Nombre: Jorge Daniel Lopez lopez

Inicio de capacitación: 06/06/2022

Fin de capacitación:

Puesto: Programador Jr

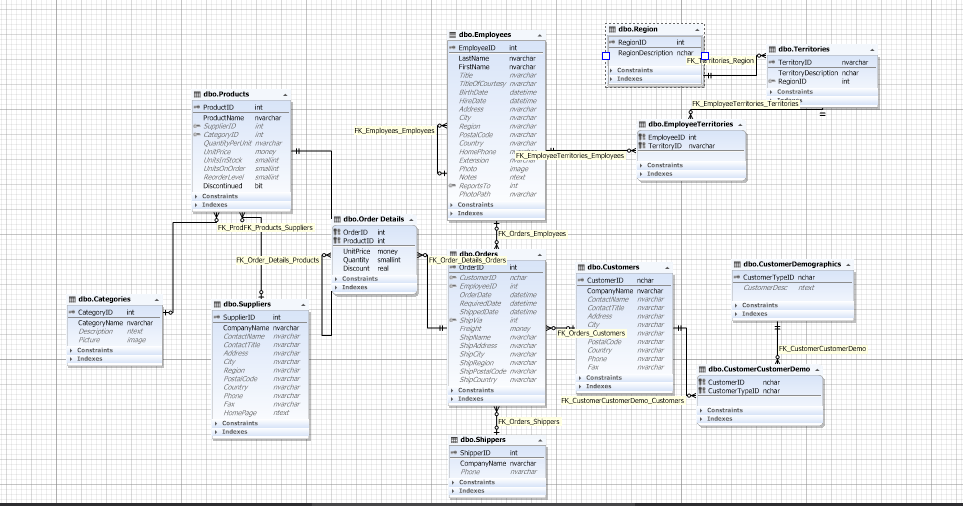
SQL

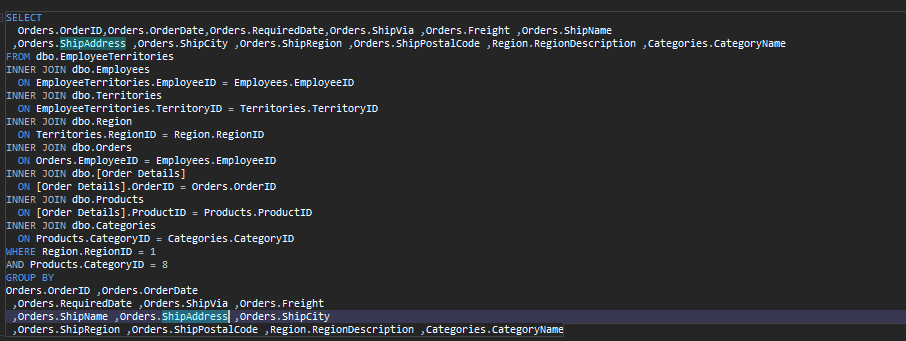
T-SQL

Transact-SQL es una extensión al SQL de Microsoft y Sybase. SQL Son las transacciones a la base de datos, para generar búsquedas especiales, alterar datos, modificarlos. Declarando sentencias específicas.

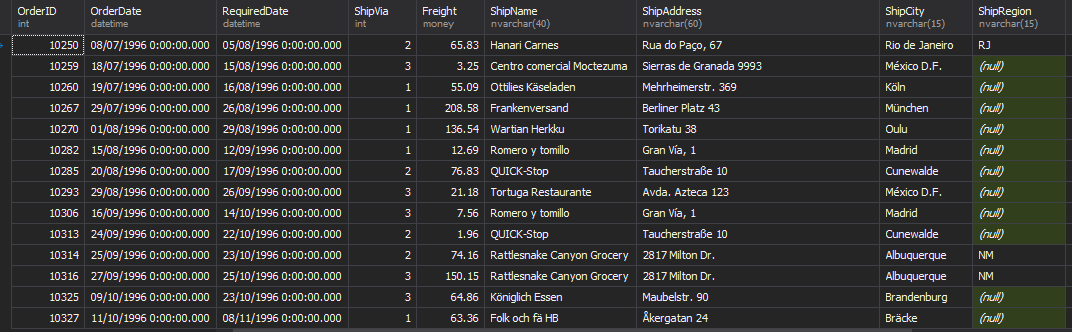
Para realizar algunos ejemplos, descargamos una BD, que es con la que estaremos trabajando, para demostrar como se usan los t-sql.

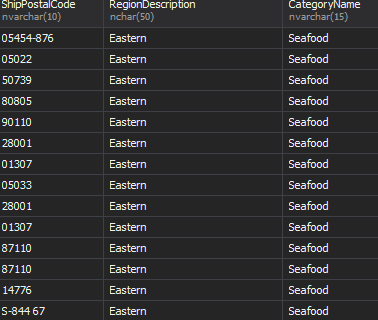
Agregamos nuestra base de datos, y nos quedará un diagrama que se muestra de la siguiente forma:

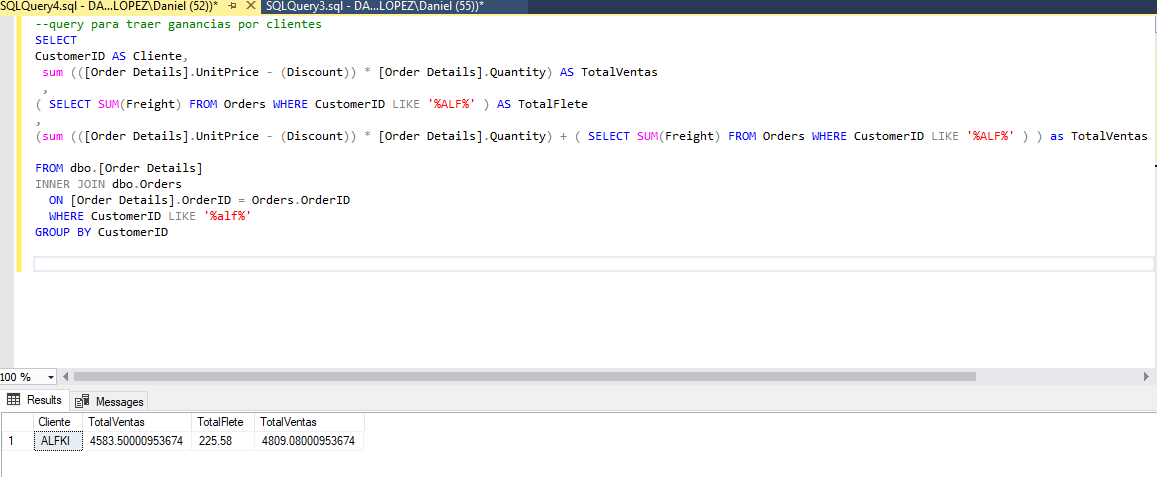


Para ejecutar una búsqueda de ordenes que hayan sido de empleados de Eastern y la categoría “Seafood” quedaría de la siguiente manera:

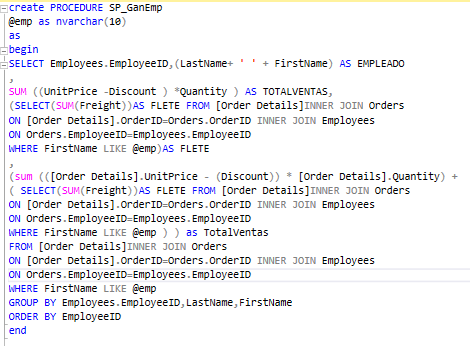
Lo cual nos mostrará un resultado de esta manera



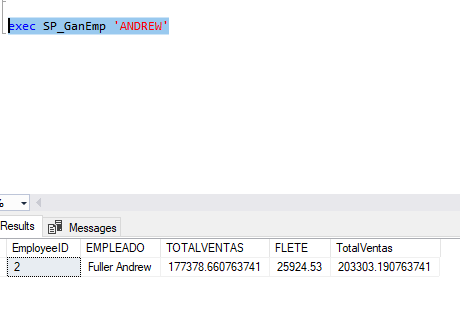


Para hacer una búsqueda especifica, tal como buscar ganancias por clientes, realizamos la siguiente consulta, la cual da el siguiente valor como resultado

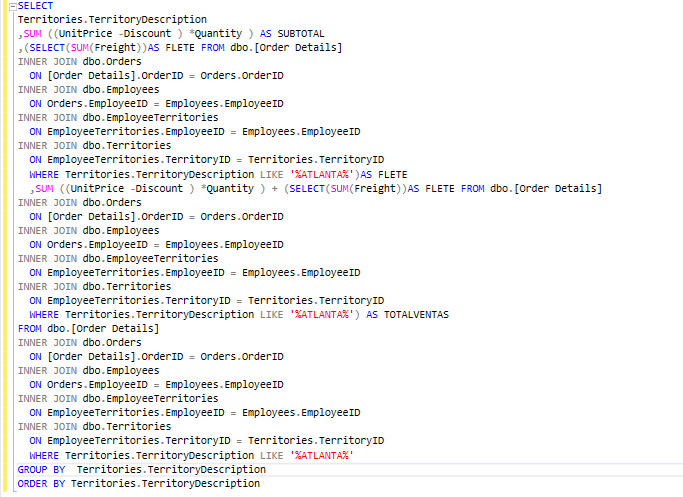
Y por empleados quedaría de la siguiente manera:



Creando un procedimiento almacenado que hará que haga la consulta con el siguiente código: esta vez buscamos con el empleado andew

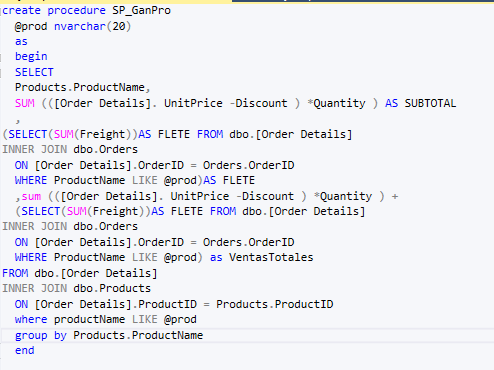


Por territorio y buscamos que nos dé una búsqueda relacionada como Atlanta

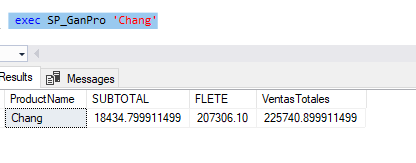
resultado



Por producto



El cual hacemos nuevamente un procedimiento almacenado para encontrar algo como deseemos, ejecutamos y nos dará el siguiente resultado:

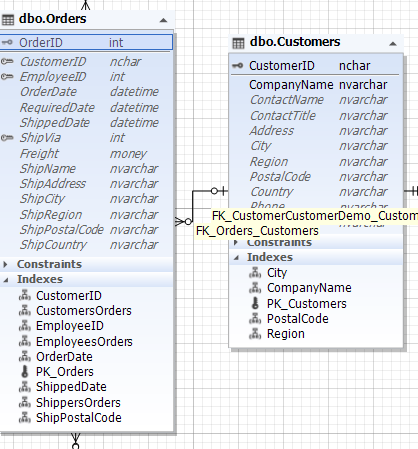


De esta manera podemos ejemplificar las diferentes formas de poder usar los t-sql

Indices sql

Se emplea un índice SQL para poder recuperar datos de una base de datos de una manera más rápida. El indexar una tabla o la vista es sin lugar a dudas, una de las mejores opciones de poder mejorar el rendimiento de las consultas y aplicaciones.

Un índice SQL es una tabla de búsqueda rápida para poder encontrar los registros que los usuarios necesitan buscar con mayor frecuencia. Ya que un índice es pequeño, rápido y optimizado para búsquedas rápidas. Además, que son muy útiles para conectar las tablas relacionales y la búsqueda de tablas grandes.

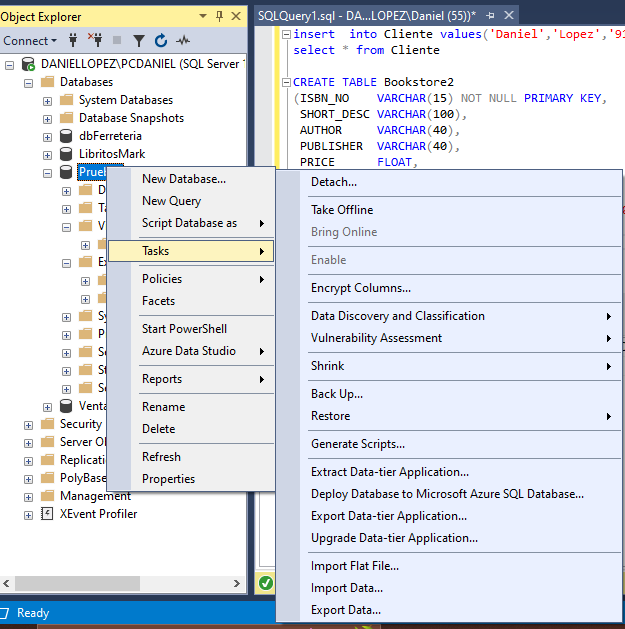
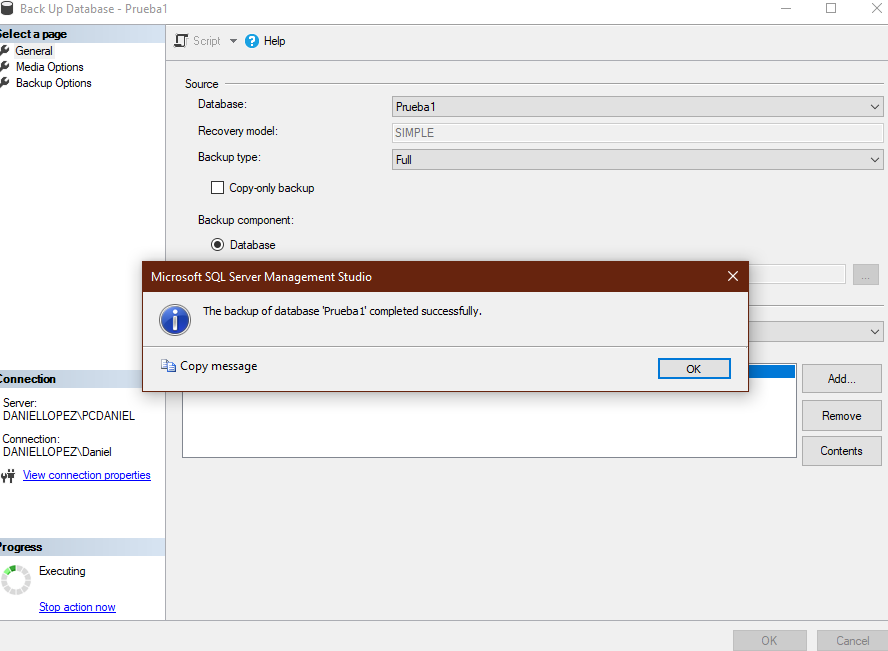
Dentro de los cuales existen las llaves PK, y FK, que son los conectores para enlazar nuestras tablas, según haya sido la relación del diagrama. Mostraré algunas

En este caso notamos que tipo de índices tiene cada tabla.

Respaldos SQL

Las copias de seguridad de SQL Server proveen una importante solución para proteger datos críticos que están almacenados en bases de datos SQL. Y para minimizar el riego de pérdida de datos, usted necesita asegurarse de que respalda sus bases de datos regularmente tomando en consideración los cambios aplicados a sus datos.

Antes de programar el trabajo de copias de seguridad, es necesario tener una estimación de cuánto espacio de disco será usado por la copia de seguridad completa de la base de datos. También, es necesario tener una estimación inicial del incremento en el tamaño de la base de datos, porque cuando este se incrementa las copias de seguridad completas requerirán más espacio de almacenamiento.



Es importante ver cuál de todas las BD que tenemos, es la que le haremos el bakcup.

ORM

ORM, por sus siglas al inglés: Object Relational Mappper, no es más que una pieza de software que nos permite interactuar con nuestra base de datos sin la necesidad de conocer SQL (El lenguaje de consultas). Todo esto utilizando el paradigma de programación orientada a objetos.

Existen algunos frameworks que ayudan al uso de ORM, tales como:

* Active Record
* Eloquent
* Peewee
* SQLAlchemy
* Entity Framework
* Hibernate

Los ORMs se encarga de traducir nuestra instrucción en el lenguaje de programación que estemos utilizando a una sentencia SQL que el gestor de base de datos pueda entender.

Utilizando una base de datos relacional, MySQL por ejemplo. Nuestra consulta quedaría de la siguiente manera:

SELECT id, username, email FROM id = 101:

Una sentencia bastante sencilla, Ahora, Utilizando el active record de ruby on rails, la consulta pudiera quedar de la siguiente manera.

User.where(id:101).select(:id, :username, :email)

Es de esta forma como nos podemos dar cuenta de la diferencia entre usar SQL, y un ORM, uno lleva su propia clase, su propio método y atributos

EntityFrameWork

Entity Framework (EF) es la solución open-source que ofrece Microsoft para un ORM en las aplicaciones .NET. Este framework permite a los desarrolladores convertir sus estructuras de datos en clases para poder trabajar con esa información usando objetos de .NET, reduciendo considerablemente la cantidad de código que sería necesario para realizar las tareas de acceso y gestión de datos en nuestra aplicación.

Lo cual, este mismo presenta algunas características que son importantes mencionar:

* **Cross-platform:** EF Core es un framework que funciona libremente en Windows, Linux y Mac.
* **Modelado:** EF genera un EDM (Entity Data Model o Modelo de Datos de Entidad) con entidades de distintos tipos de datos, los cuales podemos asignar o recuperar (get y set). Estos modelos son lo utilizados para realizar las consultas y persistencia en base de datos
* **Transacciones:** EF usa transacciones por defecto al realizar consultas a la base de datos. Adicionalmente permite modificar la gestión de transacciones en caso de necesitar algo más específico.
* **Cache:** EF incluye una cache que previene sobrecargas de la base de datos en caso de recibir una misma consulta numerosas veces.
* **Configuración:** EF permite ajustar los modelos generados añadiendo Data Annotations para sobrescribir el comportamiento por defecto.
* Migraciones: EF incluye comandos para generar y ejecutar migraciones, para modificar el esquema de la base de datos.
* Consultas: EF permite usar consultas LINQ como opción alternativa para consulta de datos. El gestor de la base de datos se encargará de traducir las consultas LINQ al lenguaje con que trabaje nuestra base de datos (SQL por ejemplo). Por último, EF también permite ejecutar consultas nativas o “raw” directamente.

SQL-LOG

La transacción de log, es una parte vital de una base de datos, ya que en ella se guarda cada una de las transacciones que se hacen a la base de datos.

 log de transacciones es el que asegura que una transacción tenga la propiedad ACID (**A**tomicity, **C**onsistency, **I**solation and **D**urability), debido a que cuando se modifica un dato en la base de datos, lo primero que se hace para asegurar que la transacción ha sido confirmada es escribir en el log de transacciones los cambios realizados por la transacción, una vez realizado esto recién se puede decir que la transacción esta completada, esta es la única manera de asegurar que una transacción sea durable en el tiempo.

El log de transacciones se maneja dependiendo del modelo de recuperación que tenga configurado la base de datos.

Hay que tener en claro que los registros dentro de log de transacciones manejan estados (activo / inactivo), de esta manera se identifica los registros que ya no son útiles y se pueden sobrescribir, o de los cuales se pueden reclamar espacio en disco. Si no hay espacio libre o registros inactivos para que el log de transacciones pueda escribir sobre ellos, entonces el este crecerá un determinado número de bytes, y como ya hemos explicado este crecimiento es costoso en términos de performance. De esto podemos derivar la conclusión de que el log de transacciones tiene un comportamiento cíclico, es decir que cuando llega al final de archivo buscara espacio libre al inicio para poder seguir escribiendo las transacciones de la base de datos.

Al momento de tener en cuenta el crecimiento del log, es bueno asignar un tamaño fijo, considerando que tanto puede crecer según sean las transacciones a la BD.

El método de las escrituras es de forma secuencial, es decir, va en un orden secuencial no es como un archivo de datos de una base de datos que escribe la data en forma aleatoria según la ubicación de las páginas de datos. Por eso mismo es más rápido en cuanto a escritura de disco, concierne porque la aguja del disco duro no debe moverse aleatoriamente, sino que esta solo debe moverse en una dirección hasta que se llegue al final del archivo y se necesite aplicar el comportamiento cíclico del archivo y se regrese al inicio del archivo para seguir ingresando transacciones.

DBForge

dbForge Studio es un entorno integrado para el desarrollo, gestión, administración, creación de informes y análisis de datos de SQL Server, MySQL y MariaDB, Oracle, PostgreSQL. SQL Manager permite crear, editar, copiar, adjuntar, separar, respaldar y restaurar bases de datos fácilmente de un servidor a otro.

Es una herramienta que, de mi punto de vista objetivo, es muy ligero para el uso, de fácil manejo, útil para realizar cualquier tipo de sucesos a la BD.

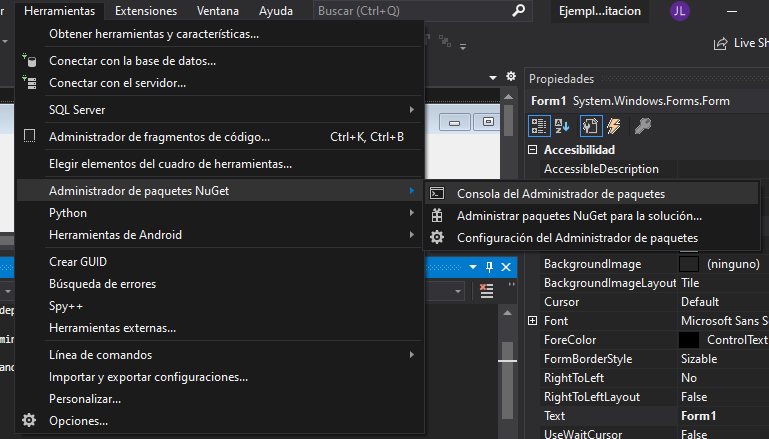
SDS

Es un IDE para hacer de forma más práctica el uso de nuestra base de datos, contiene diferentes tipos de funciones, visualizaciones que harán más fáciles los procedimientos que se vaya a realizar. Es un entorno totalmente amigable, intuitivo para poder realizar los trabajos correspondientes.

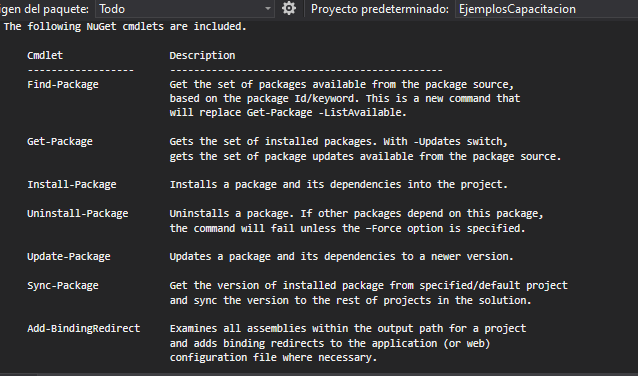
VISUAL ESTUDIO

Paquetes NUGET

Los paquetes de NuGet son unidades de código reutilizable que otros desarrolladores ponen a su disposición para que los use en sus proyectos. Los paquetes se instalan en un proyecto de Visual Studio mediante el Administrador de paquetes NuGet, la [Consola del Administrador de paquetes](https://github.com/NuGet/docs.microsoft.com-nuget.es-es/blob/live/docs/consume-packages/install-use-packages-powershell.md) o la [CLI de dotnet](https://github.com/NuGet/docs.microsoft.com-nuget.es-es/blob/live/docs/quickstart/install-and-use-a-package-using-the-dotnet-cli.md).



Para obtener algunos comandos del NuGet, es necesario escribir “get-help NuGet”, el cual nos dará una lista como la siguiente:



De acuerdo al paquete que vayamos a requerir será la instrucción que vamos a escribir.

Es una herramienta esencial para cualquier plataforma de desarrollo moderno, es un mecanismo a través del cual los desarrolladores pueden crear, compartir y consumir código útil. A menudo, este código se integra en "paquetes" que contienen código compilado (como archivos DLL) y otro contenido necesario en los proyectos que utilizan estos paquetes.

En .NET (incluido .NET Core), el mecanismo compatible con Microsoft para compartir código es NuGet, que define cómo se crean, hospedan y consumen paquetes en .NET, y ofrece las herramientas para cada uno de esos roles.

Desde un punto de vista sencillo, un paquete NuGet es un archivo ZIP con la extensión. nupkg que contiene código compilado (archivos DLL).

Los programadores con código para compartir crean paquetes y los publican en un host público o privado. Los consumidores de paquetes obtienen esos paquetes de los hosts adecuados, los agregan a sus proyectos y, después, llaman a la funcionalidad de un paquete en el código del proyecto. Después, el propio NuGet controla todos los detalles intermedios.

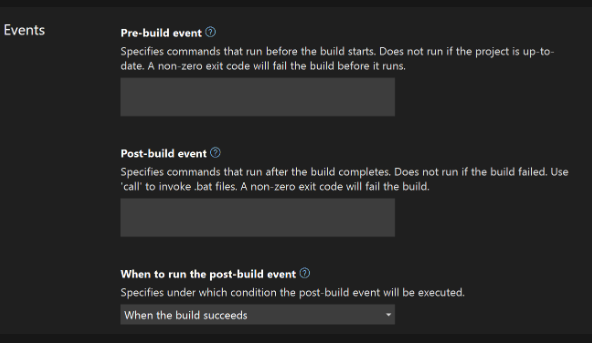
Evento PreBuild y PostBuild

Esto se refiere a que se puede ejecutar una sentencia antes de que este se compile, por ejemplo, puede ejecutar un archivo .bat antes de que comience una compilación o copiar nuevos archivos a una carpeta después de que se complete la compilación. Los eventos de compilación se ejecutan solo si la compilación alcanza con éxito esos puntos en el proceso de compilación.

Los eventos de compilación siguen la misma sintaxis que los comandos de DOS

Se tienen que usar eventos de compilación para especificar comandos que se ejecutan antes de que comience la compilación o después de que finalice.

Los eventos previos a la compilación no se ejecutan si el proyecto está actualizado y no se activa ninguna compilación.

En la sección Cuándo ejecutar el evento posterior a la compilación, especifique bajo qué condiciones ejecutar el evento posterior a la compilación.

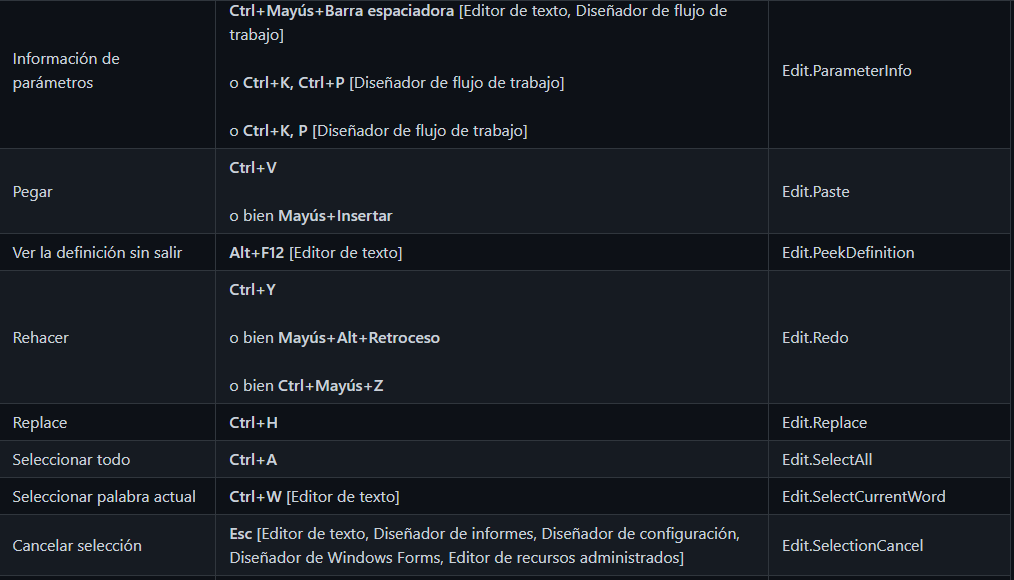
De esta manera podemos ver como se puede construir

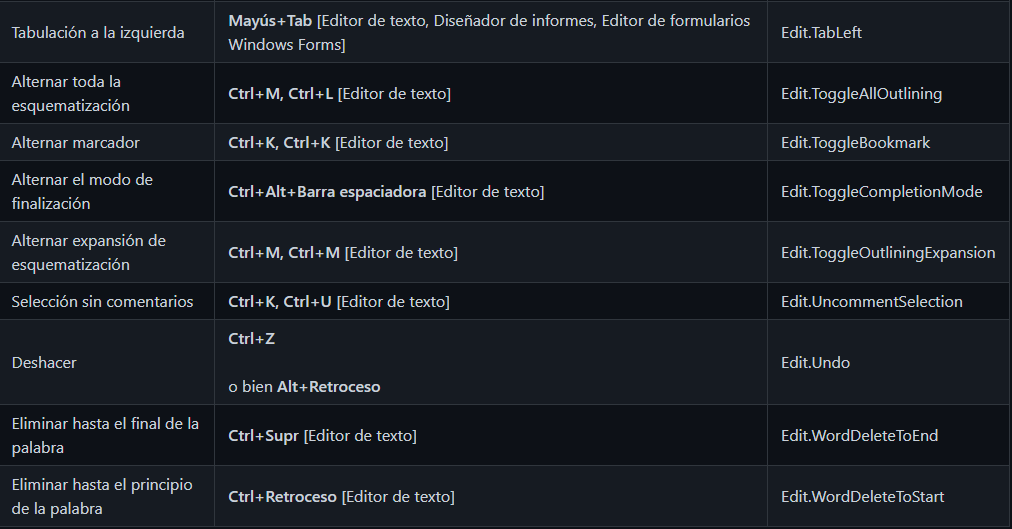
Shorcuts

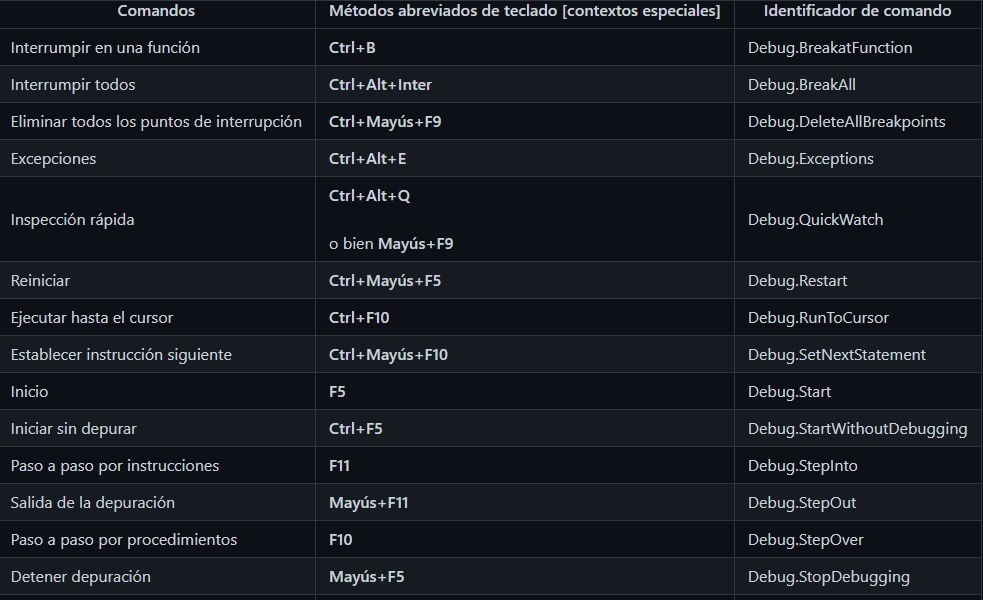
Es aquello que nos brindará los atajos para hacer algunos procesos en menos tiempo, lo cual dejaré una lista de lo que podemos usar:

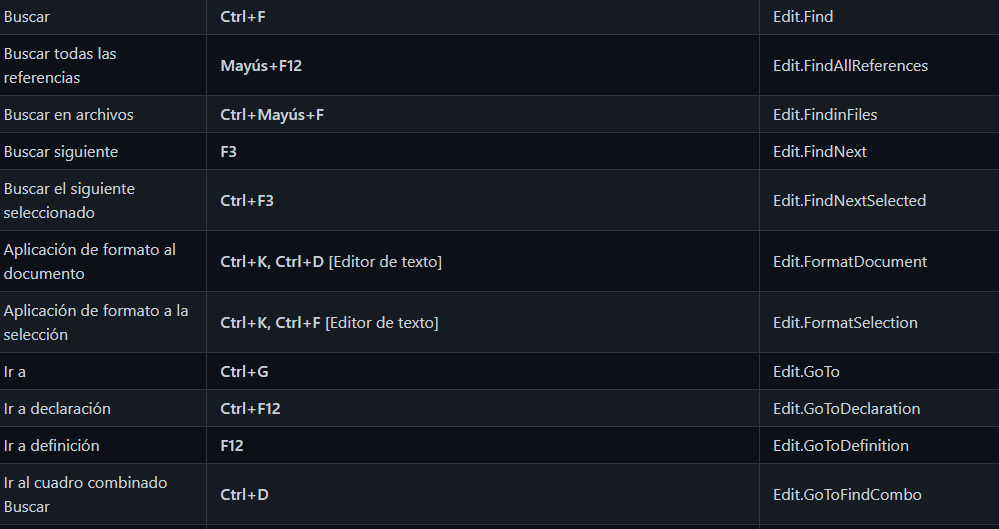










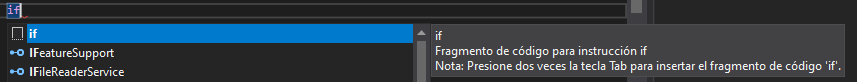




Fragmentos de código

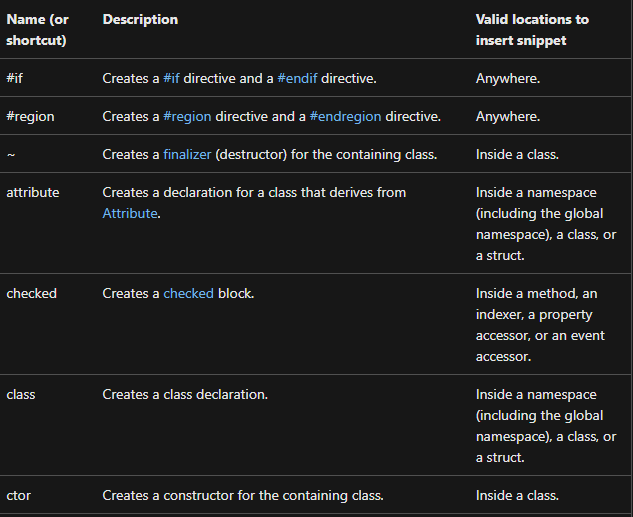
Estos fragmentos de código nos ayudarán a formar los códigos más rápido, solo basta dar dos veces “tab”, precedido de una parte de un código, y de forma automática nos dará lo que necesitamos, algunos ejemplos;

Al escribir la sentencia if

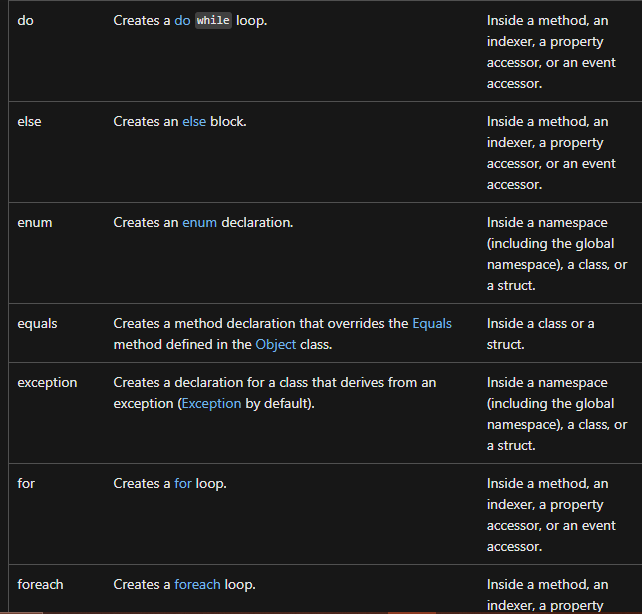


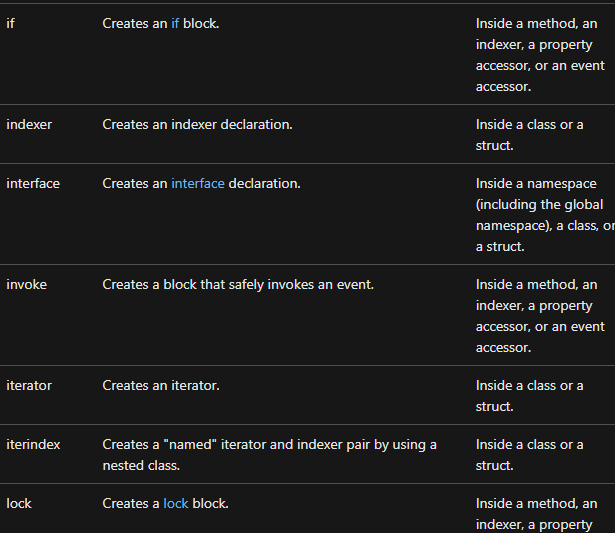
Y menciona que, si presionamos dos veces la tecla Tab, pondrá el fragmento completo

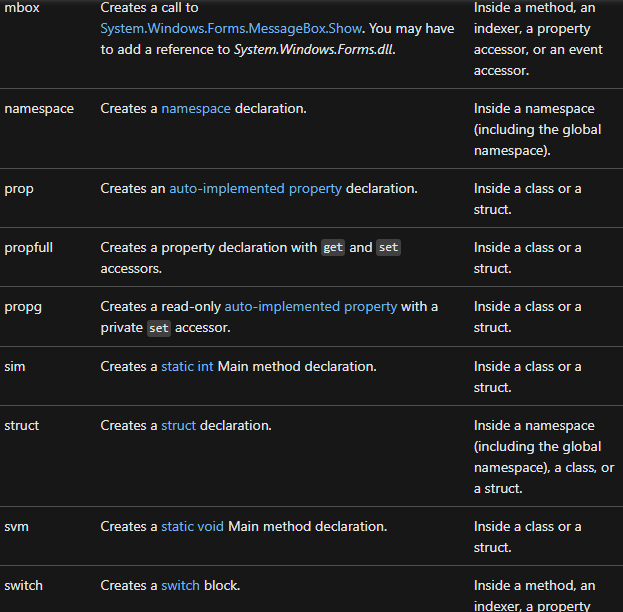
 y nos crea nuestra sentencia completa.

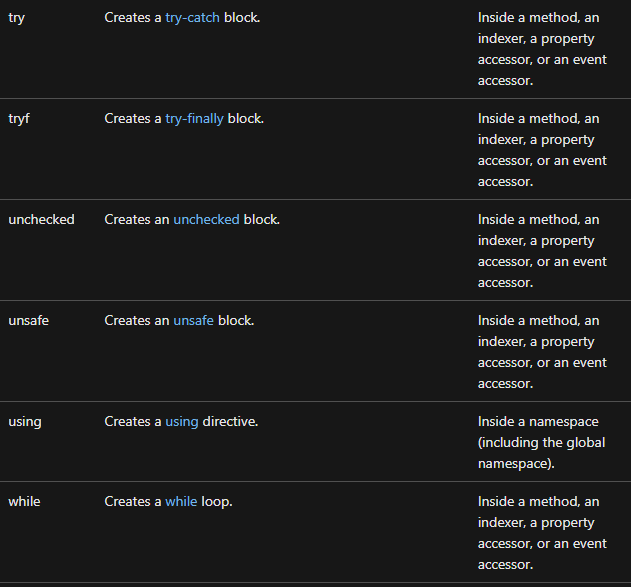


He acá una lista de como podremos usar estos fragmentos de código para agilizar nuestra programación, dejaré algunas más porque servirán en un futuro









Navegación en el código

Visual Studio ofrece numerosas formas de navegar por el código en el editor. En este tema se resumen las distintas maneras en las que se puede navegar por el código y se proporcionan vínculos a temas que incluyen más información

Puede usar los botones **Navegar hacia atrás** (**Ctrl**+**-**) y **Navegar hacia delante** (**Ctrl**+**Mayús**+**-**) de la barra de herramientas para mover el punto de inserción a ubicaciones anteriores o para volver a una ubicación más reciente desde una ubicación anterior. Estos botones retienen las últimas 20 ubicaciones del punto de inserción. Estos comandos también están disponibles en el menú **Vista**, en **Navegar hacia atrás** y **Navegar hacia delante**.

Usar barra de navegación

Puede usar la **barra de navegación** (los cuadros desplegables situados en la parte superior de la ventana de código) para navegar por el código de un código base. Puede elegir un tipo o un miembro para ir directamente a él. La barra de navegación aparece cuando se edita código en un código base de Visual Basic, C# o C++. En una clase parcial, los miembros definidos fuera del archivo de código actual pueden estar deshabilitados (se muestran en gris).



Comando para hacer navegación en el codigo

Ir a tiene los comandos siguientes, que están disponibles en el menú Editar en Ir a:

* Ir a la línea (Ctrl+G): moverse al número de línea especificado en el documento activo.
* Ir a todo (Ctrl+T o Ctrl+, ): Moverse a la línea, tipo, archivo, miembro o símbolo especificados.
* Ir al archivo (Ctrl+1, Ctrl+F): Moverse al archivo especificado en la solución.
* Ir al archivo reciente (Ctrl+1, Ctrl+R): Moverse al archivo especificado visitado recientemente en la solución.
* Ir al tipo (Ctrl+1, Ctrl+T): Moverse al tipo especificado en la solución.
* Ir al miembro (Ctrl+1, Ctrl+M): Moverse al miembro especificado en la solución.
* Ir al símbolo (Ctrl+1): Moverse al símbolo especificado en la solución.

Guía de productividad:

Es necesario dejar marcado como una de las páginas favoritas para utilizarlos en cualquier momento.

De los cuales deja en claro múltiples cosas para poder hacer un mejor uso de visual estudio, que atajos da, que formas de limpiar, como navegar fácilmente en el IDE, como trasladarse entre códigos. Si bien da varias listas de atajos para ser más productivos a la hora de estar programando, es demasiado útil cuando se tienen muchas líneas de código y múltiples formularios. Esto ayuda a que tengamos mejor movilidad y no cometamos muchos errores a la hora de escribir código.

Menciona también la forma de poder navegar en nuestro sistema, de cómo limpiar el código, qué significan los colores de subrayado, color rojo significa un error, verde es una advertencia, y los tres puntos en gris representa las sugerencias.

Nos da una serie de sugerencias de lo que podemos hacer, tales como depurar el código y que significa, que es procesar el código paso a paso, sirve para detectar de como está funcionando nuestro código, y como se comporta el mismo de acuerdo con las variables o de acuerdo a como lo estemos programando, y una lista de lo que nos ofrece es la siguiente:

* La capacidad de buscar una cadena dentro de las ventanas **Watch** (Inspeccionar), **Autos** (Autos) y **Locals** (Variables locales).
* *Ejecutar hasta hacer clic*, que permite mantener el puntero junto a una línea de código, hacer clic en el icono verde "reproducir" que aparece y ejecutar el programa hasta que llegue a esa línea.
* El **Asistente de excepciones**, que pone la información más importante en el nivel superior del cuadro de diálogo, por ejemplo, qué variable es null en NullReferenceException.
* La depuración [Retroceder](https://docs.microsoft.com/es-es/visualstudio/debugger/view-historical-application-state?view=vs-2022), que permite volver a puntos de interrupción anteriores y ver el estado de la aplicación tal y como estaba en un momento anterior.
* [Depuración de instantáneas](https://docs.microsoft.com/es-es/azure/application-insights/app-insights-snapshot-debugger), que permite investigar el estado de una aplicación web en directo en el momento en que se inició una excepción (debe estar en Azure).

Dejaré una tabla la cual muestra algunos accesos directos populares en visual estudio



Soluciones con múltiples proyectos

Proyecto

Debemos empezar a definir que serían los proyectos por que cuando cree una aplicación o un sitio web en Visual Studio, debe comenzar con un proyecto. En un sentido lógico, un proyecto contiene todos los archivos que se compilan en un archivo ejecutable, biblioteca o sitio web. Estos archivos pueden incluir código fuente, iconos, imágenes, archivos de datos, etc. Un proyecto también contiene la configuración del compilador y otros archivos de configuración que podrían ser necesarios en diversos servicios o componentes con los que el programa se comunica

Archivo de proyecto

Un archivo de proyecto Visual Studio usa MSBuild para compilar cada proyecto en una solución, y cada proyecto contiene un archivo de proyecto de MSBuild. La extensión de archivo refleja el tipo de proyecto, por ejemplo, un proyecto de C# (.csproj), un proyecto de Visual Basic (.vbproj) o un proyecto de base de datos (.dbproj). El archivo de proyecto es un documento XML que contiene toda la información y las instrucciones que MSBuild necesita para compilar el proyecto, incluidos el contenido, los requisitos de la plataforma, la información de versión, la configuración del servidor web o del servidor de bases de datos y las tareas que se deben llevar a cabo.

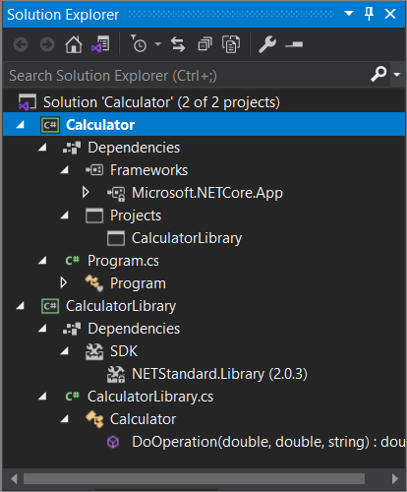
Soluciones

Un proyecto está incluido dentro de una solución. A pesar de su nombre, una solución no es una "respuesta", sino simplemente un contenedor con uno o más proyectos relacionados, junto con información de compilación, la configuración de ventanas de Visual Studio y archivos varios que no estén asociados a un proyecto determinado.

En Visual Studio se usan dos tipos de archivo ( *.sln* y *.suo*) para almacenar la configuración de las soluciones:



Carpeta de soluciones

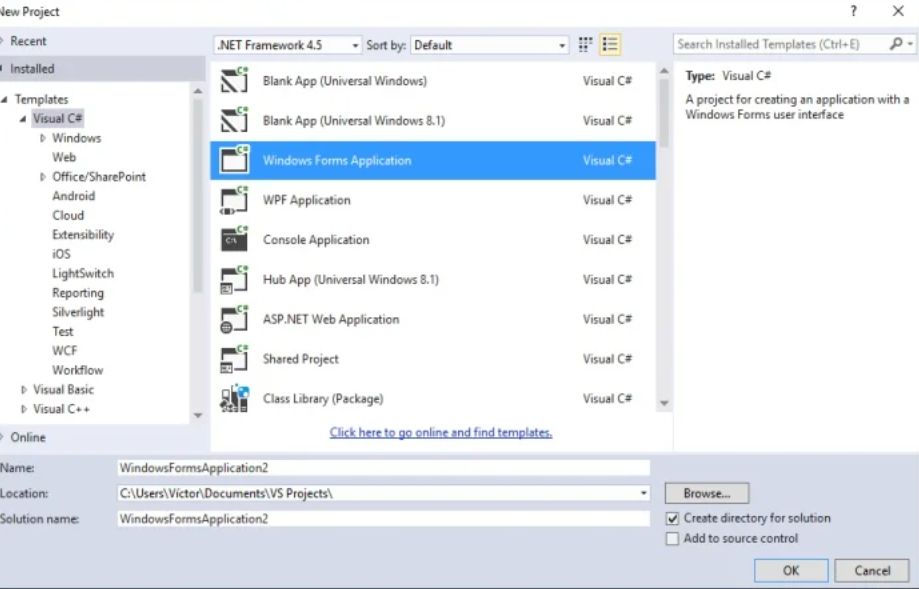
Una "carpeta de solución" es una carpeta virtual que solo se encuentra en el Explorador de soluciones, donde se puede usar para agrupar proyectos en una solución. Si desea buscar un archivo de una solución en un equipo, vaya a Herramientas>Opciones>Proyectos y soluciones>Ubicaciones. Para obtener más información, vea Cuadro de diálogo Opciones: Ubicaciones de proyectos y soluciones>.

Explorador de soluciones

Después de crear un proyecto nuevo, puede usar el Explorador de soluciones para ver y administrar el proyecto, la solución y sus elementos asociados. En la siguiente ilustración se muestra el Explorador de soluciones con una solución de C# que contiene dos proyectos:

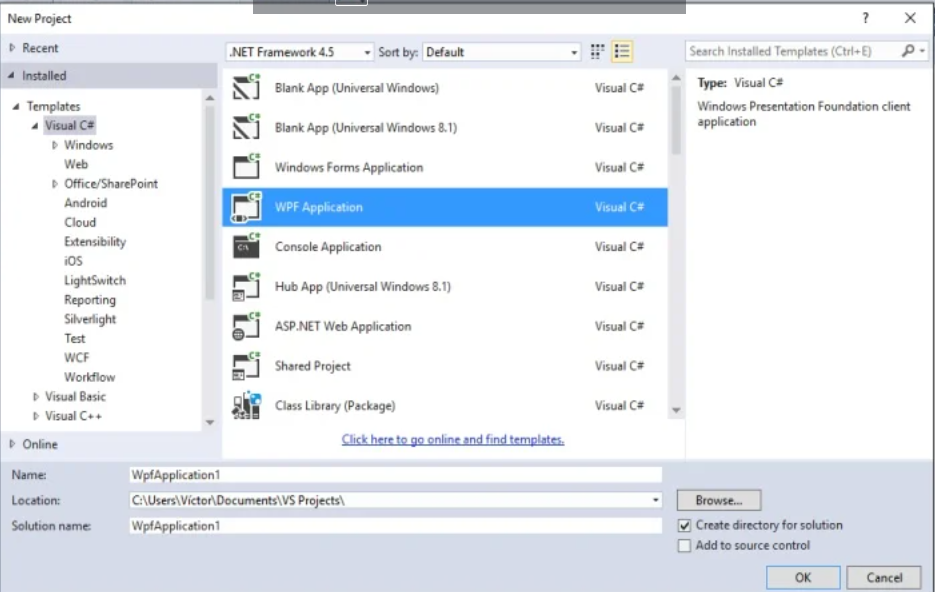
Elección de proyecto a ejecutar

Visual estudio tiene múltiples tipos de proyectos, de acuerdo a la necesidad que este presente, dejaré algunos de los proyectos más utilizados actualmente



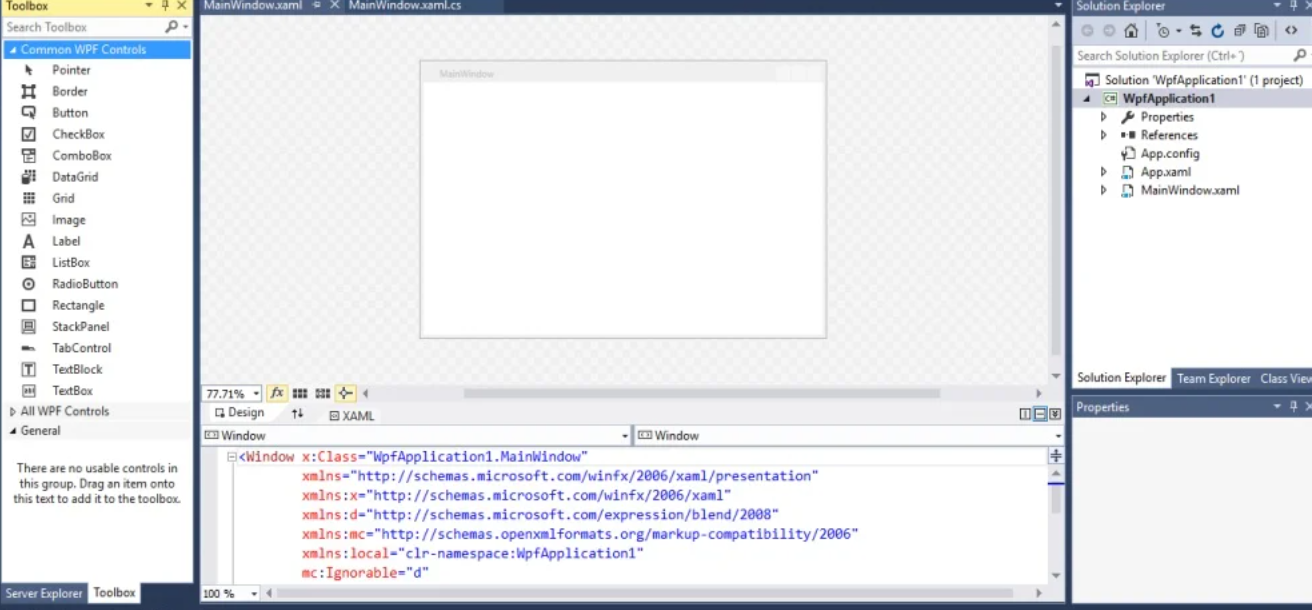
**Windows Forms**

Normalmente se escoge un proyecto de este tipo cuando se necesitan sistemas de uso diario en oficina, la utilización de teclado y mouse es indispensable. En una sola ventana (interfaz gráfica) pueden aparecer muchos controles de captura o de carga de información: TextBox, Labels, Grids, Menus, etc.

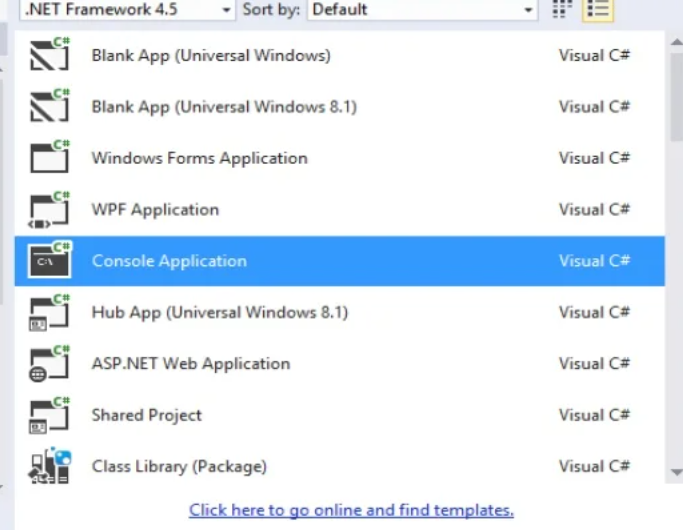


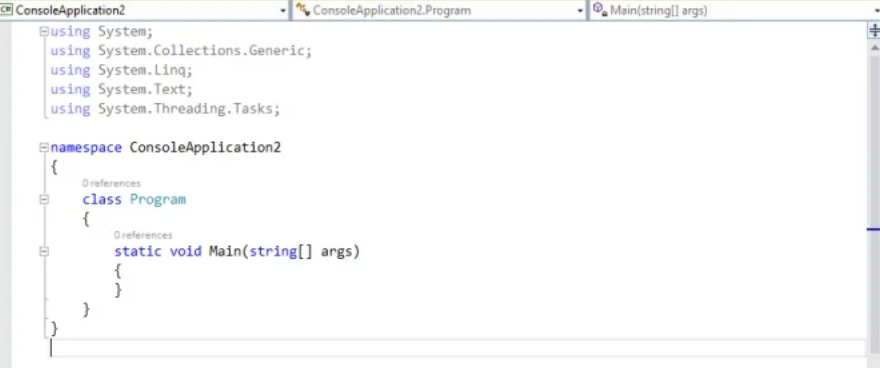
**Windows Presentation Foundation (WPF)**

Con el mismo objetivo que Windows Forms, la diferencia muy notable entre Windows Forms y WPF, es el uso de lenguaje XAML que separa el código de diseño del código funcional.

Windows Presentation Foundation (WPF) y XAML se combinan en un completo sistema de presentación para crear aplicaciones para el escritorio de Windows con experiencias de usuario visualmente impresionantes que incorporan interfaz de usuario, elementos multimedia y modelos empresariales complejos.

**Console Application**

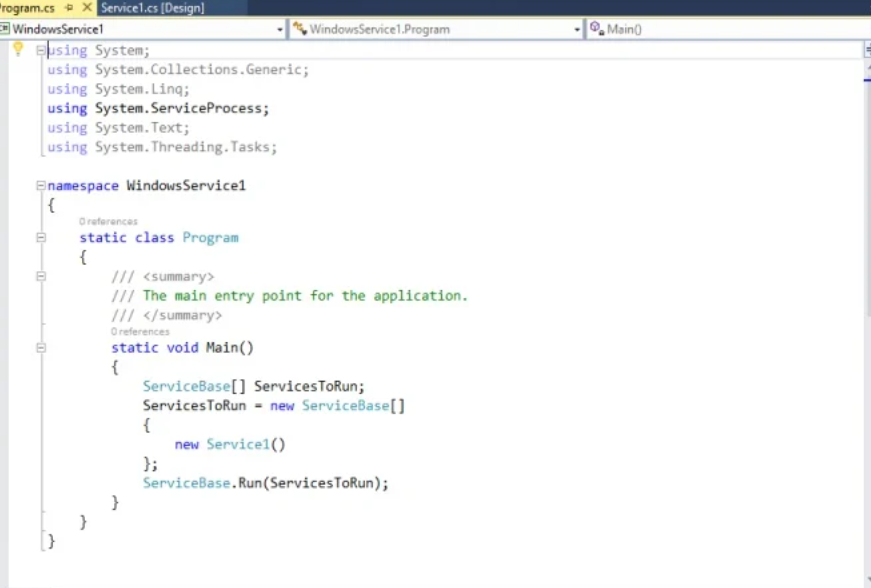
Este tipo de proyectos carecen de interfaz gráfica enriquecida, por ello son rápidas en su ejecución. Las aplicaciones de tipo consola se limitan a exponer un prompt como entrada de datos. Son recomendables cuando no se requiere demasiada entrada de información de los usuarios o los resultados de ejecución no necesitan un formato específico. Utilizan una interfaz como del tipo MSDOS



**Windows Services**

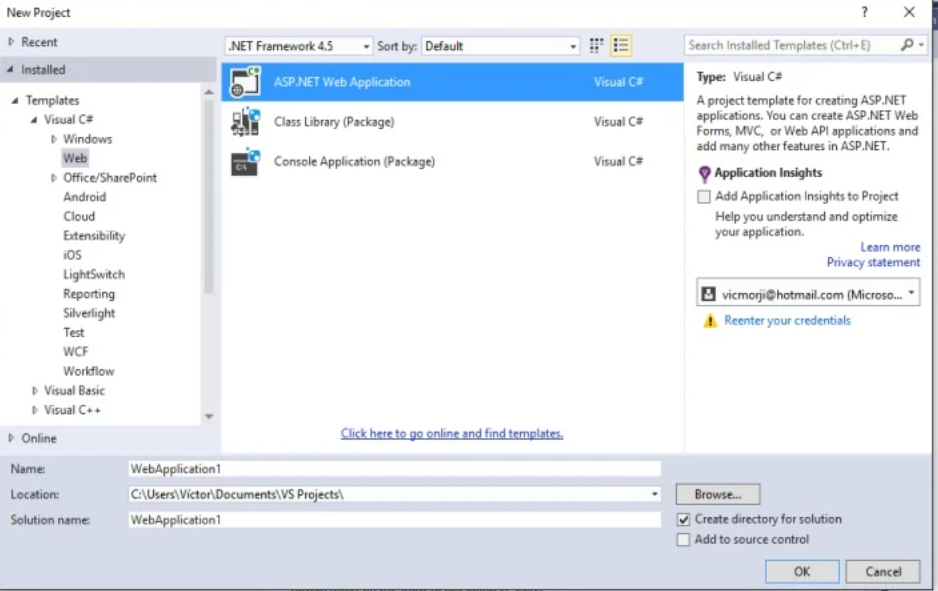
Un servicio de Windows es aquel que trabaja en paralelo, puede empezar a funcionar cuando arranca el sistema operativo, los servicios son aplicaciones en segundo plano que funcionan sin intervención del usuario.

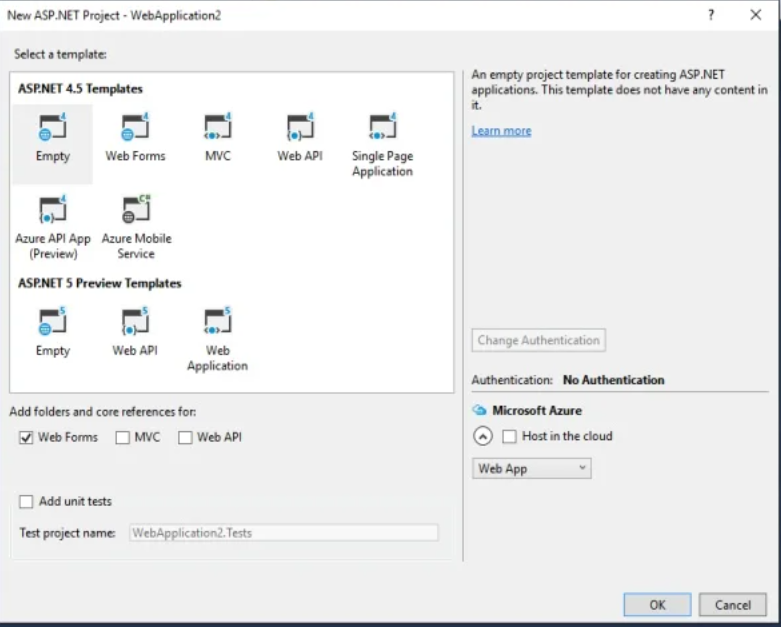




**ASP.Net**

Los proyectos ASP.Net trabajan con un comportamiento parecido a un proyecto tipo Windows Forms/WPF en el sentido de que existe la posibilidad de utilizar controles de usuario hacia nuestra página web (como si fuera una ventana de Windows). Aquí existen los llamados WebForms, que combinan HTML con lenguaje ASP.Net. Cada control que se coloca en nuestro WebForm tiene eventos programables y un montón de atributos a manipular, estos controles trabajan de lado del servidor, otorgando una respuesta por cada petición desde un browser.







**Silverlight**

Cuando las animaciones web estaban reinadas por Flash, llegó Microsoft Silverlight como otra opción. Los proyectos Silverlight permiten construir aplicaciones funcionales y enriquecidas al estilo de aplicaciones de escritorio WPF y XAML, la particularidad es que se utilizan desde el navegador web. De tal manera que, toda la experiencia de usuario y robustez de una aplicación de escritorio podía llevarse adentro de una página web. Al momento de escribir este artículo, Silverlight a perdido mucho terreno, puesto que los frameworks de JavaScript han evolucionado mucho y son preferibles para construir interfaces de usuario.

**Windows Phone**

Las verdaderas primeras aplicaciones móviles con tecnología Microsoft se llamaron Windows Phone (y digo esto porque antes existió un intento llamado Windows Mobile). Este tipo de proyectos utiliza XAML para el diseño la aplicación y C#/VB para programar la funcionalidad

**Xamarin**

Los proyectos de Xamarin permiten generar aplicaciones nativas para Windows, iOS y Android utilizando como lenguaje de programación C# y compartiendo el mismo código.

Archivos PDP

PDB es el acrónimo de Program Database. Es un formato propietario de Microsoft que almacena información de depuración de las aplicaciones ejecutables en Windows (.exe y .dll). Se crea a partir del código fuente para obtener una relación exhaustiva de los símbolos (variables, objetos, métodos...) del programa y su ubicación en el código y relacionarlos con el código compilado resultante. De esta manera, si necesitamos depurar la aplicación, las interfaces de depuración pueden "casar" las instrucciones de bajo nivel que se están ejecutando con las líneas de código concretas con las que se corresponden.

Con un PDB puedes depurar el código fuente a partir de volcados de memoria (y/o el ejecutable) con Visual Studio y herramientas como WinDBG

*Pero, si los archivos .pdb son para depuración, ¿por qué cuando genero una aplicación en modo release, para producción, también se me generan archivos .pdb? ¿Para qué me sirve eso?*

El motivo parece obvio, pero no lo es tanto. Y es que, si no tuviésemos este tipo de archivos, **no podríamos depurar la aplicación en producción**.

¿Cómo? ¿Depurar en producción? Parece un contrasentido. Al fin y al cabo, ya depuramos la aplicación mientras estamos desarrollándola ¿no?

En efecto. Durante las pruebas y el desarrollo ya depuramos la aplicación, por lo que la podemos desplegar a producción con ciertas garantías. Sin embargo, debemos tener en cuenta varias cosas:

* Hay muchas cosas que se nos van a escapar durante el desarrollo, que pueden producir errores una vez la aplicación está en funcionamiento.
* En ciertos tipos de aplicaciones, al estar en producción y tener muchos usuarios simultáneos, se dan situaciones muy difíciles de reproducir en desarrollo o pruebas, especialmente las que tienen que ver con código multisubproceso/asincronía.
* Si la aplicación se distribuye a muchos usuarios finales es muy probable que ciertos entornos personalizados (configuración, sistema operativo, nivel de parche, *drivers*...) den problemas que no hayamos podido tener en cuenta durante el desarrollo. En ese caso necesitaremos el .pdb para depurar el ejecutable concreto que presente los problemas.
* Cuando compilamos para *release* se le aplican multitud de optimizaciones al código final que no están al compilar en *debug* (no tanto por el *switch* /debug como el /optimize que se aplica en este caso). Generalmente esto no tiene impacto alguno sobre el funcionamiento de la aplicación más allá de mejorar su rendimiento, pero en ciertas ocasiones sí que pueden introducir cambios sutiles que afecten al resultado final e introduzcan nuevos bugs.
* Si hacemos *profiling* y optimización del ejecutable final, necesitaremos el .pdb para localizar las líneas que se pueden optimizar.

Diferencias entre Debug, Release, X86, X64

Un proyecto de Visual Studio tiene configuraciones independientes para las versiones de lanzamiento y depuración de un programa.Compile la versión de depuración (debug) para depurar y la versión de lanzamiento (release) para la versión de este tipo.

Se usan varias configuraciones para determinar si se compilará la versión de depuración o la comercial.Cuando se utiliza la configuración de compilación avanzada, el tipo de compilación viene determinado por los valores **Configuraciones de solución**, **Administrador de configuración** y **Propiedades del proyecto**, que incluyen la **Ruta de acceso de los resultados**.Cuando se utiliza la configuración de compilación simplificada, el tipo de compilación depende de si se hace clic en el comando **Iniciar depuración** o el comando **Compilar**.

Cuando compilas en debug, estás compilando para depurar, por lo cual la optimización es nula. Por el contrario, cuando compilas en Release, es para sacar una versión "comercial" del producto, por lo que se realizan muchas optimizaciones de código. Lo de x86 o x64 es dependiendo de si el producto se podrá ejecutar en pc de 32 o 64 bits

El "modo release" usualmente activa todas las optimizaciones importantes, para que el resultado sea veloz; mientras que el debug hace lo contrario. No es para que el resultado sea lento, sino para que al depurar el programa siga nuestras instrucciones

Bases de C#

Data table

Un objeto DataTable es prácticamente una tabla de datos, el cual puede estar relacionado con otro DataTable. Un ejemplo de como podemos usar un DataTable para hacer y crear tablas.

//Implementacion del datatable

DataTable tabla = new DataTable();

// Variables para las columnas y las filas

DataColumn column;

DataRow row;

// Se tiene que crear primero la columna asignandole Nombre y Tipo de datos

column = new DataColumn();

column.DataType = System.Type.GetType("System.Int32");

column.ColumnName = "Codigo";

tabla.Columns.Add(column);

// Se tienen que crear todas las columnas que queramos

column = new DataColumn();

column.DataType = Type.GetType("System.String");

column.ColumnName = "Descripcion";

tabla.Columns.Add(column);

column = new DataColumn();

column.DataType = Type.GetType("System.Decimal");

column.ColumnName = "ValorUnitario";

tabla.Columns.Add(column);

// Se crea una fila por cada registro que necesitemos agregar

for (int i = 1; i < 11; i++)

{

row = tabla.NewRow();

row["Codigo"] = i;

row["Descripcion"] = "Nombre del codigo " + i.ToString();

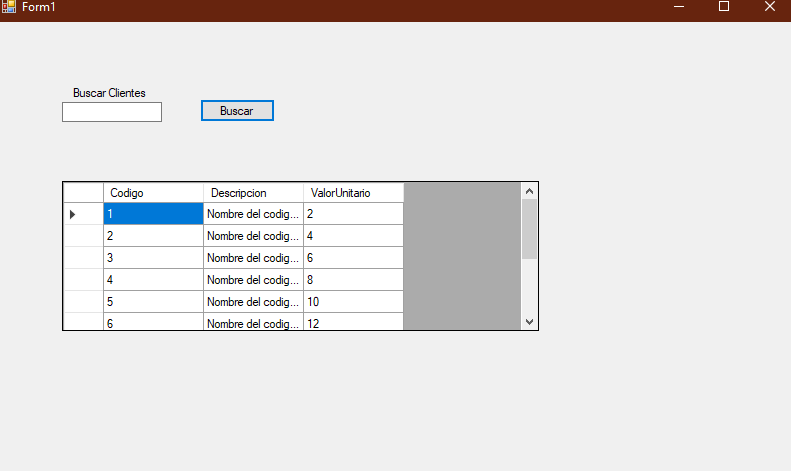
row["ValorUnitario"] = i \* 2;

tabla.Rows.Add(row);

}

dataGridView1.DataSource = tabla;

De lo cual hacemos un botón que ejecutará nuestro código, y veremos de la siguiente manera



DataSet

Un objeto DataSet es un contenedor, similar lógicamente a una base de datos. Es el que contendrá la totalidad de datos del conjunto sobre el que se desea trabajar. El tipo DataTable representa, como su propio nombre indica, una tabla.

Con una instancia de DataSet se puede usar cualquier número de objetos DataAdapter . Cada DataAdapter se puede usar para rellenar uno o varios objetos DataTable y resolver de nuevo las actualizaciones en el origen de datos correspondiente.

Para poder usar necesitamos usar una clase conexión que es la que hará contacto con nuestra base de datos la cual quedará de la siguiente manera:

public static class Conexion

{

public static int conf;

public static DataSet ds = new DataSet();

public static string s = "Server=DANIELLOPEZ\\PCDANIEL; DataBase= Northwind; Integrated Security=True";

public static SqlConnection conexion = new SqlConnection(s);

public static SqlCommand comando = new SqlCommand();

public static SqlDataAdapter sqlDA = new SqlDataAdapter();

static Conexion()

{

//CÓDIGO DEL CONSTRUCTOR

comando.Connection = conexion;

}

public static Tuple<int, DataSet> transacBD(string query, Boolean select)

{

DataSet dss = new DataSet();

if (select == true)

{

comando.CommandType = CommandType.Text;

comando.CommandText = query;

sqlDA.SelectCommand = comando;

conf = sqlDA.Fill(dss);

return Tuple.Create(conf, dss);

}

else

{

conexion.Open();

comando.CommandText = query;

conf = comando.ExecuteNonQuery();

conexion.Close();

return Tuple.Create(conf, dss);

}

}

}

Y vemos a continuación una variable de tipo dataSet, que es donde se están almacenando los datos que trae a partir de una base de datos sql. La cual retorna a la sentencia donde se esté llamando. Un ejemplo de como se usa la siguiente conexión sería de la siguiente manera:

//Ejemplo de como usar un data set

DataSet ejem = new DataSet();

var query = Conexion.transacBD("select CustomerID,CompanyName,ContactName,Address from customers where customerID like'%"+txtNombre.Text+"%' or 'CompanyName"+txtNombre.Text +"' ",true);

Conexion.conf = query.Item1;

Conexion.ds = query.Item2;

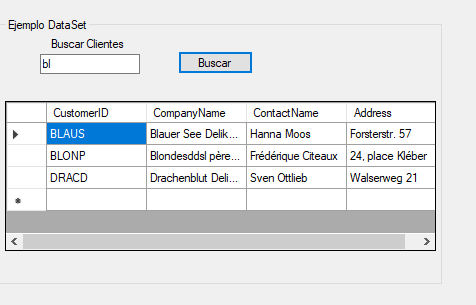
if (Conexion.ds.Tables[0].Rows.Count != 0)

{

dataGridView2.DataSource = Conexion.ds.Tables[0];

}

Lo cual nos hará una búsqueda de los clientes, según la base de datos que estuviéramos haciendo, la cual con un diseño ya establecido anteriormente quedará de la siguiente manera:



La búsqueda funciona con el CustomerID y CompanyName y se despliega en automático nuestra lista

Constructores

¿Cuál es la función de un constructor?

En **Programación** Orientada a Objetos (POO), un **constructor** es una subrutina cuya misión es inicializar un objeto de una clase. En el **constructor** se asignan los valores iniciales del nuevo objeto.

Cada vez que se crea una clase o estructura, se llama a su constructor. Una clase o estructura puede tener varios constructores que toman argumentos diferentes. Los constructores permiten al programador establecer valores predeterminados, limitar la creación de instancias y escribir código flexible y fácil de leer.

En C# podemos definir un método que se ejecute inicialmente y en forma automática. Este método se lo llama constructor.

El constructor tiene las siguientes características:

* Tiene el mismo nombre de la clase.
* Es el primer método que se ejecuta.
* Se ejecuta en forma automática.
* No puede retornar datos.
* Se ejecuta una única vez.
* Un constructor tiene por objetivo inicializar atributos.

Pondremos un ejemplo para aclarar lo siguiente:

public class Constructor1

{

int a;

int b;

//declaramos el constructor

public Constructor1(int x, int y)

{

a = x;

b = y;

}

public int Suma()

{

return a + b;

}

public int resta()

{

return a - b;

}

}

Este acaba de realizar una suma y una resta, nosotros podemos incluir lo que necesitemos. Se crea el constructor, declaramos el constructor y asignamos las variables, y para poderlos emplear debemos de hacer lo siguiente:

int n1,n2;

n1 = Convert.ToInt32( txtNum1.Text);

n2 = Convert.ToInt32(txtNum2.Text);

int rest, sum;

string cal;

Constructor1 ob= new Constructor1(n1,n2) ;

rest = ob.resta();

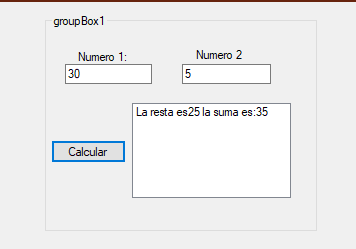
sum = ob.Suma();

cal=Convert.ToString("La resta es"+ rest + " "+ "la suma es:" + sum);

listBox1.Items.Add(cal);

declaramos las variables donde se asignaran los valores, y empezamos.

Tenemos que instanciar nuestro constructor a ob, y ya de ahí debemos retomar los datos para hacer suma o resta, la cual guardamos nuestro resultado a una variable, la cual luego asignamos un listbox que hará que se imprima lo que escribimos en los textbox, lo cual se muestra de la siguiente manera:



Anonymus type

Los tipos anónimos brindan una forma conveniente de encapsular un conjunto de propiedades de solo lectura en un solo objeto sin tener que definir explícitamente un tipo primero. El compilador genera el nombre del tipo y no está disponible en el nivel del código fuente. El compilador infiere el tipo de cada propiedad.

Los tipos anónimos se crean utilizando el newoperador junto con un inicializador de objeto. Para obtener más información acerca de los inicializadores de objetos, consulte Inicializadores de objetos y colecciones.

Un ejemplo práctico, junto con la demostración anterior del dataset, vemos como se usa una variable de tipo anónima, en este caso toma un valor que da soporte a un query, sin antes mencionar que es para llamar a una clase constructora que hace conexión a la base de datos.

var query = Conexion.transacBD("select CustomerID,CompanyName,ContactName,Address from customers where customerID like'%"+txtNombre.Text+ "%' or CompanyName like'%" + txtNombre.Text +"%' ",true);

Conexion.conf = query.Item1;

Conexion.ds = query.Item2;

if (Conexion.ds.Tables[0].Rows.Count != 0)

{

dataGridView2.DataSource = Conexion.ds.Tables[0];

}

Los tipos anónimos contienen una o más propiedades públicas de solo lectura. No son válidos otros tipos de miembros de clase, como métodos o eventos. La expresión que se utiliza para inicializar una propiedad no puede ser null una función anónima o un tipo de puntero.

Enumerable Type

En el lenguaje C#, enum(también denominado enumeración) es un tipo de valor definido por el usuario que se utiliza para representar una lista de constantes enteras con nombre. Se crea utilizando la enum palabra clave dentro de una clase, estructura o espacio de nombres. Mejora la legibilidad y el mantenimiento de un programa y reduce la complejidad.

La sintaxis básica para declarar en “enum” es:

public class ejemplenum

{

public enum meses

{

enero,

febrero,

abril,

junio,

agosto,

octubre,

noviembre,

diciembre

}

De esa manera indico que estoy asignando los valores al enum

}

// carga los datos en el load

private void EjemploConstructor\_Load(object sender, EventArgs e)

{

//Para unsar los enumerable tipe

string[] values = Enum.GetNames(typeof(ejemplenum.meses));

foreach (string s in values)

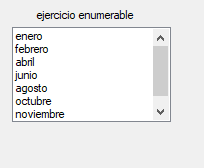
{

listBox2.Items.Add(s);

}

}

De esa manera puedo obtener los datos de lo que haya escrito, lo cual se muestra de la siguiente manera

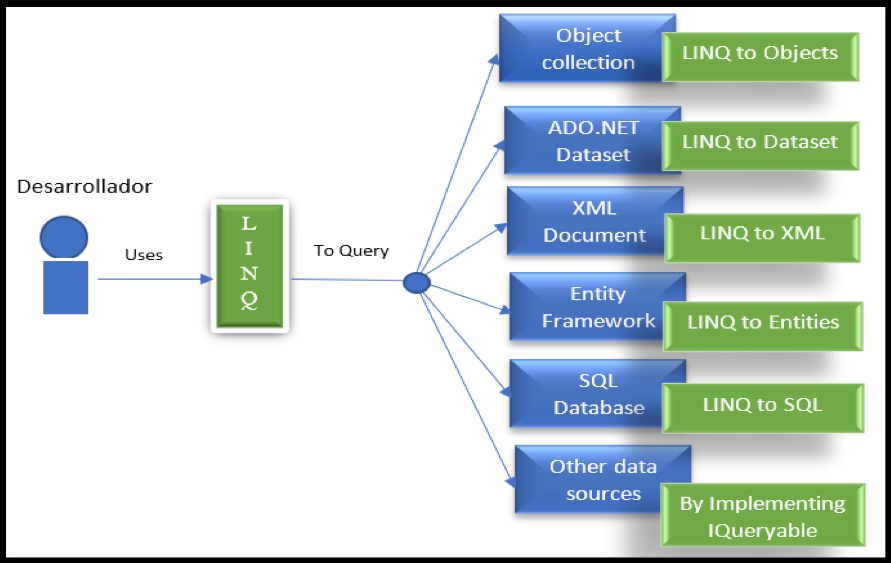


LINQ (lenguaje Integrated Query)

LINQ (Language Integrated Query) es una sintaxis de consulta uniforme en C # y VB.NET utilizada para guardar y recuperar datos de diferentes fuentes. Está integrado en C # o VB, lo que elimina la falta de correspondencia entre los lenguajes de programación y las bases de datos, y proporciona una interfaz de consulta única para diferentes tipos de fuentes de datos.

Por ejemplo, SQL es un lenguaje de consulta estructurado que se usa para guardar y recuperar datos de una base de datos. De la misma manera, LINQ es una sintaxis de consulta estructurada construida en C # y VB.NET utilizada para guardar y recuperar datos de diferentes tipos de fuentes de datos, como una colección de objetos, una base de datos SQL server, XML, MySQL o un servicio web entre otros.

LINQ siempre trabaja con objetos para que pueda usar los mismos patrones básicos de codificación para consultar y transformar los datos.



Realizamos un pequeño ejercicio para aplicar de lo que significa LINQ

class Carro

{

public string idCarro { get; set; }

public string Modelo { get; set; }

public string Year { get; set; }

public string Valor { get; set; }

public string idMarca { get; set; }

public string GetCArro()

{

string ds = ("idCarro: "+idCarro+"Modelo: "+ Modelo+"año :"+ Year+ "Valor: "+Valor+"La marca Corresponde a la empresa" +idMarca);

return ds;

}

}

class Marca

{

public int idMarca { get; set; }

public string nombre { get; set; }

}

Creamos nuestras clases, que será lo que estaremos ocupando, como si fueran tablas en bd

class controlCarro

{

/// <summary>

/// Constructor que ayudará a agregar datos a las tablas creadas anteriormente

/// </summary>

public controlCarro()

{

ListaCarro = new List<Carro>();

ListaMarca = new List<Marca>();

ListaMarca.Add(new Marca {idMarca=1, nombre= "Nissan" } );

ListaMarca.Add(new Marca { idMarca = 2, nombre = "Chevrolet" });

ListaMarca.Add(new Marca { idMarca = 3, nombre = "Ford" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo="N25-Versa", Year = "2020",Valor = "2500000", idMarca="1" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "C87-Camaro", Year = "2022", Valor = "12500000", idMarca = "2" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "F69-Fiesta", Year = "2022", Valor = "250060", idMarca = "3" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "N35-350z", Year = "2021", Valor = "980000", idMarca = "1" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "C87-Aveo", Year = "2020", Valor = "250600", idMarca = "2" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "F52-Mustang", Year = "2021", Valor = "293000", idMarca = "3" });

}

public List<Carro> ListaCarro;

public List<Marca> ListaMarca;

En este fragmento de código obtenemos los datos que estamos buscando y usamos un foreach para recorrer todos los datos que se encontraron.

public void getCarro( string año )

{

IEnumerable<Carro> Obmarca = from Carro in ListaCarro where Carro.Year == año select Carro;

foreach (Carro marca1 in Obmarca)

{

marca1.GetCArro();

}

}

}

Y al final asignamos una clase para llamar a ejecutar todo este código

private void btnMuestra\_Click(object sender, EventArgs e)

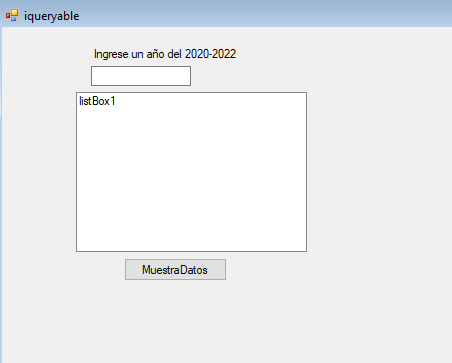
{

controlCarro crC = new controlCarro();

crC.getCarro(txtYear.Text);

}

Asigné dentro de un botón, que se ve de esta forma

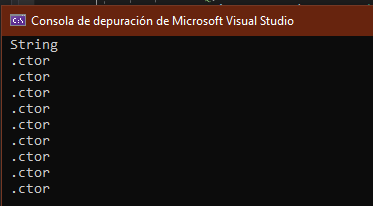
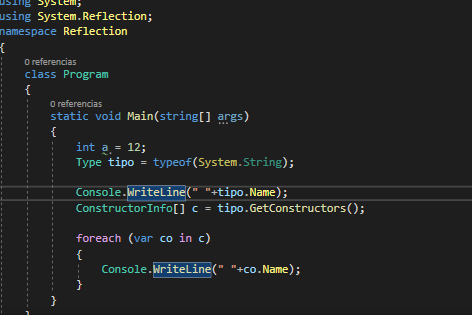


Lo que hace es hacer una búsqueda de los carros cuyo año nosotros pongamos

Reflexion

Reflection proporciona objetos (de tipo [Type](https://docs.microsoft.com/en-us/dotnet/api/system.type) ) que describen ensamblajes, módulos y tipos. Puede usar la reflexión para crear dinámicamente una instancia de un tipo, vincular el tipo a un objeto existente u obtener el tipo de un objeto existente e invocar sus métodos o acceder a sus campos y propiedades. Si está utilizando atributos en su código, la reflexión le permite acceder a ellos. Para obtener más información, consulte [Atributos](https://docs.microsoft.com/en-us/dotnet/standard/attributes/) .

Aquí hay un ejemplo simple de reflexión utilizando el método [GetType()](https://docs.microsoft.com/en-us/dotnet/api/system.object.gettype" \l "system-object-gettype) , heredado por todos los tipos de la Objectclase base, para obtener el tipo de una variable:



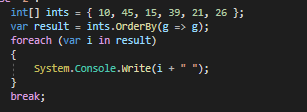
Por lo tanto esto nos muestra el contenido de cada variable, mostramos que información tiene el constructor, y por eso se ve de esa manera.

Métodos de Extensión

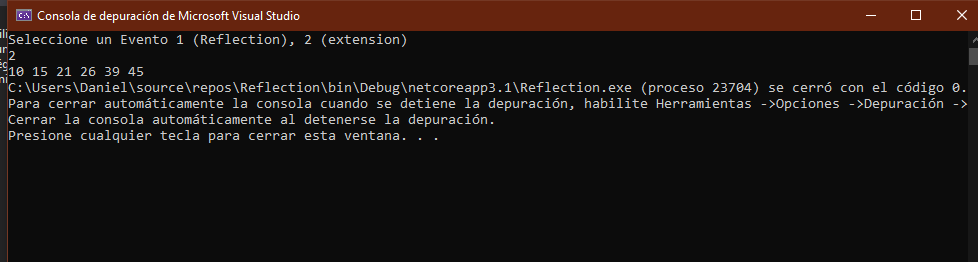
Los métodos de extensión permiten "agregar" métodos a los tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original. Los métodos de extensión son métodos estáticos, pero se les llama como si fueran métodos de instancia en el tipo extendido. En el caso del código de cliente escrito en C#, F# y Visual Basic, no existe ninguna diferencia aparente entre llamar a un método de extensión y llamar a los métodos definidos en un tipo.

Los métodos de extensión más comunes son los operadores de consulta LINQ estándar, que agregan funciones de consulta a los tipos [System.Collections.IEnumerable](https://docs.microsoft.com/es-es/dotnet/api/system.collections.ienumerable) y [System.Collections.Generic.IEnumerable<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.ienumerable-1) existentes. Para usar los operadores de consulta estándar, inclúyalos primero en el ámbito con una directiva using System.Linq. A partir de ese momento, cualquier tipo que implemente [IEnumerable<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.ienumerable-1) parecerá tener métodos de instancia como [GroupBy](https://docs.microsoft.com/es-es/dotnet/api/system.linq.enumerable.groupby), [OrderBy](https://docs.microsoft.com/es-es/dotnet/api/system.linq.enumerable.orderby), [Average](https://docs.microsoft.com/es-es/dotnet/api/system.linq.enumerable.average), etc. Puede ver estos métodos adicionales en la finalización de instrucciones de IntelliSense al escribir "punto" después de una instancia de un tipo [IEnumerable<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.ienumerable-1), como [List<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.list-1) o [Array](https://docs.microsoft.com/es-es/dotnet/api/system.array).

Ejemplo de Order by:



Que se vería de la siguiente manera: se ve claramente como se ordena los números internos

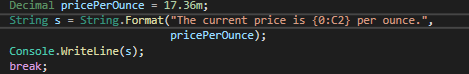


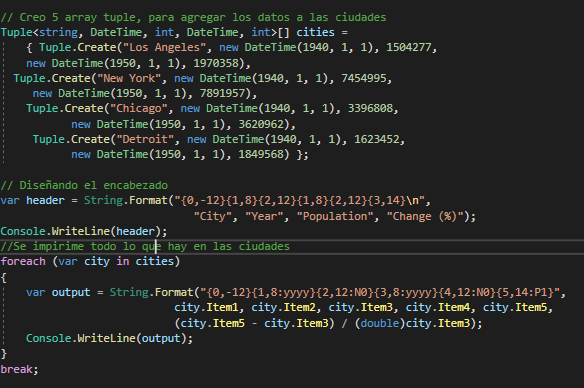
[buscar más ejemplos]

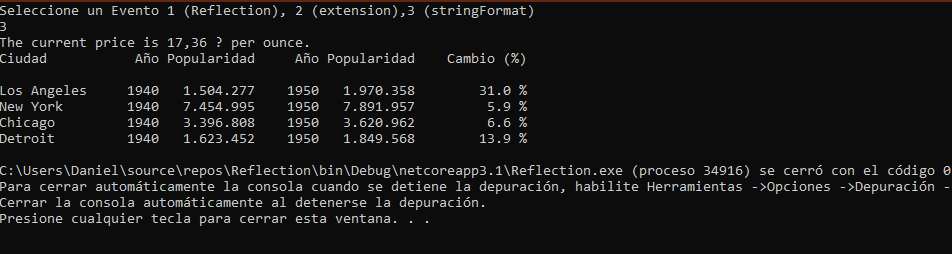
String Metodo

Convierte el valor de los objetos en cadenas según los formatos especificados y los inserta en otra cadena.

Utilice String.Format si necesita insertar el valor de un objeto, variable o expresión en otra cadena. Por ejemplo, puede insertar el valor de un valor decimal en una cadena para mostrárselo al usuario como una sola cadena:







Hay una diversidad de formas que podemos emplear el metodostring, todo sea de acuerdo a lo que se esté necesitando.

String Interpolation

La función de interpolación de cadenas se basa en la función de formato compuesto y proporciona una sintaxis más legible y conveniente para incluir resultados de expresiones formateadas en una cadena de resultados.

Para identificar un literal de cadena como una cadena interpolada, antepóngalo con el $símbolo. Puede incrustar cualquier expresión de C# válida que devuelva un valor en una cadena interpolada. En el siguiente ejemplo, tan pronto como se evalúa una expresión, su resultado se convierte en una cadena y se incluye en una cadena de resultado:

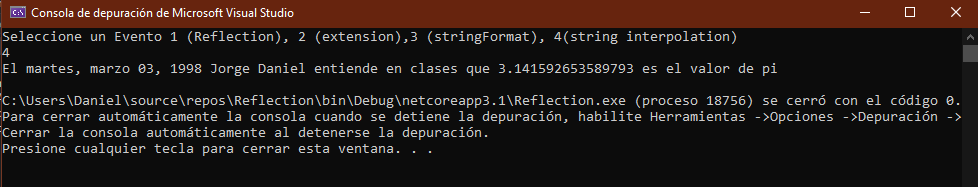
Especifique una cadena de formato compatible con el tipo de resultado de la expresión siguiendo la expresión de interpolación con dos puntos (":") y la cadena de formato:

{<interpolationExpression>:<formatString>}

El siguiente ejemplo muestra cómo especificar cadenas de formato estándar y personalizadas para expresiones que producen resultados numéricos o de fecha y hora:

var date = new DateTime(1998, 3, 3);

Console.WriteLine($"El {date:dddd, MMMM dd, yyyy} Jorge Daniel entiende en clases que {Math.PI} es el valor de pi");

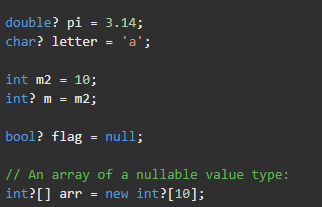


NUllable Types

Un *tipo de valor que acepta valores NULL representa todos los valores de su*[tipo de valor](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types)T? subyacente y un valor [NULL](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/null) adicional . Por ejemplo, puede asignar cualquiera de los tres valores siguientes a una variable: , o . Un tipo de valor subyacente no puede ser un tipo de valor anulable en sí mismo. Tbool?truefalsenullT

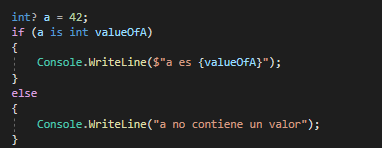
Cualquier tipo de valor que acepta valores NULL es una instancia de la estructura genérica System.Nullable<T> . Puede hacer referencia a un tipo de valor que acepta valores NULL con un tipo subyacente Ten cualquiera de las siguientes formas intercambiables: Nullable<T>o T?.

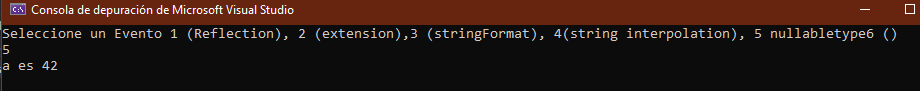
Dado que un tipo de valor se puede convertir implícitamente en el tipo de valor que acepta valores NULL correspondiente, puede asignar un valor a una variable de un tipo de valor que acepta valores NULL como lo haría con su tipo de valor subyacente. También puede asignar el nullvalor. Por ejemplo:



El valor predeterminado de un tipo de valor que admite valores NULL representa null, es decir, es una instancia cuya propiedad Nullable<T>.HasValue devuelve false.

Siempre puede usar las siguientes propiedades de solo lectura para examinar y obtener un valor de una variable de tipo de valor anulable:





Nullable<T>.HasValue indica si una instancia de un tipo de valor que acepta valores NULL tiene un valor de su tipo subyacente.

Nullable<T>.Value obtiene el valor de un tipo subyacente si HasValue es true. Si HasValue es false, la propiedad Value genera una InvalidOperationException .

NewtonJson

En la actualidad los datos JSON se utilizan sobre todo al recibir datos en formato JSON de un servicio web con AJAX y obtener datos de él. Para poder obtener datos en formato JSON o crear texto JSON a partir de un objeto personalizado, usaremos la serialización y la deserialización JSON en C#.

**¿Qué es JSON?**

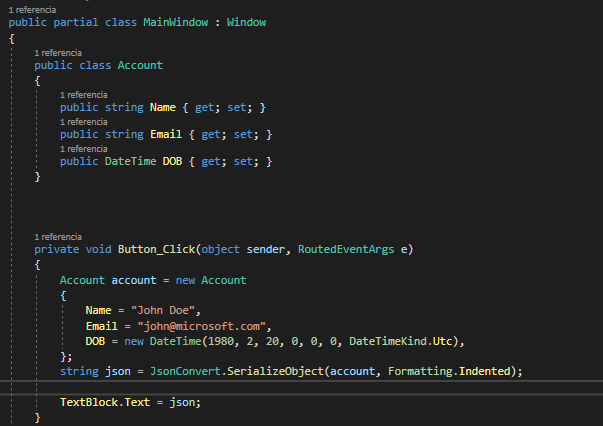
JSON (JavaScript Object Notation) es un formato liviano para el intercambio de datos. Es fácil para los humanos leer, escribir, analizar y generar un JSON y es un formato de texto que es completamente independiente del lenguaje.

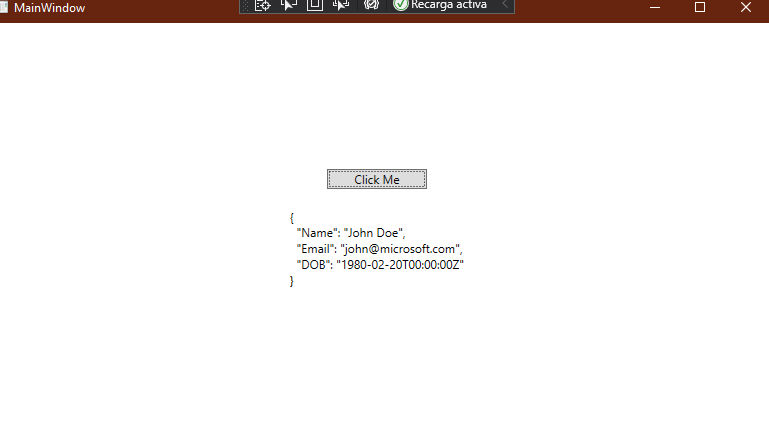
El formato JSON permite las estructuras de datos siguientes:

* **Colección de pares: nombre/valor, esta estructura de datos es compatible con diferentes lenguajes de programación.**
* **Lista ordenada de valores: puede ser una matriz, lista, vector o secuencia, etc.**

**¿Qué es Serialización?**

En ciencias de la computación, la serialización (o marshalling en inglés) consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros.





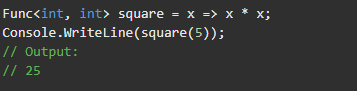
Con lo cual producimos un texto como ese, que es una estructura en JSON

Lambda

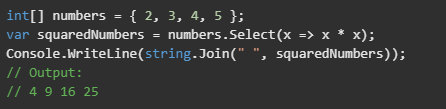
Una expresión lambda es una sintaxis para crear funciones anónimas en línea. Una expresión lambda es una función anónima que puede usar para crear delegados o tipos de árbol de expresiones. Al utilizar expresiones lambda, puede escribir funciones locales que pueden pasarse como argumentos o devolverse como el valor de las llamadas a funciones.

Una expresión lambda se crea utilizando el operador => . Ponga los parámetros en el lado izquierdo del operador. En el lado derecho, ponga una expresión que pueda usar esos parámetros; esta expresión se resolverá como el valor de retorno de la función. Más raramente, si es necesario, se puede usar un {code block} completo en el lado derecho. Si el tipo de retorno no es nulo, el bloque contendrá una declaración de retorno.

Cualquier expresión lambda se puede convertir en un tipo de delegado . El tipo de delegado al que se puede convertir una expresión lambda se define por los tipos de sus parámetros y el valor de retorno. Si una expresión lambda no devuelve un valor, se puede convertir a uno de los Actiontipos de delegado; de lo contrario, se puede convertir a uno de los Functipos de delegado. Por ejemplo, una expresión lambda que tiene dos parámetros y no devuelve ningún valor se puede convertir en un delegado Action<T1,T2> . Una expresión lambda que tiene un parámetro y devuelve un valor se puede convertir en un delegado Func<T,TResult> . En el siguiente ejemplo, la expresión lambda x => x \* x, que especifica un parámetro con nombre xy devuelve el valor del cuadrado, se asigna a una variable de tipo delegado:



Puede usar expresiones lambda en cualquier código que requiera instancias de tipos delegados o árboles de expresión, por ejemplo, como un argumento para el método Task.Run(Action) para pasar el código que debe ejecutarse en segundo plano. También puede usar expresiones lambda cuando escribe LINQ en C# , como muestra el siguiente ejemplo:



*El cuerpo de una expresión lambda puede consistir en una llamada de método. Sin embargo, si está creando árboles de expresión que se evalúan fuera del contexto de .NET Common Language Runtime (CLR), como en SQL Server, no debe usar llamadas a métodos en expresiones lambda. Los métodos no tendrán significado fuera del contexto de .NET Common Language Runtime (CLR).*

**Parámetros de entrada de una expresión lambda**

Encierra los parámetros de entrada de una expresión lambda entre paréntesis. Especifique cero parámetros de entrada con paréntesis vacíos:



Si una expresión lambda tiene solo un parámetro de entrada, los paréntesis son opcionales: 

Dos o más parámetros de entrada están separados por comas:



A veces, el compilador no puede inferir los tipos de parámetros de entrada. Puede especificar los tipos explícitamente como se muestra en el siguiente ejemplo:



**Expresiones lambda y tuplas**

A partir de C# 7.0, el lenguaje C# proporciona soporte integrado para tuplas . Puede proporcionar una tupla como argumento para una expresión lambda y su expresión lambda también puede devolver una tupla. En algunos casos, el compilador de C# utiliza la inferencia de tipos para determinar los tipos de componentes de tupla.

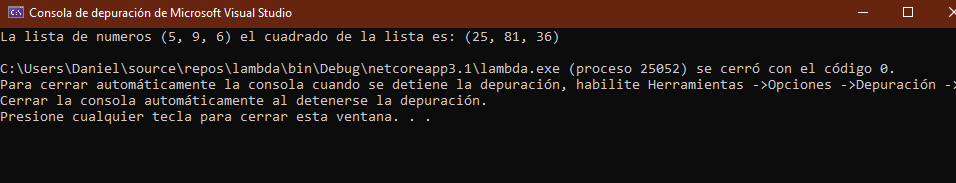
Una tupla se define encerrando una lista delimitada por comas de sus componentes entre paréntesis. El siguiente ejemplo utiliza una tupla con tres componentes para pasar una secuencia de números a una expresión lambda, que duplica cada valor y devuelve una tupla con tres componentes que contiene el resultado de las multiplicaciones de las mismas para obtener el cuadrado de los valores.

Func<(int, int, int), (int, int, int)> doubleThem = ns => (ns.Item1 \* ns.Item1, ns.Item2 \* ns.Item2, ns.Item3\* ns.Item3);

var numbers = (5,9, 6);

var doubledNumbers = doubleThem(numbers);

Console.WriteLine($"La lista de numeros {numbers} el cuadrado de la lista es: {doubledNumbers}")



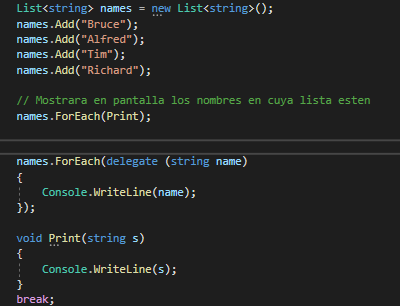
Action, function, delegates

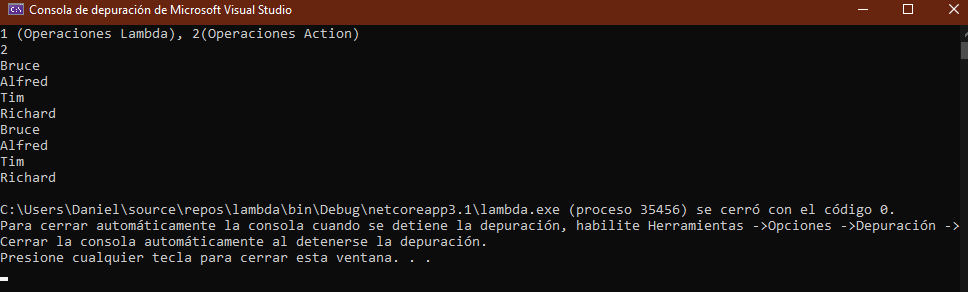
**Action**

Encapsula un método que tiene un único parámetro y no devuelve ningún valor.

Action<T>, es un delegado genérico que encapsula un método que posee un solo parámetro y no retorna un valor. Es decir, un Action<T> nos permite crear un delegado sin tener que declarar el tipo. Como un método void

El siguiente ejemplo demuestra el uso del delegado [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0) para imprimir el contenido de un objeto [List<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0) . En este ejemplo, el Printmétodo se usa para mostrar el contenido de la lista en la consola. Además, el ejemplo de C# también demuestra el uso de métodos anónimos para mostrar el contenido en la consola. Tenga en cuenta que el ejemplo no declara explícitamente una variable [Action<T> .](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0)En su lugar, pasa una referencia a un método que toma un solo parámetro y que no devuelve un valor al método [List<T>.ForEach](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.foreach?view=net-6.0) , cuyo único parámetro es un delegado [Action<T> .](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0)De manera similar, en el ejemplo de C#, un [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0)El delegado no se crea explícitamente porque la firma del método anónimo coincide con la firma del delegado [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0) que espera el método [List<T>.ForEach](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.foreach?view=net-6.0) .





**Por lo tanto un delegado:**

Un delegado es un tipo que representa referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto. Cuando se crea una instancia de un delegado, puede asociar su instancia a cualquier método mediante una signatura compatible y un tipo de valor devuelto. Puede invocar (o llamar) al método a través de la instancia del delegado.

Los delegados se utilizan para pasar métodos como argumentos a otros métodos. Los controladores de eventos no son más que métodos que se invocan a través de delegados. Cree un método personalizado y una clase, como un control de Windows, podrá llamar al método cuando se produzca un determinado evento. En el siguiente ejemplo se muestra una declaración de delegado:

Para terminar de aclarar Action<T>, es un delegado genérico que encapsula un método que posee un solo parámetro y no retorna un valor. Es decir, un Action<T> nos permite crear un delegado sin tener que declarar el tipo.

Cualquier método de cualquier clase o struct accesible que coincida con el tipo de delegado se puede asignar al delegado. El método puede ser estático o de instancia. Esta flexibilidad significa que puede cambiar las llamadas de método mediante programación, o bien agregar código nuevo a las clases existentes.



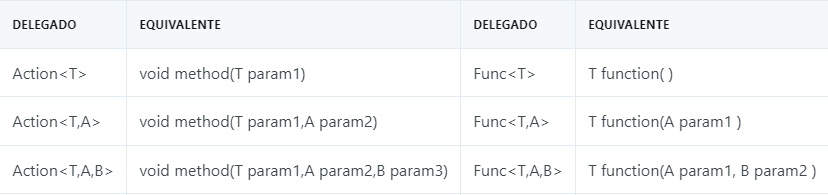
Los delegados tienen las propiedades siguientes:

* Los delegados son similares a los punteros de función de C++, pero los primeros están completamente orientados a objetos y, a diferencia de los punteros de C++ de funciones de miembro, los delegados encapsulan una instancia de objeto y un método.
* Los delegados permiten pasar los métodos como parámetros.
* Los delegados pueden usarse para definir métodos de devolución de llamada.
* Los delegados pueden encadenarse entre sí; por ejemplo, se puede llamar a varios métodos en un solo evento.
* No es necesario que los métodos coincidan exactamente con el tipo de delegado.
* Las expresiones lambda son una manera más concisa de escribir bloques de código alineado. En determinados contextos, las expresiones lambda se compilan en tipos de delegado.

**Function**

Una Func es muy parecida a una action<T>, de hecho ambos son delegados genéricos, la diferencia es que en el caso de la Func<T>, esta devuelve un valor.

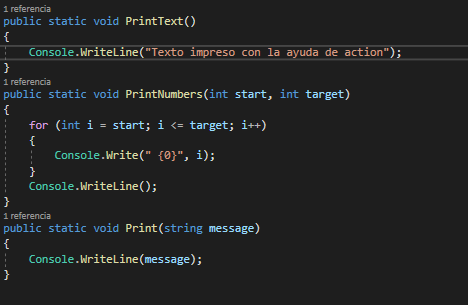
**¿Cuantos parámetros como máximo puede tener un Action<> o Func<>?**



Ejemplo de como se usan los Function y Action



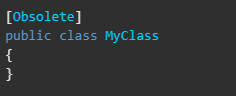
Se tuvo que crear una librería de clases para observar cómo funciona el siguiente programa



Atributes

Los atributos proporcionan una forma de asociar información con el código de forma declarativa. También pueden proporcionar un elemento reutilizable que se puede aplicar a una variedad de objetivos.

Considere el [Obsolete]atributo. Se puede aplicar a clases, estructuras, métodos, constructores y más. Declara que *el* elemento está obsoleto. Luego depende del compilador de C# buscar este atributo y realizar alguna acción en respuesta.



Los atributos proporcionan un método eficaz para asociar metadatos o información declarativa con código (ensamblajes, tipos, métodos, propiedades, etc.). Después de asociar un atributo con una entidad de programa, el atributo se puede consultar en tiempo de ejecución mediante una técnica llamada *reflexión*.

Los atributos tienen las siguientes propiedades:

* Los atributos agregan metadatos a su programa. *Los metadatos* son información sobre los tipos definidos en un programa. Todos los ensamblados de .NET contienen un conjunto específico de metadatos que describen los tipos y los miembros de tipo definidos en el ensamblado. Puede agregar atributos personalizados para especificar cualquier información adicional que se requiera. Para obtener más información, consulte [Creación de atributos personalizados (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/attributes/creating-custom-attributes) .
* Puede aplicar uno o más atributos a ensamblajes completos, módulos o elementos de programa más pequeños, como clases y propiedades.
* Los atributos pueden aceptar argumentos de la misma forma que los métodos y las propiedades.
* Su programa puede examinar sus propios metadatos o los metadatos de otros programas utilizando la reflexión.

Crear un atributo es tan simple como heredar de la Attributeclase base.

Los atributos se pueden utilizar en varios "objetivos". Los ejemplos anteriores los muestran en las clases, pero también se pueden usar en:

* Asamblea
* Clase
* Constructor
* Delegar
* enumeración
* Evento
* Campo
* Parámetro genérico
* Interfaz
* Método
* Módulo
* Parámetro
* Propiedad
* valor de retorno
* estructura

Los atributos son utilizados por muchas herramientas y marcos. NUnit usa atributos como [Test]y [TestFixture]que usa el corredor de pruebas de NUnit. ASP.NET MVC usa atributos como [Authorize] y proporciona un marco de filtro de acción para realizar preocupaciones transversales en las acciones de MVC. PostSharp usa la sintaxis de atributos para permitir la programación orientada a aspectos en C#.

Aquí hay algunos atributos notables integrados en las bibliotecas de clases base de .NET Core:

* [Obsolete]. Este se usó en los ejemplos anteriores y vive en el espacio de Systemnombres. Es útil proporcionar documentación declarativa sobre una base de código cambiante. Se puede proporcionar un mensaje en forma de cadena y se puede usar otro parámetro booleano para pasar de una advertencia del compilador a un error del compilador.
* [Conditional]. Este atributo está en el espacio de System.Diagnosticsnombres. Este atributo se puede aplicar a métodos (o clases de atributos). Debe pasar una cadena al constructor. Si esa cadena no coincide con una #definedirectiva, el compilador de C# eliminará todas las llamadas a ese método (pero no al método en sí). Por lo general, esto se usa con fines de depuración (diagnóstico).
* [CallerMemberName]. Este atributo se puede usar en parámetros y reside en el espacio de System.Runtime.CompilerServicesnombres. Este es un atributo que se usa para inyectar el nombre del método que está llamando a otro método. Por lo general, esto se usa como una forma de eliminar las "cadenas mágicas" al implementar INotifyPropertyChanged en varios marcos de interfaz de usuario.

Los atributos aportan poder declarativo a C#, pero son una forma de código de metadatos y no actúan por sí mismos.

Servicios de Windows

¿Qué son los servicios de Windows y para qué sirven?

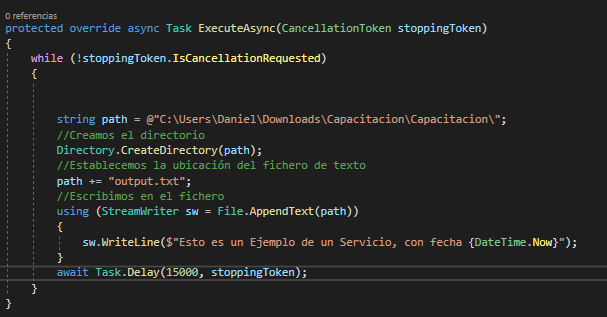
Los servicios de Microsoft Windows, anteriormente conocidos como servicios NT, le permiten crear aplicaciones de larga ejecución que se ejecutan en sesiones propias de Windows.

Los servicios de Microsoft Windows, anteriormente conocidos como servicios NT, le permiten crear aplicaciones de larga ejecución que se ejecutan en sesiones propias de Windows. Estos servicios se pueden iniciar automáticamente al arrancar el equipo, se pueden pausar y reiniciar, y no muestran ninguna interfaz de usuario. Estas características hacen que los servicios sean idóneos para su uso en un servidor o siempre que necesite una funcionalidad de larga duración que no interfiera con otros usuarios que estén trabajando en el mismo equipo. También puede ejecutar servicios en el contexto de seguridad de una cuenta de usuario específica que sea diferente del usuario conectado o de la cuenta de equipo predeterminada.

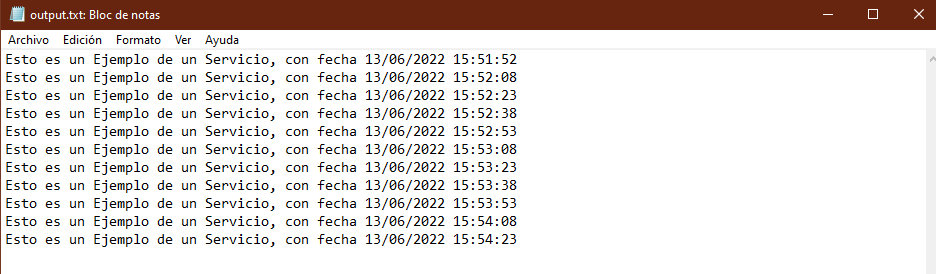
Puede crear servicios fácilmente mediante la creación de una aplicación que se instale como un servicio. Por ejemplo, suponga que desea supervisar datos del contador de rendimiento y reaccionar a valores de umbral. Puede escribir una aplicación de servicio de Windows que escuche los datos del contador de rendimiento, implemente la aplicación y comience a recopilar y analizar datos.

Cree el servicio como un proyecto de Microsoft Visual Studio y defina el código dentro de él que controle qué comandos se pueden enviar al servicio y qué acciones se deben realizar cuando se reciben esos comandos. Los comandos que se pueden enviar a un servicio incluyen iniciar, pausar, reanudar y detener el servicio; también puede ejecutar comandos personalizados.

Después de crear y compilar la aplicación, puede instalarla mediante la ejecución de la utilidad de línea de comandos InstallUtil.exe y el paso de la ruta de acceso al archivo ejecutable del servicio. A continuación, puede usar el Administrador de control de servicios para iniciar, detener, pausar, reanudar y configurar el servicio. También puede realizar muchas de estas mismas tareas en el nodo Servicios Explorador de servidores o mediante la clase.



Lo que hace este fragmento de código es que cada 15 segundos se actualiza el mensaje en un archivo txt, que guardé en la carpeta de capacitación, el cual dejó un mensaje de la siguiente manera



Esto solo es un ejemplo, estos archivos se ejecutan en segundo plano, y sin interrupciones

restFull Api

En el nivel más básico, una [API](https://www.ibm.com/cloud/learn/api) es un mecanismo que permite a una aplicación o servicio acceder a un recurso dentro de otra aplicación o servicio. La aplicación o servicio que realiza el acceso se denomina cliente y la aplicación o servicio que contiene el recurso se denomina servidor.

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. El informático Roy Fielding es el creador de la transferencia de estado representacional (REST).

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones.Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.

En otras palabras, las API le permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla.

Imagínelas como si fueran los mediadores entre los usuarios o clientes y los recursos o servicios web que quieren obtener. Con ellas, las empresas pueden compartir recursos e información mientras conservan la seguridad, el control y la autenticación, lo cual les permite determinar el contenido al que puede acceder cada usuario.

Otra ventaja de las API es que usted no necesita saber cómo se recibe el recurso ni de dónde proviene.

REST no es un protocolo ni un estándar, sino más bien un conjunto de límites de arquitectura. Los desarrolladores de las API pueden implementarlo de distintas maneras.

Cuando el cliente envía una solicitud a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información se entrega por medio de HTTP en uno de estos formatos: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato. JSON es el lenguaje de programación más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.

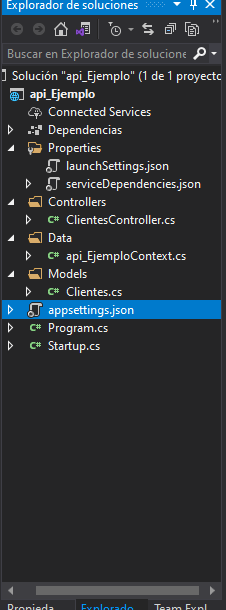
También es necesario tener en cuenta otros aspectos. Los encabezados y los parámetros también son importantes en los métodos HTTP de una solicitud HTTP de la API de RESTful, ya que contienen información de identificación importante con respecto a los metadatos, la autorización, el identificador uniforme de recursos (URI), el almacenamiento en caché, las cookies y otros elementos de la solicitud. Hay encabezados de solicitud y de respuesta, pero cada uno tiene sus propios códigos de estado e información de conexión HTTP.

Para que una API se considere de RESTful, debe cumplir los siguientes criterios:

* Arquitectura cliente-servidor compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP.
* Comunicación entre el cliente y el servidor [sin estado](https://www.redhat.com/es/topics/cloud-native-apps/stateful-vs-stateless), lo cual implica que no se almacena la información del cliente entre las solicitudes de GET y que cada una de ellas es independiente y está desconectada del resto.
* Datos que pueden almacenarse en caché y optimizan las interacciones entre el cliente y el servidor.
* Una interfaz uniforme entre los elementos, para que la información se transfiera de forma estandarizada. Para ello deben cumplirse las siguientes condiciones:
  + Los recursos solicitados deben ser identificables e independientes de las representaciones enviadas al cliente.
  + El cliente debe poder manipular los recursos a través de la representación que recibe, ya que esta contiene suficiente información para permitirlo.
  + Los mensajes autodescriptivos que se envíen al cliente deben contener la información necesaria para describir cómo debe procesarla.
  + Debe contener hipertexto o hipermedios, lo cual significa que cuando el cliente acceda a algún recurso, debe poder utilizar hipervínculos para buscar las demás acciones que se encuentren disponibles en ese momento.
* Un sistema en capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores (los encargados de la seguridad, del equilibrio de carga, etc.) que participan en la recuperación de la información solicitada.
* Código disponible según se solicite (opcional), es decir, la capacidad para enviar códigos ejecutables del servidor al cliente cuando se requiera, lo cual amplía las funciones del cliente.

Si bien la API de REST debe cumplir todos estos parámetros, resulta más fácil de usar que un protocolo definido previamente, como SOAP (protocolo simple de acceso a objetos), el cual tiene requisitos específicos, como la mensajería XML y la seguridad y el cumplimiento integrados de las operaciones, que lo hacen más lento y pesado.

Por el contrario, REST es un conjunto de pautas que pueden implementarse según sea necesario. Por esta razón, las API de REST son más rápidas y ligeras, cuentan con mayor capacidad de ajuste y, por ende, resultan ideales para el [Internet de las cosas (IoT)](https://www.redhat.com/es/topics/internet-of-things-570051) y el [desarrollo de aplicaciones para dispositivos móviles](https://www.redhat.com/es/topics/mobile).

Para hacer un pequeño ejemplo debemos de hacer un nuevo proyecto en Visual Estudio, de tipo Web/asp core. Net, y seleccionar del tipo Api.  
A lo cual debemos de agregar los siguientes controladores, clases, lo cual nos quedará de la siguiente manera: en donde tenemos nuestro modelo donde ponemos una tabla donde se recibe datos de la base de datos que creamos de ejemplo, se hace un controlador para es el crud que tiene el API, lo cual se ve de la siguiente manera:

[Route("api/[controller]")]

[ApiController]

public class ClientesController : ControllerBase

{

private readonly api\_EjemploContext \_context;

public ClientesController(api\_EjemploContext context)

{

\_context = context;

}

// GET: api/Clientes

[HttpGet]

public async Task<ActionResult<IEnumerable<Clientes>>> GetClientes()

{

return await \_context.Clientes.ToListAsync();

}

// GET: api/Clientes/5

[HttpGet("{id}")]

public async Task<ActionResult<Clientes>> GetClientes(int id)

{

var clientes = await \_context.Clientes.FindAsync(id);

if (clientes == null)

{

return NotFound();

}

return clientes;

}

// PUT: api/Clientes/5

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see https://go.microsoft.com/fwlink/?linkid=2123754.

[HttpPut("{id}")]

public async Task<IActionResult> PutClientes(int id, Clientes clientes)

{

if (id != clientes.idCliente)

{

return BadRequest();

}

\_context.Entry(clientes).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ClientesExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/Clientes

// To protect from overposting attacks, enable the specific properties you want to bind to, for

// more details, see https://go.microsoft.com/fwlink/?linkid=2123754.

[HttpPost]

public async Task<ActionResult<Clientes>> PostClientes(Clientes clientes)

{

\_context.Clientes.Add(clientes);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetClientes", new { id = clientes.idCliente }, clientes);

}

// DELETE: api/Clientes/5

[HttpDelete("{id}")]

public async Task<ActionResult<Clientes>> DeleteClientes(int id)

{

var clientes = await \_context.Clientes.FindAsync(id);

if (clientes == null)

{

return NotFound();

}

\_context.Clientes.Remove(clientes);

await \_context.SaveChangesAsync();

return clientes;

}

private bool ClientesExists(int id)

{

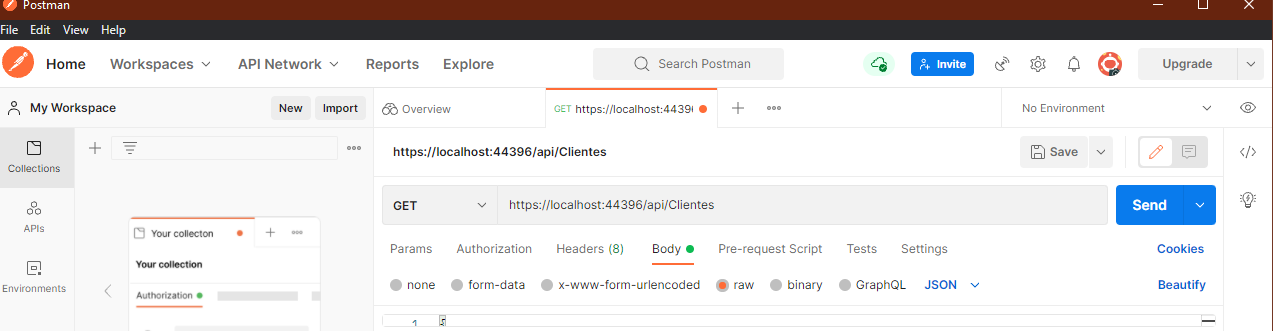
return \_context.Clientes.Any(e => e.idCliente == id);

}

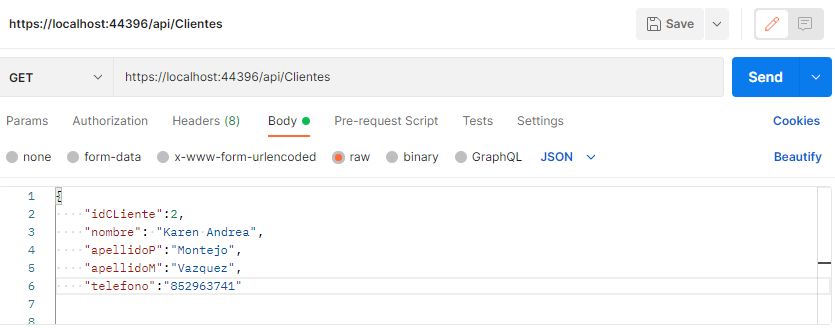
}

Muestra de que manera se hace una búsqueda, para agregar, editar y eliminar.  
Para realizar las pruebas se usó un programa llamado Postman, para editar desde ahí, y este fue el resultado:

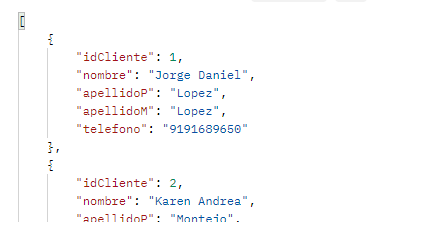
En primer lugar se tenía que establecer por medio de una URL el host de nuestra API, y nos devolvió solamente un [] vacío, lo cual teníamos que agregar según nuestra base de datos, en este caso fue de clientes, donde solamente tenía pocos parámtros

Figura que muestra que debemos poner nuestro host, y por medio del get se obtiene los resultados que encuentre según la base de datos

agregar



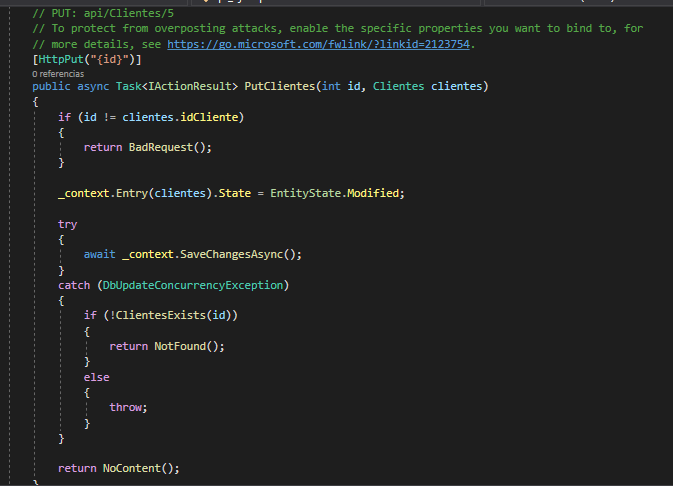
En el apartado de body, raw, y seleccionar json, para pasar los nuevos clientes.

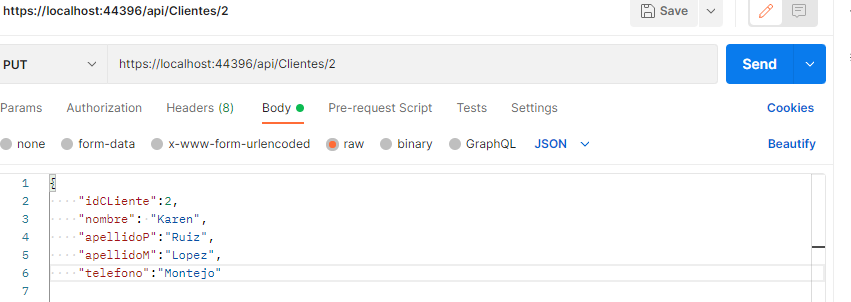


Nos vamos de nuevo al Get, y obtenemos los resultados, lo cual se ilustra de la siguiente manera

editar

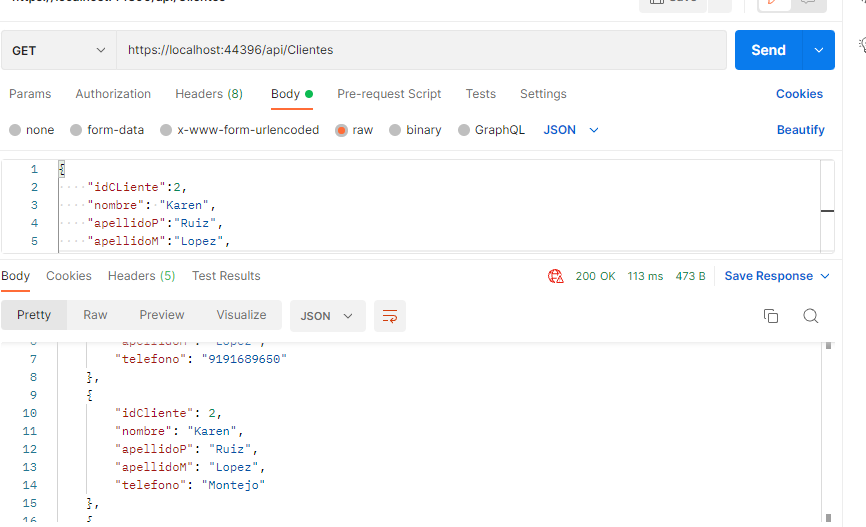
Para editar, hay que ir al apartado de Put, y en el url pasarle el id, ya que es un valor que pide en el PUT





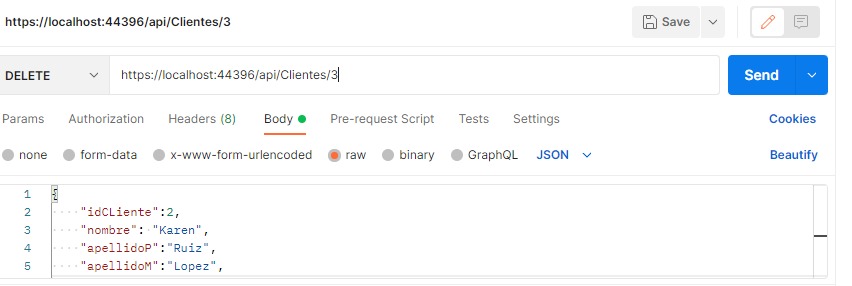
Por lo tanto agrego el id 2, picamos al botón de send, y se edita

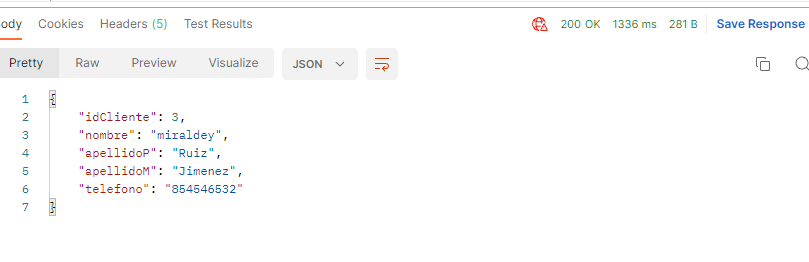
Picamos al send, y notamos que sí se editó



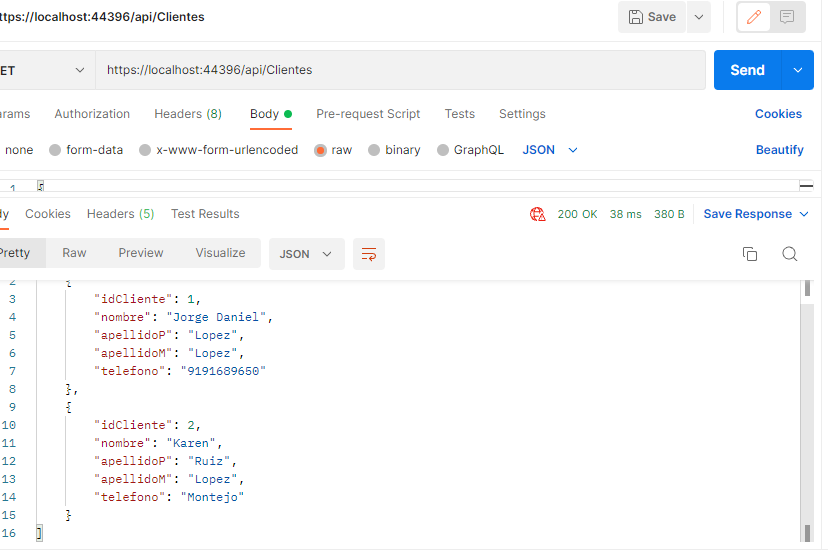
Como en la forma anterior, nuevamente hay que pasarle un parámetro, que será el id del que vamos a eliminar

Eliminar





Muestra lo que se eliminó



Y efectivamente se eliminó

De esta forma usamos un API, que es un CRUD sencillo

Open Api

OpenAPI es un estándar para la descripción de las interfaces de programación, o application programming interfaces (API). La especificación OpenAPI define un formato de descripción abierto e independiente de los fabricantes para los servicios de API. En particular, OpenAPI puede utilizarse para describir, desarrollar, probar y documentar las API compatibles con REST.

La actual especificación OpenAPI surgió del proyecto predecesor Swagger. La empresa de desarrollo SmartBear sometió la especificación existente de Swagger a una licencia abierta y dejó el mantenimiento y desarrollo posterior en manos de la iniciativa OpenAPI.

Una cuestión que puede causar confusión es la distinción entre OpenAPI y Swagger. OpenAPI es una especificación, es decir, una descripción abstracta que no está ligada a una aplicación técnica concreta. Hasta la versión 2.0, esta especificación todavía se llamaba Swagger y luego fue renombrada como especificación OpenAPI. Sin embargo, las herramientas proporcionadas por SmartBear, la empresa que la desarrolló originalmente, siguen existiendo con el nombre de Swagger.

Con OpenAPI, una API puede describirse de manera uniforme. Esto se conoce como “definición API” y se genera en un formato legible por máquina. En particular, se utilizan dos lenguajes: YAML y JSON.

La especificación OpenAPI define una serie de propiedades que pueden utilizarse para desarrollar una API propia. Estas propiedades se agrupan en los llamados **objetos** (en inglés, *objects*). En la actual versión 3.0.3, OpenAPI define la estructura de los siguientes objetos, entre otros:

* **Info Object:** versión, nombre, etc. de la API.
* **Contact Object:** datos de contacto del proveedor de la API.
* **License Object:** licencia bajo la cual la API proporciona sus datos.
* **Server Object:** nombres del host, estructura del URL y puertos del servidor a través del cual se dirige la API.
* **Components Object:**componentes encapsulados que pueden utilizarse varias veces dentro de una definición de API.
* **Paths Object:** rutas relativas a los puntos finales de la API que se utilizan junto con el servidor del objeto.
* **Path Item Object**: operaciones permitidas para una ruta específica como GET, PUT, POST, DELETE.
* **Operation Object:**especifica, entre otras cosas, los parámetros y las respuestas del servidor que se esperan de una operación.

**¿Cuáles son las áreas de aplicación de OpenAPI?**

En general, OpenAPI se utiliza para describir API REST de manera uniforme. Como esta descripción, es decir, la definición API, está disponible en un formato legible por máquina, se pueden generar automáticamente diversos artefactos virtuales a partir de ella. En concreto, estos incluyen:

* Creación de documentación API: La documentación basada en HTML se genera automáticamente a partir de la definición API legible por máquina. Esta sirve como material de consulta para los desarrolladores que acceden a los servicios API. Si la definición API cambia, la documentación se vuelve a generar para que ambas concuerden.
* Creación de conexiones en diferentes lenguajes de programación: Con las herramientas apropiadas, se puede crear una biblioteca de software adecuada del lado del cliente a partir de la definición API en un lenguaje de programación compatible. Esto permite a los programadores de todo tipo acceder a la API. La biblioteca de software se incorpora de manera convencional. Por lo tanto, el acceso a los servicios de API tiene lugar, por ejemplo, mediante el acceso a las funciones dentro del mismo entorno de programación.
* Elaboración de casos de prueba: Cada componente de un software debe someterse a diversas pruebas para asegurar su funcionalidad. En concreto, es preciso volver a probar un componente de software cada vez que se cambia el código subyacente. A partir de la definición API, se pueden generar estos casos de prueba automáticamente para poder comprobar la funcionalidad de los componentes del software en todo momento.

En última instancia, cabe señalar que no todas las API pueden representarse utilizando OpenAPI. Sin embargo, las API REST son compatibles sin duda.

Para el proyecto anterior bastó con hacerle algunas modificaciones para poder implementar nuestro swagger, el cual en nuestro startups.cs agregamos la siguiente línea de comando

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version = "v1" });

// Set the comments path for the Swagger JSON and UI.

var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";

var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);

c.IncludeXmlComments(xmlPath);

});

Para tener nuestros servicios

Posteriormente agregamos el sigguiente commando para hacer que funcione correctamente

app.UseHttpsRedirection();

//app.UseMvc();

// Enable middleware to serve generated Swagger as a JSON endpoint.

app.UseSwagger();

// Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),

// specifying the Swagger JSON endpoint.

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");

});

Dentro del mismo startup.cs

Lo único que nos hace falta es darle indicaciones de lo que sucede en cada caso, asi que tenemos que ir a nuestro controlador para agregar algunos comentarios, al poner tres veces “///” en automático nos da algo como esto

/// <summary>

/// Edita un cliente de la base de datos, donde existen dos parametros de seleccion

/// </summary>

/// <param name="id"></param>

/// <param name="clientes"></param>

/// <returns></returns>

Lo cuál es la forma de hacer un comentario.

Lo cual a la vista es de esta forma



Typescript

Anotaciones:

TypeScript (TS) es un lenguaje de programación construido a un nivel superior de JavaScript (JS). Esto quiere decir que TypeScript dota al lenguaje de varias características adicionales que hacen que podamos escribir código con menos errores, más sencillo, coherente y fácil de probar, en definitiva, más limpio y sólido.

Fue creado por Microsoft en 2012 y, desde entonces, su adopción no ha hecho más que crecer. Especialmente, desde que Google decidió adoptarlo como lenguaje por defecto para desarrollar con Angular. Aunque, hoy en día, podemos desarrollar con TypeScript en cualquiera de los frameworks o librerías más punteras, como son React para el frontend o Node para el backend.

TypeScript va un paso más allá de ECMAScript 6 o posterior y añade más funcionalidad a ECMAScript, como tipado fuerte, anotaciones o módulos.

Como TypeScript es un superconjunto de JavaScript, todo el código escrito en JS es válido para TS. Pero no lo contrario. Esto lo podemos comprobar gracias a los navegadores web. Ya que no entienden el código escrito en TypeScript y, para que lo hagan, es necesario hacer una ‘’transpilación’’ del código a JavaScript.

La transpilación es aplicable hacia cualquier versión de ECMAScript, tanto para navegadores antiguos como para navegadores más actuales.

TypeScript vs. JavaScript

Si profundizamos más en TypeScript y JavaScript, obtenemos una **comparativa más exhaustiva** de ambos:

* **Tipado estático, genérico, estructural y enumerados:** en JS no existen como tal los tipados, aunque los enumerados se pueden simular con clases sencillas. Mientras que TS es un lenguaje fuertemente tipado, donde se pueden crear tipos genéricos o interfaces.
* **Modularización:**TS ofrece un soporte directo para módulos, mientras que JS lo hace a través de ECMAScript 6.
* **Tuplas:** JS no las soporta, pero si TS.
* **Orientación a objetos:** la sintaxis de TS para la [programación orientada a objetos](https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/) es muy similar a la de otros lenguajes como [Java](https://profile.es/blog/variables-tipos-datos-java/) o C#. Y además añade clases abstractas y modificadores de acceso, entre otras características. En JS también se puede programar orientado a objetos, pero es algo más complejo.
* **Decoradores:** JS no tiene soporte para decoradores, mientras que TS si.
* **Interfaces:**Como hemos mencionado, en TS las interfaces son imprescindibles, brindándote la posibilidad de crear escenarios más avanzados. En JS no existe soporte para interfaces.

Con esta comparativa podemos observar que **TypeScript es un lenguaje muchísimo más maduro que JavaScript .**Y, aunque sea más difícil de aprender, la curva de aprendizaje tampoco es muy complicada en relación a los beneficios que te va a aportar como desarrollador para el futuro.

¿Utilizar TS o JS?

En **proyectos pequeños**, usar TypeScript es un poco brusco, por lo que quizá no merezca la pena. Ahora bien, en **proyectos medianos o grandes**, y sobre todo en un equipo de desarrollo, escribir código en TS ofrece grandes ventajas que se van a notar a corto y a largo plazo.

Si trabajas con **frameworks**, usar TypeScript es casi una obligación. **Angular**, como hemos dicho antes, está escrito en este lenguaje y, por tanto, su documentación. Si a esto le sumamos que en todas las empresas de **desarrollo software** los principales proyectos utilizan alguno de estos frameworks o librerías, blanco y en botella.

Mi consejo sería que aprendieras TypeScript ya que, si quieres [desarrollarte como profesional](https://profile.es/blog/desarrollador-senior/), va a ser más útil. Además de llevarte el aprendizaje de JavaScript y una **base muy sólida para el uso de Angular.**

Data Anotations

Básicamente, DataAnnotations nos permite llevar a cabo validaciones de datos de acuerdo a nuestras necesidades. Esas necesidades son decoraciones que indicaremos a los miembros de nuestras entidades y Dtos. Una decoración corresponderá a una validación.

Para «jugar» con DataAnnotations, deberemos agregar un ensamblado a las referencias de nuestro proyecto. Este ensamblado es System.ComponentModel.DataAnnotations.

**¿Cómo marcamos el tipo de validación de un miembro?**

Para marcar el tipo de validación de un miembro, utilizaremos diferentes atributos.

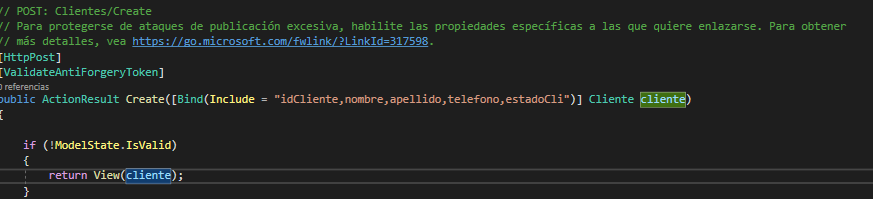
Estos atributos pueden anidarse de manera que un campo pueda tener más de una validación.

Entre estos atributos encontramos los siguientes:

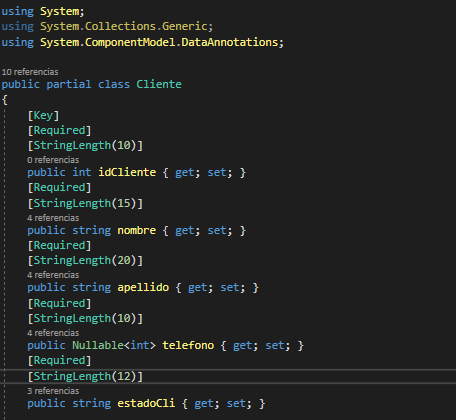
* ***Required***: perteneciente a *System.ComponentModel.DataAnnotations.RequiredAttribute*, marca que el miembro debe tener un campo obligatorio. Esta decoración puede ser utilizada junto a ***ErrorMessage*** para indicar un mensaje personalizado de error en el caso de que no se cumpla esta validación.
* ***Range***: perteneciente a *System.ComponentModel.DataAnnotations.RangeAttribute*, marca un rango de valores entre los que debe estar comprendido el valor pasado al miembro.
* ***StringLength***: perteneciente a *System.ComponentModel.DataAnnotations.StringLengthAttribute*, indica un tamaño del campo ***string***. Esta decoración puede ir en conjunción con ***MinimunLength*** para indicar incluso un tamaño mínimo del campo ***string***.
* ***RegularExpression***: perteneciente a *System.ComponentModel.DataAnnotations.RegularExpressionAttribute*, indica una expresión regulada que debe ser utilizada para validar el miembro.
* ***DataType***: perteneciente a *System.ComponentModel.DataAnnotations.DataTypeAttribute*, indica un nombre de un tipo adicional que debe asociarse a un campo de datos.
* ***CustomValidation***: perteneciente a *System.ComponentModel.DataAnnotations.CustomValidationAttribute*, nos permite validar a través de validaciones personalizadas.

Debemos de tener nuestro proyecto previamente creado para realizar nuestro data model.

A lo cual en nuestro controlador debemos de poner una condición, que hará que nos devuelva al mismo modelo, o que no aplique el cambio que deseas realizar.



Con esto nos hace una pequeña validación, a lo consiguiente lo que se debe de realizar es poner nuestras validaciones, en nuestro modelo donde está recibiendo la información de la tabla.



Según esté nuestra base de datos tienes que poner el stringLength.



Se tiene que agregar el siguiente código para que funcione nuestros required, y ya todo el código queda de la siguiente manera



Una por campo, porque esto funciona de forma individual. Y el código va en nuestra vista, la vista que contiene el created, y de igual forma debes de hacerlo por cada vista que requieras usarlo.



Por lo tanto no podremos acceder a ingresar nuestros datos, a menos que cumplamos con lo que pide.

Inyección de Independencias

La **inyección de dependencias** es una de las técnicas utilizadas para implementar el principio de inversión de dependencias.

**¿Qué entendemos por inyección de dependencias?**

Cuando tenemos un objeto que necesita de otro para funcionar correctamente, tenemos definida una dependencia. Esta **dependencia** puede ser altamente acoplada o levemente acoplada. Si el acoplamiento es bajo el objeto independiente es fácilmente cambiable; si por el contrario es altamente acoplado, el reemplazo no es fácil y dificulta el diseño de los tests.

La inyección de dependencias es una metodología utilizada en los patrones de diseño que consiste en especificar comportamientos a componentes.

Se trata de extraer responsabilidades a un componente para delegarlas a otros componentes, de tal manera que cada componente solo tiene una responsabilidad (Principio de Responsabilidad Única). Estas responsabilidades pueden cambiarse en tiempo de ejecución sin ver alterado el resto de comportamientos.

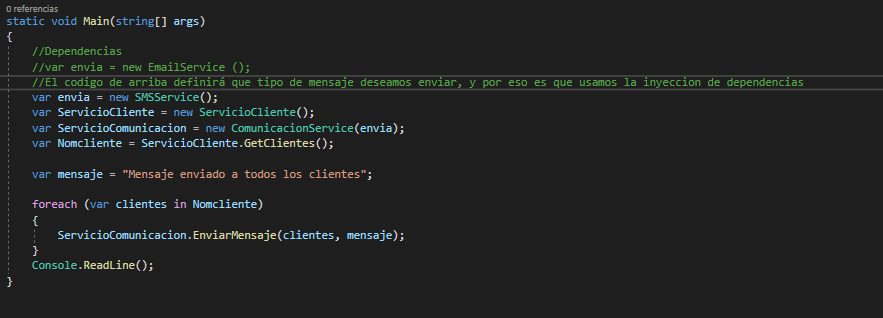
**La inversión de dependencias es un principio** que describe un conjunto de técnicas destinadas a **disminuir el acoplamiento entre los componentes de una aplicación**. Es uno de los principios SOLID más populares y utilizados en la creación de aplicaciones, *frameworks* y componentes por las ventajas que aporta a las mismas.

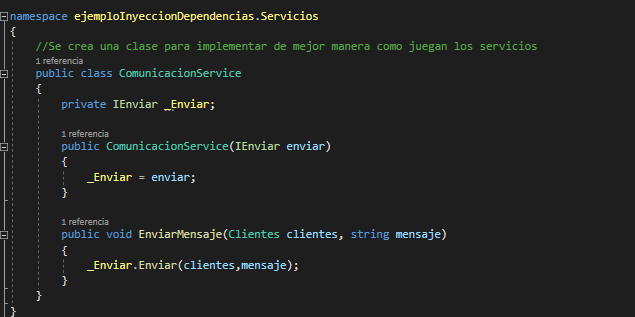
La inversión de dependencias suele también conocerse como **inversión de control**. En inglés, los términos más frecuentemente utilizados son "dependency inversion", abreviado como **"DI"**, e "inversion of control" o simplemente **"IoC"**.

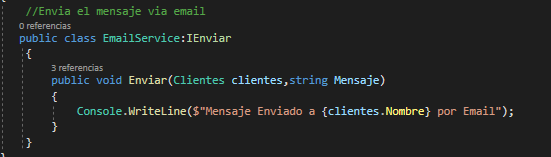
Muy resumidamente, **el Principio de Inversión de Dependencias propone evitar las dependencias rígidas entre componentes** mediante las siguientes técnicas:

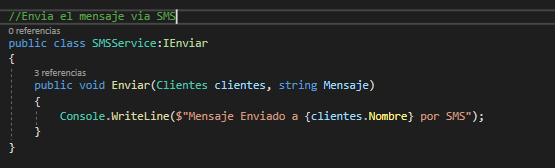
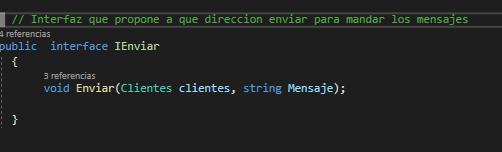
* **Utilizar abstracciones** (interfaces) en lugar de referencias directas entre clases, lo que facilita que podamos reemplazar componentes con suma facilidad.
* **Hacer que una clase reciba referencias a los componentes que necesite para funcionar**, en lugar de permitir que sea ella misma quien los instancie de forma directa o a través de factorías.

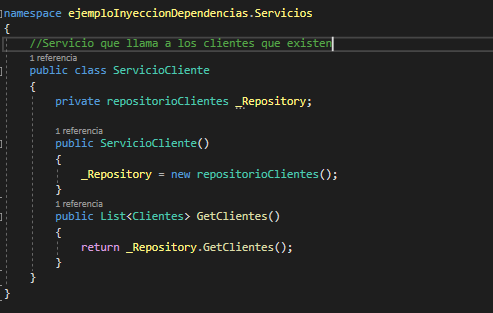
Para ejemplificar lo siguiente, tenemos el siguiente Main, que lo que hace es recibir y mandar los mensajes de acuerdo de que servicio necesitamos enviar, en este caso, queremos mandar mensajes a nuestros clientes, pero en ocasiones lo hacemos por SMS o Por Correo (aunque puede ser por cualquier servicio que queremos realizar), para ello se tiene que hacer una instancia de clase, para poder acceder lo que hay detrás de ello, y enviar de que forma necesitamos que llegue el mensaje. Así que dejaré los siguientes códigos por servicios (clases) que se está usando para que esto sea funcional

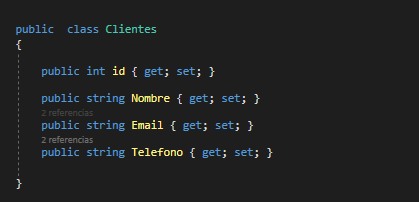


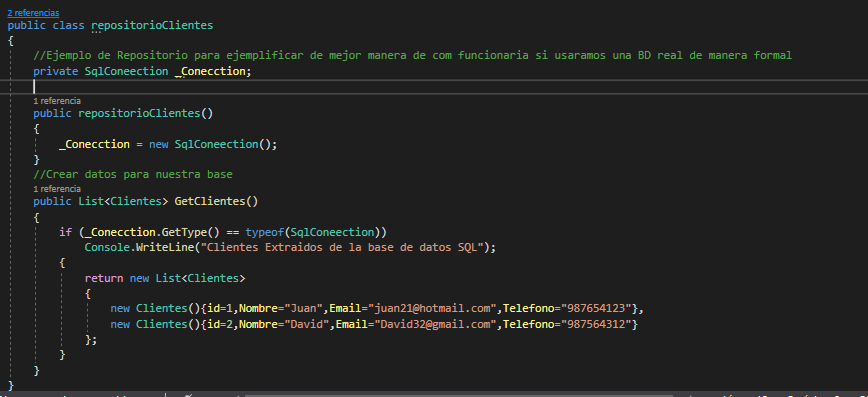
El servicio de comunicación, que es como un mediador para recibir y mandar el mensaje



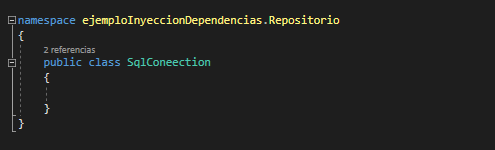


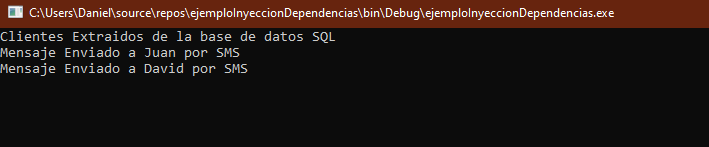


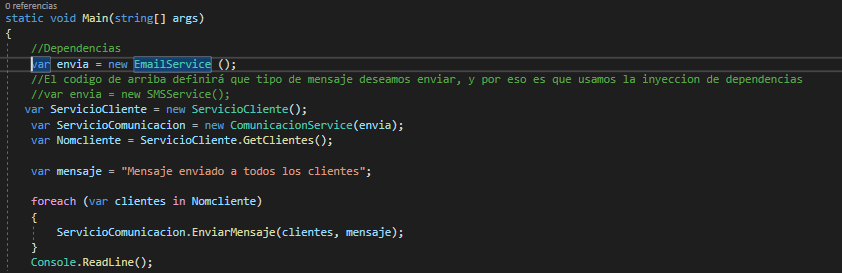
Las propiedades que servirán de tabla de clientes

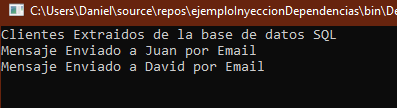
Repositorio de clientes, que es donde simulamos que tenemos los datos en nuestra base de datos y agregamos los clientes que necesitamos

Con el fin de simular que tenemos una base de datos SQL, hacemos parecer que tenemos una



Este es el mensaje que da, si modificamos el main, como esta al principio, y necesitamos enviarlo por correo, hacemos lo siguiente

Solamente activamos el email service y el mensaje muestra lo siguiente



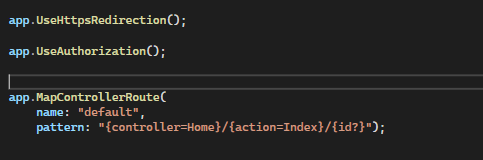
En conclusión, esto nos ayuda a compactar código y a reutilizar, sin necesidad de hacer muchas líneas de código, en este caso uno por uno, para que nos muestre los resultados que deseamos tener.

Así que es una buena forma para programar, y realizar nuestros servicios.

Enrutamientos en los APIS

En ASP.NET Web API, un *controlador* es una clase que maneja solicitudes HTTP. Los métodos públicos del controlador se denominan *métodos de acción* o simplemente *acciones* . Cuando el marco de API web recibe una solicitud, enruta la solicitud a una acción.

Para determinar qué acción invocar, el marco utiliza una *tabla de enrutamiento* . La plantilla de proyecto de Visual Studio para Web API crea una ruta predeterminada:



Si aloja una API web, debe establecer la tabla de enrutamiento directamente en el HttpSelfHostConfigurationobjeto.

Cada entrada en la tabla de enrutamiento contiene una *plantilla de ruta* . La plantilla de ruta predeterminada para la API web es "api/{controller}/{id}". En esta plantilla, "api" es un segmento de ruta literal y {controller} e {id} son variables de marcador de posición.

Cuando el marco de API web recibe una solicitud HTTP, intenta hacer coincidir el URI con una de las plantillas de ruta en la tabla de enrutamiento. Si ninguna ruta coincide, el cliente recibe un error 404. Por ejemplo, los siguientes URI coinciden con la ruta predeterminada:

* /api/contactos
* /api/contactos/1
* /api/productos/gizmo1

Sin embargo, el siguiente URI no coincide porque carece del segmento "api":

* /contactos/1

Una vez que se encuentra una ruta coincidente, la API web selecciona el controlador y la acción:

* Para encontrar el controlador, la API web agrega "Controlador" al valor de la variable {controlador} .
* Para encontrar la acción, Web API busca el verbo HTTP y luego busca una acción cuyo nombre comience con ese nombre de verbo HTTP. Por ejemplo, con una solicitud GET, Web API busca una acción con el prefijo "Get", como "GetContact" o "GetAllContacts". Esta convención se aplica solo a los verbos GET, POST, PUT, DELETE, HEAD, OPTIONS y PATCH. Puede habilitar otros verbos HTTP usando atributos en su controlador. Veremos un ejemplo de eso más adelante.
* Otras variables de marcador de posición en la plantilla de ruta, como {id}, se asignan a parámetros de acción.

**verbos HTTP**

En lugar de usar la convención de nomenclatura para verbos HTTP, puede especificar explícitamente el verbo HTTP para una acción decorando el método de acción con uno de los siguientes atributos:

* [HttpGet]
* [HttpPut]
* [HttpPost]
* [HttpDelete]
* [HttpHead]
* [HttpOptions]
* [HttpPatch]

GitFlow

Es un flujo de trabajo aplicado a un repositorio Git. Vincent Driessen fue el encargado de popularizarlo, definiendo un modelo estricto de ramificación diseñado en torno a los lanzamientos del proyecto. Es ideal para proyectos que lleven una planificación de entregas iterativas. Permite la paralelización del desarrollo mediante ramas independientes para la preparación, mantenimiento y publicación de versiones del proyecto así como soporta la reparación de errores en cualquier momento.

**Ramas principales**

Todo proyecto, por defecto, debería tener al menos dos ramas infinitas para su desarrollo. Esta metodología define que deben existir dos ramas principales:

* master
* develop

Diagrama

Descripción generada automáticamente

**Rama master**

Contiene cada una de las versiones estables del proyecto. Cualquier *commit* que subamos en esta rama debe estar preparado para que se pueda incluir en producción.

**Rama develop**

Contiene el código de desarrollo de la siguiente versión planificada del proyecto. En ella se incluirán cada una de las nuevas características que se desarrollen. Esta rama puede incorporarse tanto en una rama *release* (que veremos más adelante) como en la rama *master*, para su despliegue en producción.

**Ramas de apoyo**

Junto a las ramas *master* y *develop*, existe un conjunto de ramas de apoyo, que como hemos descrito anteriormente, su objetivo es el permitir el desarrollo en paralelo entre los miembros del equipo, la resolución de problemas en producción de forma rápida, etc. A diferencia de las ramas principales, estas están limitadas en tiempo. Serán eliminadas eventualmente. Los diferentes tipos de ramas que se usarán son:

* feature
* release
* hotfix

**Ramas feature**

* Estas ramas tienen que surgir de la rama develop. Cada una de estas ramas almacenan código de desarrollo con nuevas características. Típicamente existen solamente en los repositorios locales de los desarrolladores y no en el repositorio origen. Una vez terminado su desarrollo, se incorporarán nuevamente a la rama develop, que contendrá la última versión de código en desarrollo.
* Convención de nombres: estas ramas se pueden nombrar de cualquier forma, excepto master, develop, release-\*, o hotfix-\*.

**Ramas release**

Como las ramas feature, las ramas release también tienen que surgir de la rama develop. Contienen el código de la versión que se va a liberar próximamente. Es un paso previo y preparatorio para la versión definitiva de producción. En ella se incluye todo el código de develop necesario para el lanzamiento. Puede que contenga algún error pequeño que se debe de arreglar en este momento para no incluirlo en producción. Una vez finalizada la rama, esta se debe incluir tanto en la rama develop como en la rama master.

Convención de nombres: deben de seguir la siguiente convención: release-\*, sustituyendo el \* por el número de versión (1.1, 2.3, 4.7, etc)

**Ramas hotfix**

Estas ramas surgen de la rama *master*. Contienen una versión de producción con un error que se desea arreglar urgentemente. Una vez arreglado el error, se incluye el contenido de esta rama en las ramas *master* y *develop* para subsanar el error. Además, hay que marcar la versión arreglada de producción con un *tag* en la rama *master*.

*Convención de nombres:* deben de seguir la siguiente convención: ***hotfix-\****, sustituyendo el \* por el número de la revisión (1.1.5, 2.3.1, 4.7.9, etc)