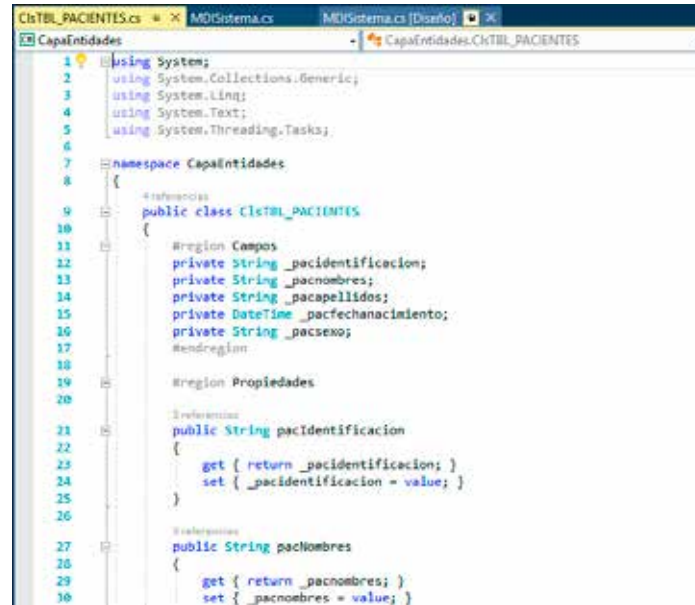


Figura 17. Adicionar proyecto biblioteca de clases.

- **Capa de entidades:** hace referencia a los campos y propiedades según los campos que contenga cada una de las tablas. Esta capa es referenciada por las demás capas del sistema.



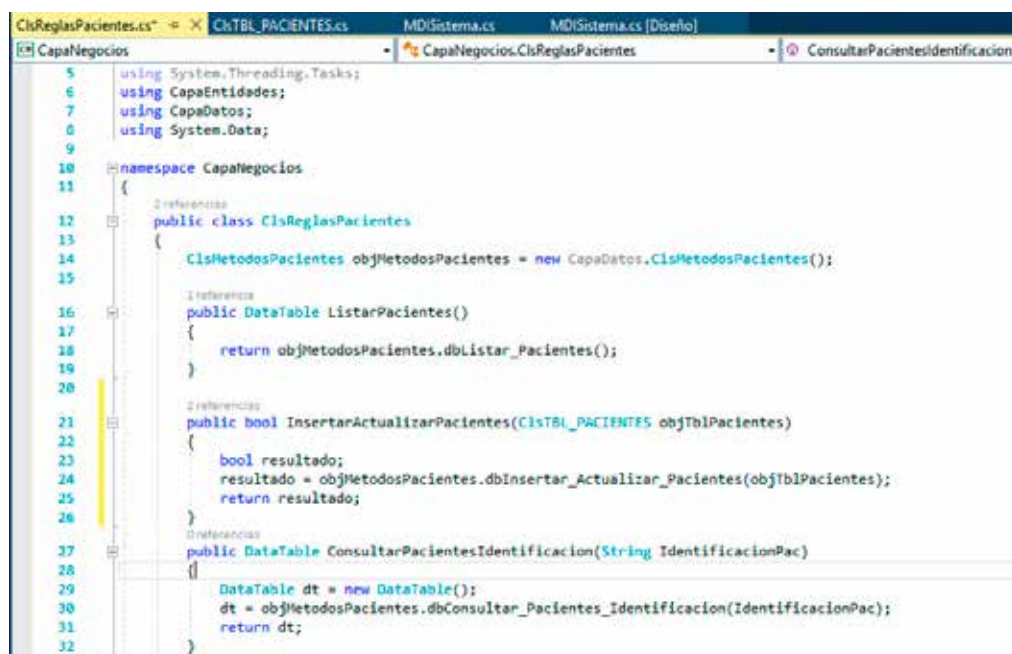
```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidades
8  {
9      #region Campos
10     public class ClsTBL_PACIENTES
11     {
12         private String _pacidentificacion;
13         private String _pacnombres;
14         private String _pacapellidos;
15         private DateTime _pacfechanacimiento;
16         private String _pacsexo;
17         #endregion
18
19         #region Propiedades
20
21         #region Paciente
22         public String pacIdentificacion
23         {
24             get { return _pacidentificacion; }
25             set { _pacidentificacion = value; }
26         }
27
28         public String pacNombres
29         {
30             get { return _pacnombres; }
31             set { _pacnombres = value; }
32         }
33     }
34 }

```

Figura 18. Ejemplo capa de entidades.

- **Reglas de negocio o lógica de negocio:** esta capa detalla la lógica del sistema de información. Se conecta a la *Capa datos* y a la *Capa entidades*.



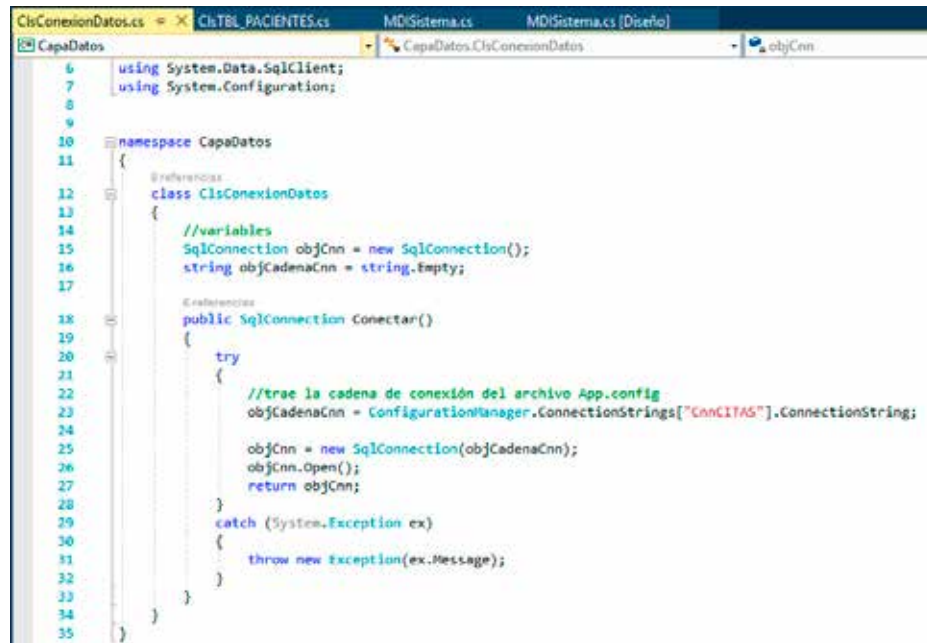
```

5  using System.Threading.Tasks;
6  using CapaEntidades;
7  using CapaDatos;
8  using System.Data;
9
10 namespace CapaNegocios
11 {
12     #region Reglas de Negocio
13     public class ClsReglasPacientes
14     {
15         private ClsMetodosPacientes objMetodosPacientes = new CapaDatos.ClsMetodosPacientes();
16
17         #region Listar Pacientes
18         public DataTable ListarPacientes()
19         {
20             return objMetodosPacientes.dbListar_Pacientes();
21         }
22
23         #region Insertar/Actualizar Pacientes
24         public bool InsertarActualizarPacientes(ClsTBL_PACIENTES objTblPacientes)
25         {
26             bool resultado;
27             resultado = objMetodosPacientes.dbInsertar_Actualizar_Pacientes(objTblPacientes);
28             return resultado;
29         }
30
31         #region Consultar Pacientes
32         public DataTable ConsultarPacientesIdentificacion(String IdentificacionPac)
33         {
34             DataTable dt = new DataTable();
35             dt = objMetodosPacientes.dbConsultar_Pacientes_Identificacion(IdentificacionPac);
36             return dt;
37         }
38     }
39 }

```

Figura 19. Ejemplo capa de negocios.

- **Acceso a datos:** en esta capa se establece el acceso al motor de bases de datos, los métodos de consultas y operaciones de inserción, actualización y eliminación.



```

6      using System.Data.SqlClient;
7      using System.Configuration;
8
9
10     namespace CapaDatos
11     {
12         class ClsConexionDatos
13         {
14             //variables
15             SqlConnection objCnn = new SqlConnection();
16             string objCadenaCnn = string.Empty;
17
18             //metodos
19             public SqlConnection Conectar()
20             {
21                 try
22                 {
23                     //trae la cadena de conexión del archivo App.config
24                     objCadenaCnn = ConfigurationManager.ConnectionStrings["CnnCITAS"].ConnectionString;
25
26                     objCnn = new SqlConnection(objCadenaCnn);
27                     objCnn.Open();
28                     return objCnn;
29                 }
30                 catch (System.Exception ex)
31                 {
32                     throw new Exception(ex.Message);
33                 }
34             }
35         }
36     }

```

Figura 20. Ejemplo de conexión a datos.

En esta porción de código se visualiza que hay una variable (objCadenaCnn) que extrae la cadena de conexión "CnnCITAS" definida en un archivo de configuración App.config, de esta manera, al existir cualquier cambio en la cadena de conexión (por ejemplo, nombre del servidor, usuario o contraseña) solo se cambia en este archivo y la aplicación obtiene estos valores y continúa su procesamiento.

El archivo App.config es un archivo del sistema en formato XML que contiene los valores globales de la aplicación y se encuentra dentro del proyecto principal.



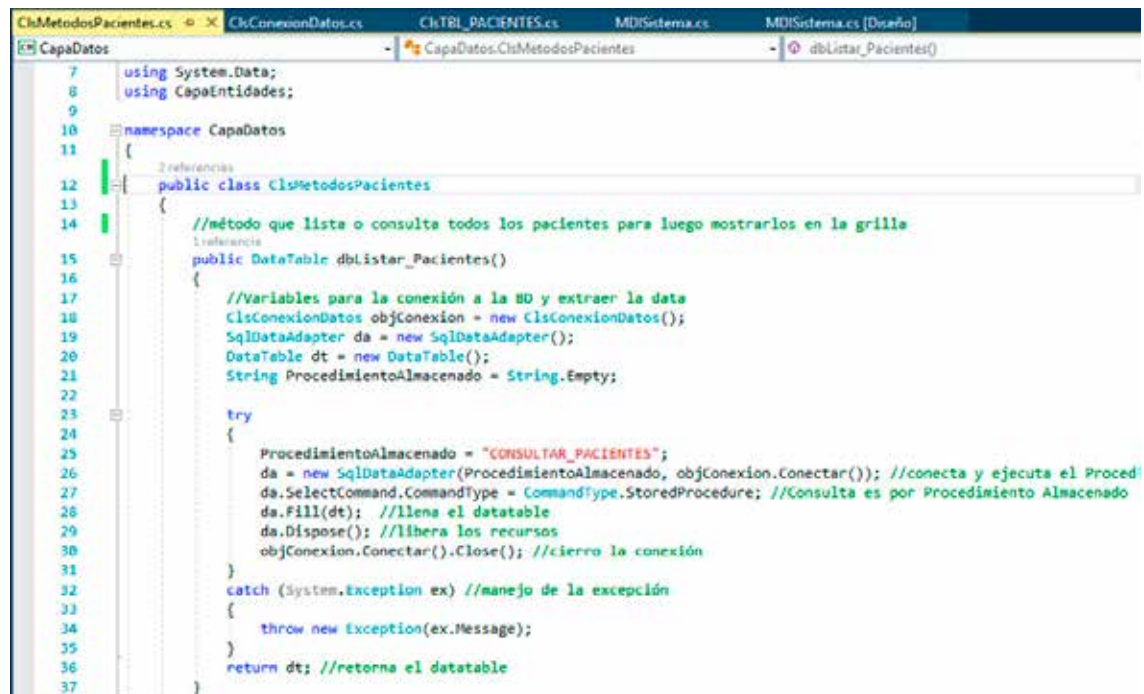
```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <configuration>
3      <startup>
4          <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
5      </startup>
6      <connectionStrings>
7          <add name="CnnCITAS" connectionString="Data Source=(local)\SQL2016;Initial Catalog=CITAS;Persist Security Info=True" />
8      </connectionStrings>
9      <appSettings>
10         <add key="PaisConfig" value="CO" />
11     </appSettings>
12 </configuration>

```

Figura 21. Archivo App.config.

Una porción de ejemplo de código de la CapaDatos para los métodos de la tabla Pacientes se muestra a continuación:



```

7  using System.Data;
8  using CapaEntidades;
9
10 namespace CapaDatos
11 {
12     public class clsMetodosPacientes
13     {
14         //método que lista o consulta todos los pacientes para luego mostrarlos en la grilla
15         public DataTable dbListar_Pacientes()
16         {
17             //Variables para la conexión a la BD y extraer la data
18             clsConexionDatos objConexion = new clsConexionDatos();
19             SqlDataAdapter da = new SqlDataAdapter();
20             DataTable dt = new DataTable();
21             String ProcedimientoAlmacenado = String.Empty;
22
23             try
24             {
25                 ProcedimientoAlmacenado = "CONSULTAR_PACIENTES";
26                 da = new SqlDataAdapter(ProcedimientoAlmacenado, objConexion.Conectar()); //conecta y ejecuta el Procedi
27                 da.SelectCommand.CommandType = CommandType.StoredProcedure; //Consulta es por Procedimiento Almacenado
28                 da.Fill(dt); //llena el datatable
29                 da.Dispose(); //libera los recursos
30                 objConexion.Conectar().Close(); //cierra la conexión
31             }
32             catch (System.Exception ex) //manejo de la excepción
33             {
34                 throw new Exception(ex.Message);
35             }
36             return dt; //retorna el datatable
37         }
38     }
39 }

```

Figura 22. Método listar pacientes.

- **Interfaz de usuario o UI (User Interface):** este es el proyecto principal (identificado porque su nombre está en negrilla) y en él se encuentra el formulario MDI llamado MDISistema, los formularios que componen la aplicación, el archivo App.config (descrito en la sección anterior) y se adicionó una carpeta llamada imágenes que puede contener archivos de imágenes e íconos utilizados dentro de la aplicación.

El formulario **FrmPacientes** incluye los siguientes objetos:

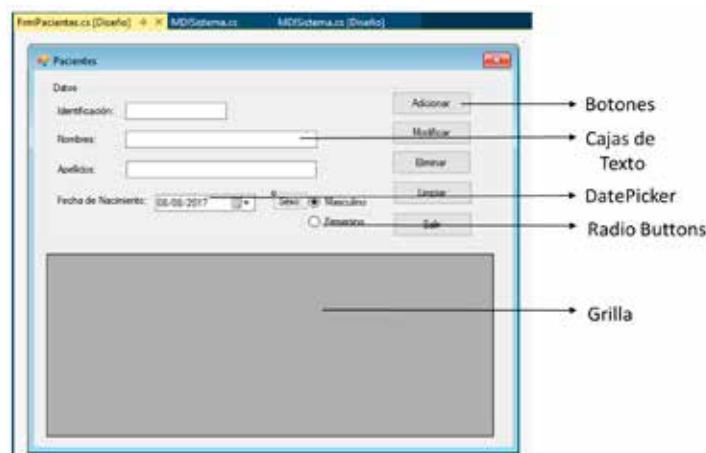


Figura 23. Objetos del formulario pacientes.



Al abrir el formulario (evento Load) se realiza consulta que trae todos los pacientes y se cargan en la grilla, como se muestra la imagen:

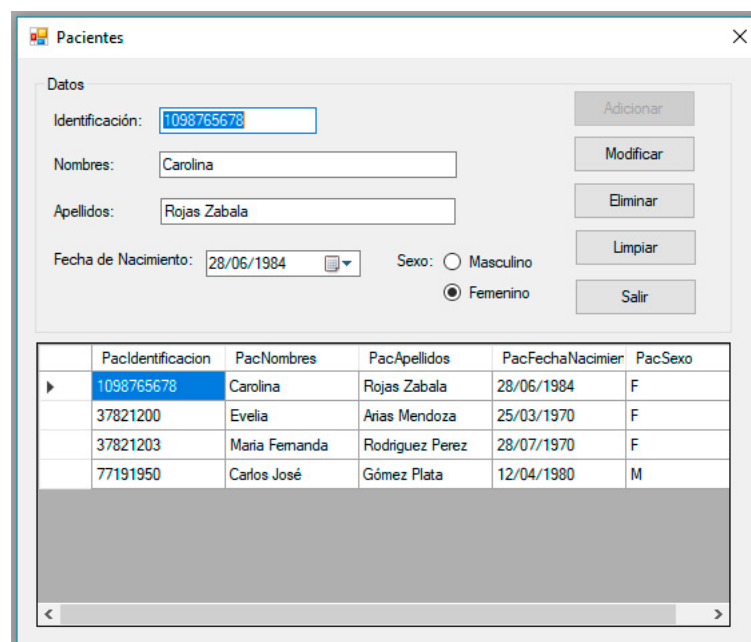
```

10 using CapaEntidades;
11 using CapaNegocios;
12
13 namespace SinTratamientos
14 {
15     #region Referencias
16     public partial class FrmPacientes : Form
17     {
18         //variables que conectan a las otras Capas
19         ClsReglasPacientes objReglasPacientes = new ClsReglasPacientes();
20         ClsTBL_PACIENTES objTblPacientes = new ClsTBL_PACIENTES();
21
22         #region Referencia
23         public FrmPacientes()
24         {
25             InitializeComponent();
26         }
27
28         #region Referencia
29         private void FrmPacientes_Load(object sender, EventArgs e)
30         {
31             //llenar la grilla
32             CargarDatosGrillaPacientes();
33         }
34
35         #region Referencia
36         public void CargarDatosGrillaPacientes()
37         {
38             DataTable dt = new DataTable();
39             dt = objReglasPacientes.ListarPacientes();
40
41             gridPacientes.DataSource = dt;
42         }
43     }
44 }

```

Figura 24. Formulario pacientes – Cargar grilla.

Formulario de Pacientes en ejecución:



**Pacientes**

Datos

Identificación:

Nombres:

Apellidos:

Fecha de Nacimiento:  Sexo: ☐ Masculino ☒ Femenino

Adicionar Modificar Eliminar Limpiar Salir

	PacIdentificacion	PacNombres	PacApellidos	PacFechaNacimier	PacSexo
▶	1098765678	Carolina	Rojas Zabala	28/06/1984	F
	37821200	Evelia	Arias Mendoza	25/03/1970	F
	37821203	Maria Fernanda	Rodriguez Perez	28/07/1970	F
	77191950	Carlos José	Gómez Plata	12/04/1980	M

Figura 25. Formulario pacientes en ejecución.

Los demás botones tienen su propia funcionalidad. A continuación, el código del botón Adicionar:

```
private void btnAdicionar_Click(object sender, EventArgs e)
{
    bool resultado;

    try
    {
        //validar datos de entrada
        if (txtIdentificacion.Text.Trim() == "" || txtNombres.Text.Trim() == "" || txtApellidos.Text.Trim() == "" || dtPickerFechaNac.Text.Trim() == "")
            MessageBox.Show(MdiParent, "Favor digitar datos", "Información", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        {
            //asigno los valores de las cajas de texto al objeto Entidad
            objTblPacientes.pacIdentificacion = txtIdentificacion.Text.Trim();
            objTblPacientes.pacNombres = txtNombres.Text.Trim();
            objTblPacientes.pacApellidos = txtApellidos.Text.Trim();
            objTblPacientes.pacFechaNacimiento = dtPickerFechaNac.Value;
            if (rbtnSexoMasc.Checked == true)
                objTblPacientes.pacSexo = "M";
            else
                objTblPacientes.pacSexo = "F";

            resultado=objReglasPacientes.InsertarActualizarPacientes(objTblPacientes);
            if (resultado==true)
            {
                MessageBox.Show(MdiParent, "Registro adicionado", "Información", MessageBoxButtons.OK, MessageBoxIcon.Information);
                CargarDatosGrillaPacientes();
            }
            else
                MessageBox.Show(MdiParent, "Registro No adicionado", "Información", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    catch (System.Exception ex) //manejo de la excepción
    {
    }
}
```

Figura 26. Código botón adicionar paciente.

Primero se realiza la validación de que las cajas de texto tengan valores (no vacíos), luego se llenan los campos del objeto objTblPacientes de la Capa Entidades y se procede a ejecutar el método InsertarActualizarPacientes de la Capa de Negocios llevando como parámetro el objeto objTblPacientes.

```
namespace CapaNegocios
{
    2 referencias
    public class ClsReglasPacientes
    {
        ClsMetodosPacientes objMetodosPacientes = new CapaDatos.ClsMetodosPacientes();

        1 referencia
        public DataTable ListarPacientes()
        {
            return objMetodosPacientes.dbListar_Pacientes();
        }

        2 referencias
        public bool InsertarActualizarPacientes(ClsTBL_PACIENTES objTblPacientes)
        {
            bool resultado;
            resultado = objMetodosPacientes.dbInsertar_Actualizar_Pacientes(objTblPacientes);
            return resultado;
        }

        0 referencias
        public DataTable ConsultarPacientesIdentificacion(String IdentificacionPac)
        {
            DataTable dt = new DataTable();
            dt = objMetodosPacientes.dbConsultar_Pacientes_Identificacion(IdentificacionPac);
            return dt;
        }
    }
}
```

Figura 27. Capa de negocios de pacientes.

La capa de Negocios ejecuta el método `dbInsertar_Actualizar_Pacientes` que se encuentra en la capa de Datos llamado `ClsMetodosPacientes`:

```
public bool dbInsertar_Actualizar_Pacientes(ClsTBL_PACIENTES objTblPacientes)
{
    //Variables para la conexión a la BD y extraer la data
    ClsConexionDatos objConexion = new ClsConexionDatos();
    SqlCommand comando = new SqlCommand();
    String ProcedimientoAlmacenado = String.Empty;

    try
    {
        //parámetros del objeto Comando
        ProcedimientoAlmacenado = "INSERTAR_ACTUALIZAR_PACIENTES";
        comando.Connection = objConexion.Conectar();
        comando.CommandType = CommandType.StoredProcedure;
        comando.CommandText = ProcedimientoAlmacenado;
        comando.CommandTimeout = 10;

        //adiciona los parámetros
        comando.Parameters.AddWithValue("@PacIdentificacion", objTblPacientes.pacIdentificacion);
        comando.Parameters.AddWithValue("@PacNombres", objTblPacientes.pacNombres);
        comando.Parameters.AddWithValue("@PacApellidos", objTblPacientes.pacApellidos);
        comando.Parameters.AddWithValue("@PacFechaNacimiento", objTblPacientes.pacFechaNacimiento);
        comando.Parameters.AddWithValue("@PacSexo", objTblPacientes.pacSexo);

        if (comando.ExecuteNonQuery() > 0) //se ejecuta la consulta
            return true;
        else
            return false;
    }
    catch (System.Exception ex) //manejo de la excepción
    {
        throw new Exception(ex.Message);
    }
}
```

Figura 28. Método insertar actualizar pacientes.

Este método realiza la conexión a la base de datos, invoca al procedimiento almacenado "Insertar\_Actualizar\_Pacientes" enviando todos los parámetros con sus valores (del objeto `objTblPacientes` de la capa Entidad).

Por último, ejecuta este procedimiento almacenado por medio de la sentencia `ExecuteNonQuery` y devuelve si la operación fue exitosa.

**Nota:** se adiciona código fuente de la solución Windows Forms con los ejemplos expuestos y un backup de la base de datos CITAS en Microsoft SQL Server. (Para descargar los archivos consulte la versión animada de este material.)





## 8. Creación de instalador en aplicación Windows Forms

En la versión de Visual Studio 2017, los proyectos de setup o instaladores se deben descargar como una extensión de la herramienta de desarrollo.

Ir al menú Herramientas – Extensiones y actualizaciones y buscar como “Setup”

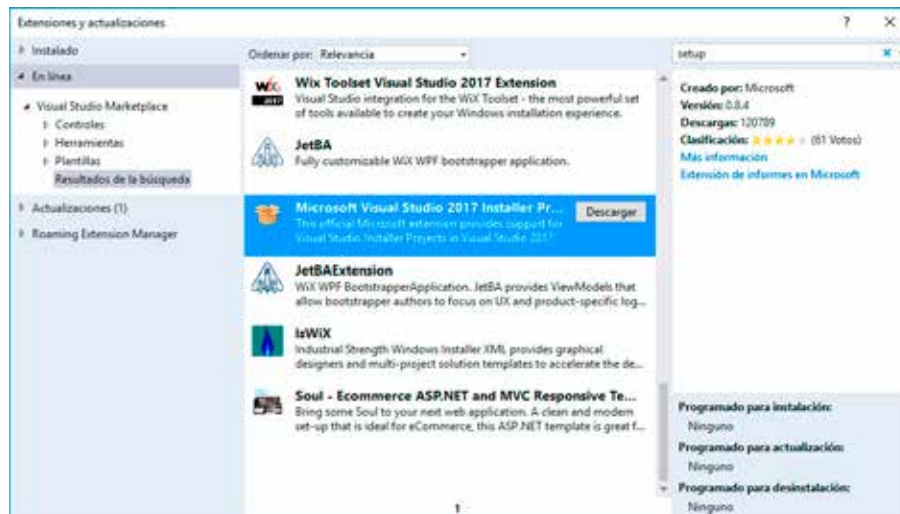


Figura 29. Instalación de extensión Visual Studio 2017 Installer.

Después de realizar la instalación, se debe cerrar la solución y reiniciar Visual Studio.

### Pasos para crear un nuevo proyecto de instalación

En el menú Archivo, elija Agregar > Nuevo proyecto. En el cuadro de selección de *Nuevo proyecto* ubicar *Otros tipos de proyectos*, y seleccionar *Instalador de Visual Studio o Visual Studio Installer*.

Seleccionar *Setup Project Wizard* como proyecto de instalación.

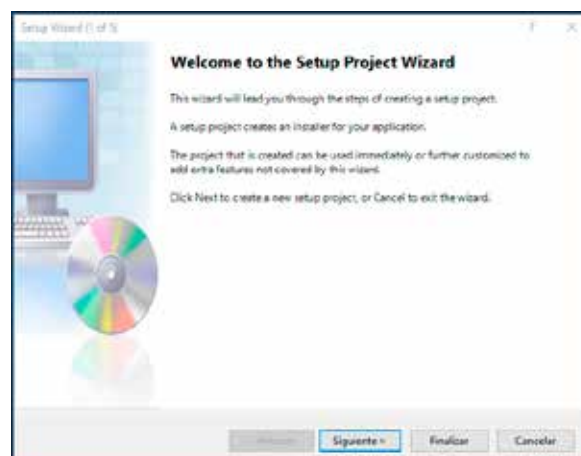


Figura 30. Bienvenida al asistente de instalación.

Al dar siguiente, el Wizard o Asistente pregunta el tipo de instalación:

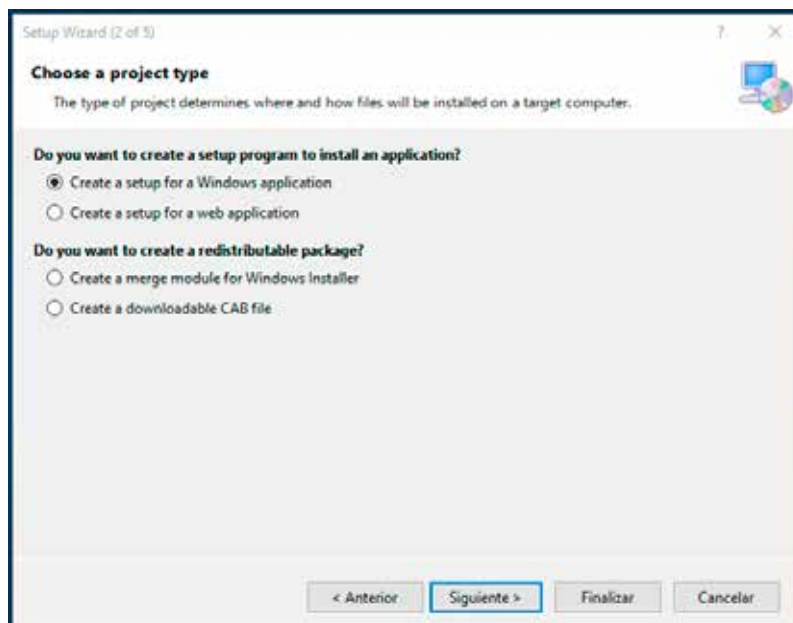


Figura 31. Seleccionar el tipo de instalación.

Luego se debe seleccionar que va a incluir el instalador (en este paso se selecciona el resultado principal de la solución que incluye los archivos .dll generados al compilar las clases y el archivo .exe al compilar el programa principal).

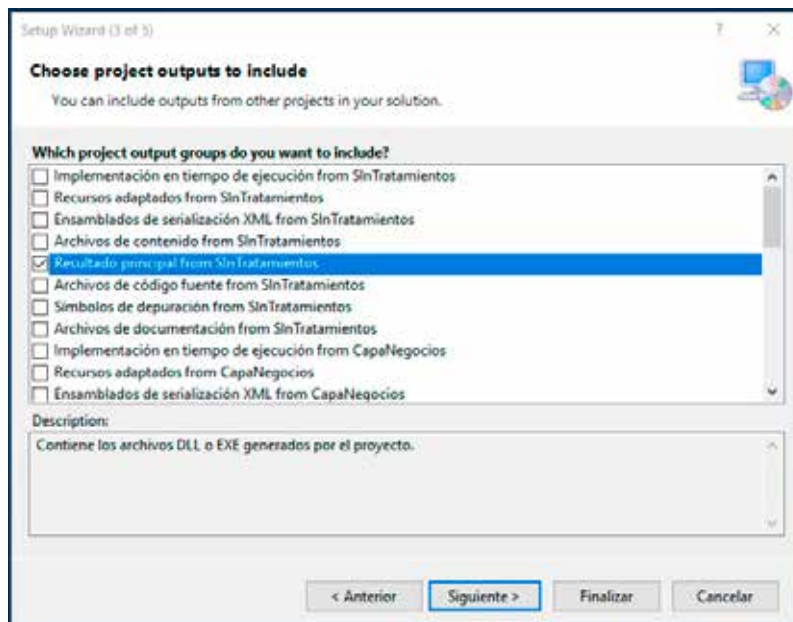


Figura 32. Paquetes de instalación.

Al finalizar, el asistente muestra el resumen del instalador y la pantalla se actualiza para definir las opciones del instalador.

El proyecto de instalación muestra las siguientes opciones:

- **Application Folder (carpeta de la aplicación):** esta carpeta contiene los archivos .exe y .dll generados en la aplicación.
- **User Desktop (escritorio del usuario):** por ejemplo, el acceso directo a la aplicación con su respectivo ícono.
- **User's Programs Menu (menú de programas de usuario):** carpeta que aparece en todos los programas.

En Application Folder > Adicionar > Resultados del proyecto asegurándose que esté seleccionado el proyecto principal, como se ilustra en la siguiente imagen:

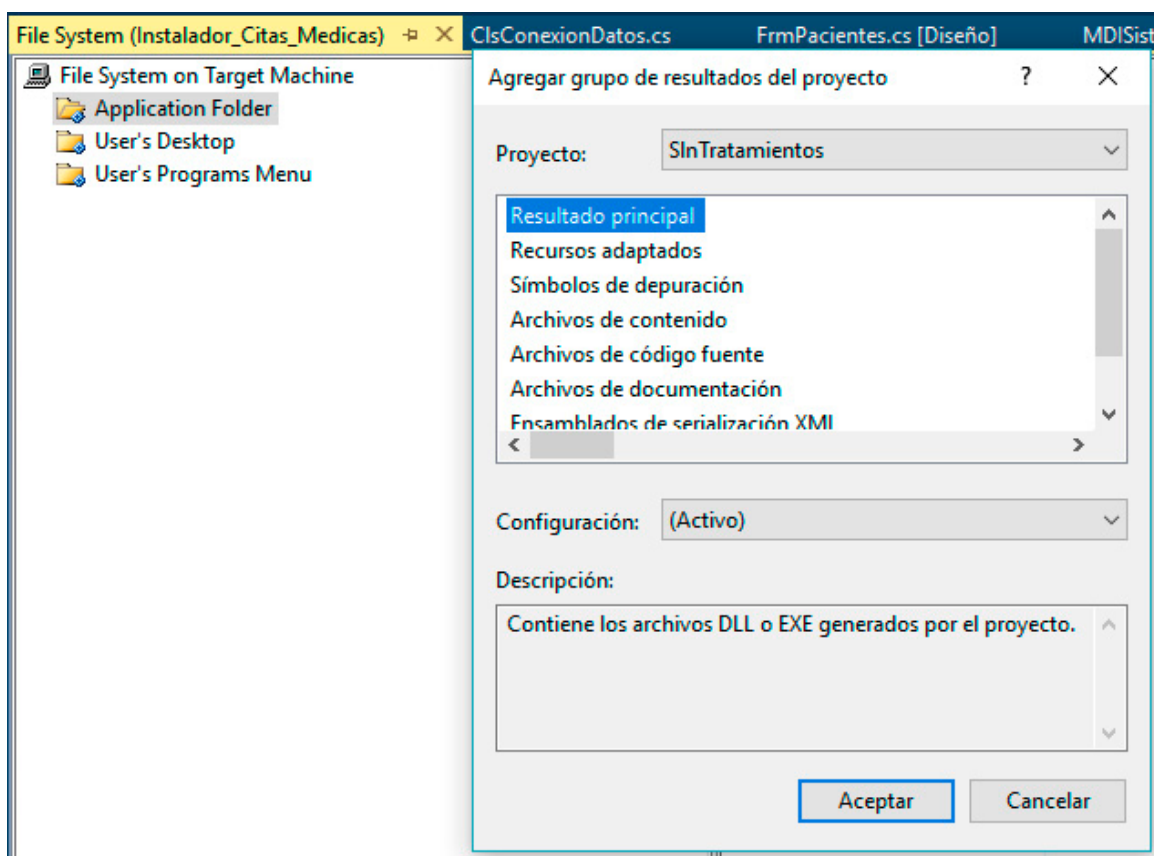


Figura 33. Instalador – Resultado principal.

De esta manera, el proyecto instalador selecciona el resultado de cada Capa y el del Proyecto Principal, quedando la siguiente información:

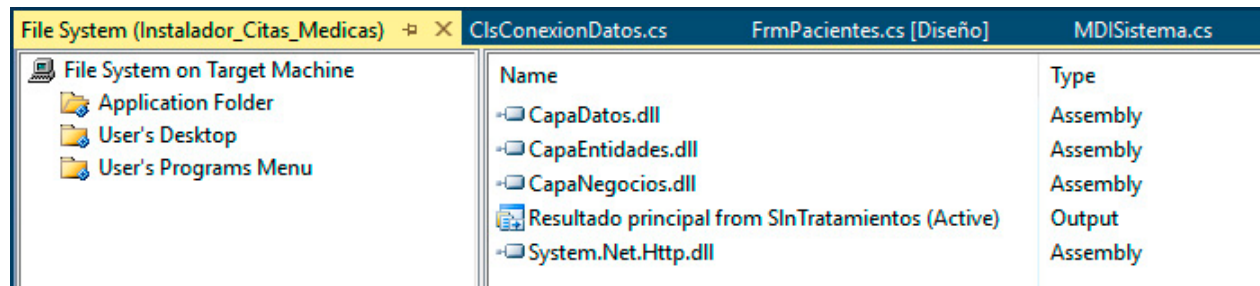


Figura 34. Resultados del proyecto.

Luego se selecciona “User’s Desktop” (carpeta escritorio del usuario) y en la parte central de la pantalla, botón derecho “Crear acceso nuevo acceso directo” tal como se muestra en la imagen:

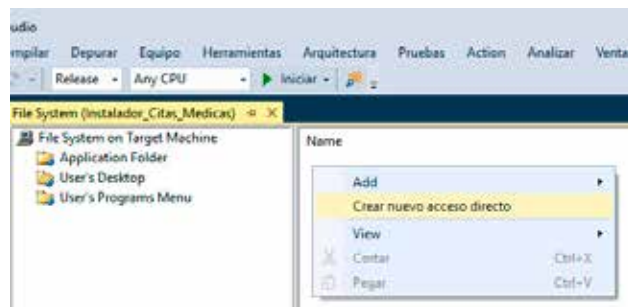


Figura 35. Crear acceso directo a la aplicación Windows.

En la ventana siguiente, escoger Application Folder (carpeta de la aplicación) – doble clic y seleccionar resultado principal de la aplicación y aceptar. Acá se pueden editar las propiedades del acceso directo, como su nombre y el ícono a mostrar.

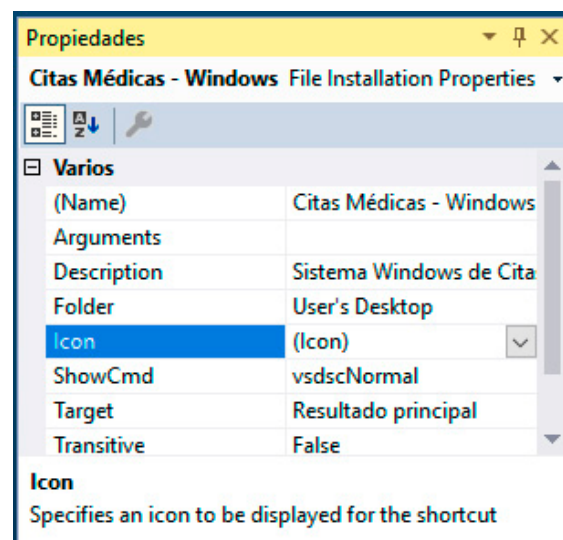


Figura 36. Propiedades del escritorio de usuario.

En la opción User's Programs Menu (menú de programas de usuario): sobre esta opción- botón derecho > Adicionar > Folder (Carpeta) y se le otorga el nombre de Citas Médicas (con este nombre quedará en "Todos los programas" del usuario final). Ahora se procede a crear el acceso directo: se selecciona la carpeta recién creada de "Citas Médicas" y en la parte central de la pantalla > botón derecho > Crear nuevo Acceso directo y ejecutar las mismas opciones de la creación del acceso directo.

En el proyecto de instalación creado – botón derecho - propiedades, y se configuran los prerequisites de instalación y el nombre del archivo instalador, como se ilustra en la siguiente imagen:

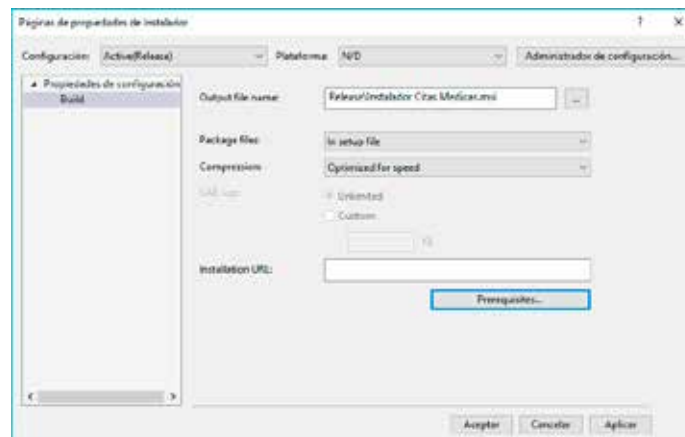


Figura 37. Propiedades del instalador.

Seleccionar el botón de los prerequisites con el fin de decirle al instalador que en caso de que no exista el Framework en el equipo de destino, se pueda descargar e instalar en el equipo.

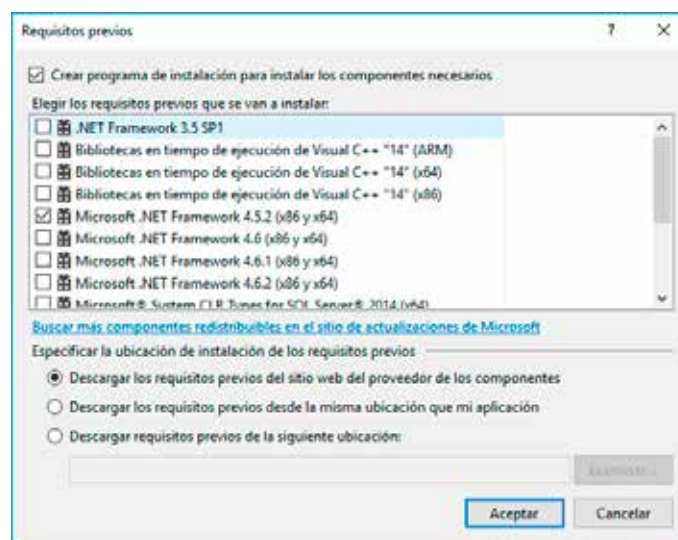


Figura 38. Prerrequisitos de instalación.

Luego se procede a compilar el proyecto de instalación. Se selecciona el Proyecto de instalación > botón derecho > Compilar.

Esta opción genera en la carpeta del instalador los archivos de instalación.

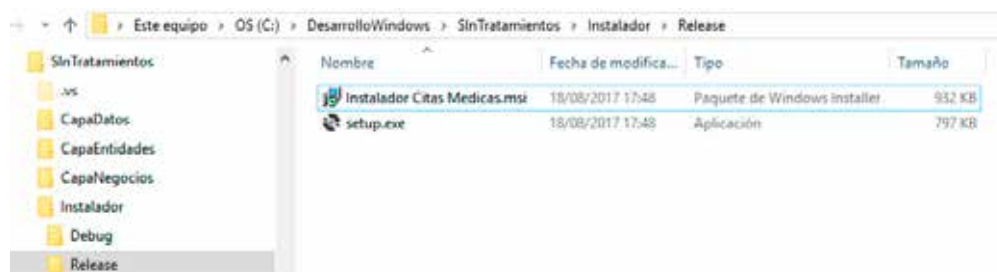


Figura 39. Archivos de instalación generados.

Al ejecutar el instalador (doble clic sobre el archivo setup.exe), el sistema genera:

Carpeta con los archivos resultados de la solución.	
Acceso directo en el escritorio.	
Carpeta en todos los programas.	



## Glosario

**POO- Programación Orientada a Objetos:** tipo de programación especial orientada a situaciones reales, donde se realizan los programas en términos de objetos, métodos y propiedades. Por ejemplo, el objeto Vehículo es el elemento principal que tiene una serie de características, como podrían ser la marca, modelo, color y que tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o parquear.

**.NET Framework:** proporciona una biblioteca de código probado y reutilizable para el desarrollo de aplicaciones. La biblioteca de clases de .NET es una biblioteca orientada a objetos que permite realizar tareas habituales de programación como funciones de cadenas (strings), recolección de datos, conectividad de bases de datos, acceso a archivos, entre otras funciones.

## Bibliografía

Bell, D. (2010). *C# para Estudiantes*. México: Pearson Educación.

Canchala, Luis. (2007). *Fundamentos de la POO*. Recuperado de <https://msdn.microsoft.com/es-es/library/bb972232.aspx>

Microsoft. (2007). *Arquitectura de ADO.NET*. Recuperado de <https://goo.gl/2TyJY1>

Microsoft. (2007). *Guía de programación de C#*. Recuperado de <https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/index>

Microsoft. (2007). *Referencia de C#*. Recuperado de <https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/index>



## Control del documento

### CONSTRUCCIÓN OBJETO DE APRENDIZAJE



#### DESARROLLO DE APLICACIONES WINDOWS EN C# USANDO VISUAL STUDIO .NET

Centro Industrial de Mantenimiento Integral - CIMI  
Regional Santander

**Líder línea de producción:** Santiago Lozada Garcés

**Asesores pedagógicos:** Rosa Elvia Quintero Guasca  
Claudia Milena Hernández Naranjo

**Líder expertos temáticos:** Rita Rubiela Rincón Badillo

**Experto temático:** Edgar Eduardo Vega Arango

**Diseño multimedia:** Eulises Orduz Amezcuita

**Programador:** Francisco José Lizcano Reyes

**Producción de audio:** Víctor Hugo Tabares Carreño  
Martha Lucía Chaves Niño

Este material puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de la licencia que el trabajo original.



**creative  
commons**

BY NC SA

Visual Studio © Reservados todos los derechos. Visual Studio es una marca registrada de Microsoft Corporation en los Estados Unidos y/o en otros países.



**Registered trademark**