

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGs Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

Este diseño se corresponde con el video de apoyo número 2.

Diseñamos el formulario de login de nuestro sistema de ventas que tendrá este aspecto:

The mockup shows a login interface. On the left, there is a blue square containing a white storefront icon and the text 'SISTEMA DE VENTA' in white. On the right, the title 'INICIAR SESION' is displayed. Below it are two input fields: 'Usuario' and 'Contraseña'. At the bottom right, there are two buttons: a blue 'Iniciar Sesión' button with a white icon and a red 'Salir' button with a white 'X' icon.

Dentro del proyecto CapaPresentación agregamos un nuevo formulario que llamaremos Login. Ajustamos a un tamaño aproximado de 548;275 e indicamos color de fondo BackColor=blanco, FormBorderStyle=None (excluye los botones de minimizar, maximizar y cerrar) y StartPosition=CenterScreen.

Agregamos un Label y modificamos sus propiedades. La propiedad Text tiene que estar vacía, AutoSize=False para poder ajustar su tamaño, Dock=Left para que se sitúe en la parte izquierda y cambiamos el tamaño hasta aproximadamente de 219;239, Back Color=(web)SteelBlue

Agregamos una nueva etiqueta con Text SISTEMA DE VENTA con fuente tamaño 15 (Font->Size) Back Color=SteelBlue, y la fuente ForeColor=White (pestaña color web).

Para introducir el logo utilizamos el objeto IconPictureBox de FontAwesome. En IconChar buscamos el icono Store, Back Color=SteelBlue, IconColor=White;

Para la introducción de datos agregamos dos TextBox y dos label que tendrán por texto Usuario y Contraseña. La Contraseña no debe visualizarse por lo que en la propiedad PasswordChar introduciremos el valor * para ocultar lo que se escriba.

Introducimos dos IconButton de FontAwesome que visualizarán el texto Iniciar Sesión y Salir y con nombre **btningresar** y **btnsalir** respectivamente. Además, modificamos sus propiedades FlatStyle=Flat, BorderColor=blanco, BorderSize=1, ForeColor=blanco. El borde del botón lo

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

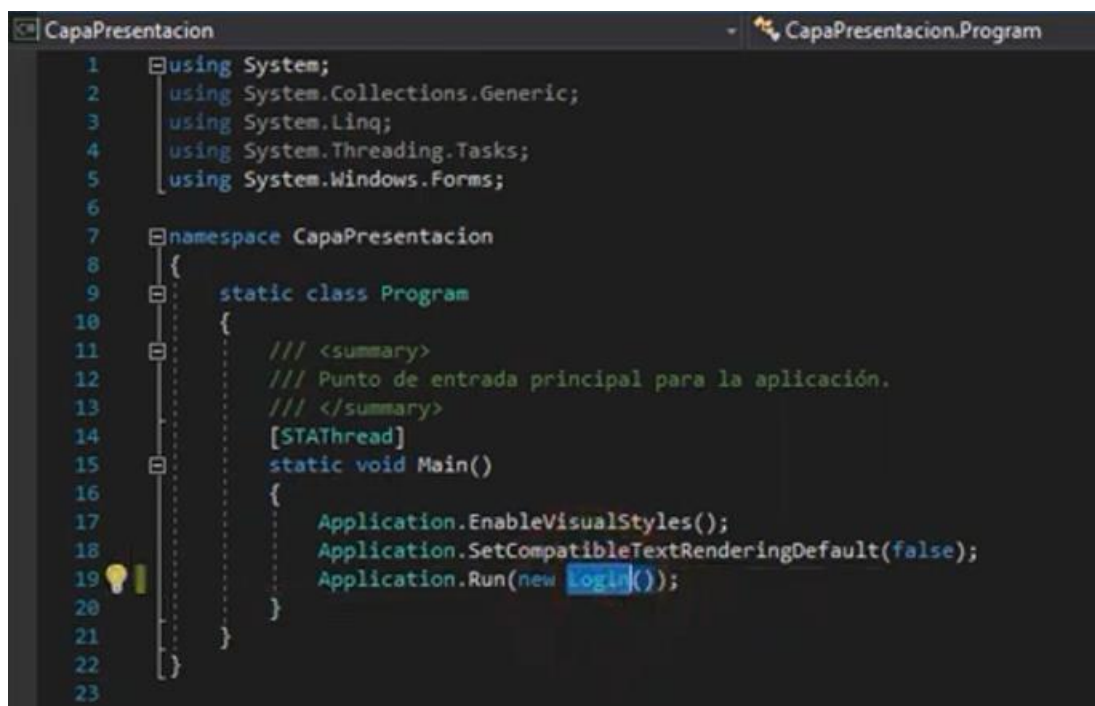
podemos indicar en FlatApearance-> BorderColor=negro

En la propiedad IconChar DoorOpen y TimeCircle respectivamente y en la propiedad BackColor RoyalBlue y Firebrick.

IconColor=blanco; IconSize=21, TextImageRelation=ImageBeforeText; TextAlign=MiddleRight.

Ajustamos su tamaño aproximadamente a 102;25 y para que cuando pasemos el ratón por encima de estos botones aparezca el icono de la mano cambiamos la propiedad Cursor y e indicamos el valor Hand.

Si ejecutamos el primer formulario que se abre es el que diseñamos anteriormente, Inicio.cs. Para que nuestra aplicación comience solicitando las credenciales tenemos que editar la clase Program.cs del proyecto CapaPresentacion y cambiar el comienzo de la aplicación al formulario Login.



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using System.Windows.Forms;
6
7  namespace CapaPresentacion
8  {
9      static class Program
10     {
11         /// <summary>
12         /// Punto de entrada principal para la aplicación.
13         /// </summary>
14         [STAThread]
15         static void Main()
16         {
17             Application.EnableVisualStyles();
18             Application.SetCompatibleTextRenderingDefault(false);
19             Application.Run(new Login());
20         }
21     }
22 }
23

```

Al pulsar el botón de salir se finalizará la aplicación.



```

private void btnCancelar_Click(object sender, EventArgs e)
{
    this.Close();
}

```

Al pulsar el botón de iniciar sesión tiene que, tras comprobar que el usuario es válido, abrir el formulario de inicio ocultando el formulario de login.

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

Apertura del siguiente formulario tras evento click:

```
private void btnIngresar_Click(object sender, EventArgs e)
{
    Inicio form = new Inicio();

    form.Show();
    this.Hide();
}
```

Cuando cerremos el formulario de inicio tiene que mostrarse de nuevo el formulario de login.

Hay un evento llamado FormClosing que se produce antes del cierre de un formulario (<https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.form.formclosing?view=windowsdesktop-6.0>)

Crearemos un nuevo evento de tipo Private en el que enviaremos un parámetro de tipo Sender que representa una referencia al control/objeto que generó el evento y el evento de cerrar formClosing. Este evento mostrará de nuevo este formulario sobre el que estamos trabajando.

En el evento de click anterior atribuiremos este nuevo evento, indicando que al formulario se una el evento que se encarga de mostrar el formulario de login que hemos ocultado anteriormente cuando se cierra el formulario Inicio.

```
private void btnCancelar_Click(object sender, EventArgs e)
{
    this.Close();
}

private void btnIngresar_Click(object sender, EventArgs e)
{
    Inicio form = new Inicio();

    form.Show();
    this.Hide();

    form.FormClosing += frm_closing;
}

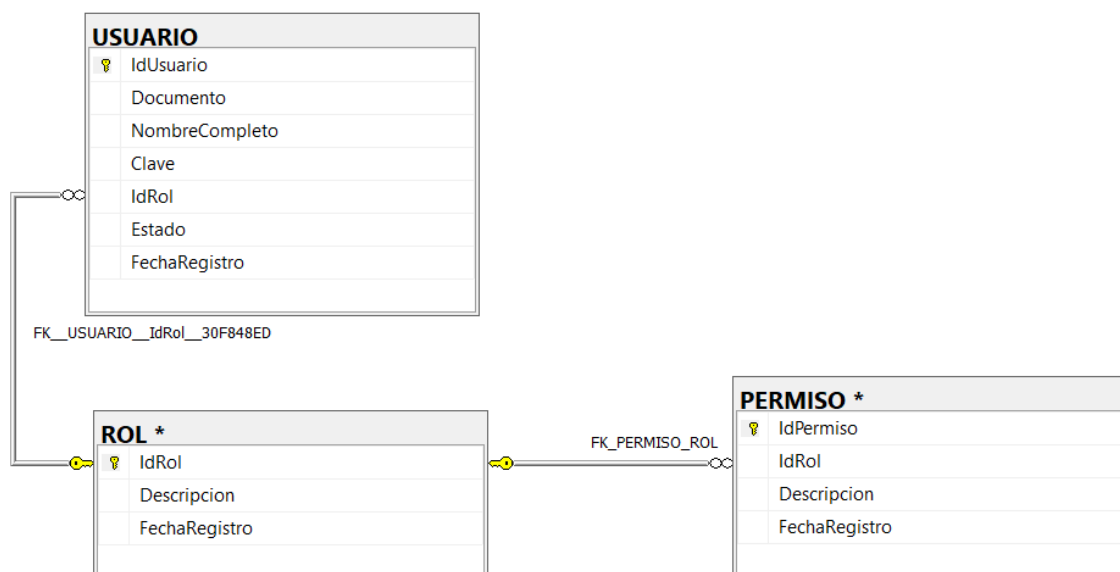
private void frm_closing(object sender, FormClosingEventArgs e) {
    this.Show();
}
```

Si lo probamos vemos que mantiene los datos en los TextBox cuando regresamos al formulario de logeo por lo que tenemos que inicializar los TextBox en el evento frm_closing.

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

```
private void frm_closing(object sender, FormClosingEventArgs e) {
    txtdocumento.Text = "";
    txtclave.Text = "";
    this.Show();
}
```

Pasaremos ahora a dar funcionalidad a nuestro formulario de logeo. Ya tenemos nuestra BBDD creada, pero está vacía. Hay tres tablas relacionadas con la identificación y permisos de los usuarios de la aplicación:



Para la funcionalidad de login solo utilizaremos los datos de Documento (usuario) y Clave almacenados en la tabla de Usuario.

Esta tabla tiene una referencia a la tabla ROL por lo que si intentamos crear un usuario y no existe el rol o este campo está vacío no se registrará el usuario.

Abrimos una nueva consulta dentro de SQL Server Management Studio:

```
--Nos situamos en nuestra base de datos con la instrucción USE
USE BDSISTEMAVENTAS;
```

```
/*Instrucción SQL INSERT INTO añade filas a nuestra tabla. Sintaxis:
```

```
INSERT INTO table_name (Column1, Column 2....)
VALUES (value1, value2, ...);
*/
```

```
--La tabla Usuario tiene una clave IdUsurio Identity que se autorrellena
--también una fecha de registro con un valor Default, estos datos no hay que introducirlos
```

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

--Estado=1 usuario activo Estado=0 usuario deshabilitado. Es lógico pensar que en la creación debería ser siempre 1.
 --No hemos indicado un valor Default en la creación de este campo, deberíamos haberlo indicado.
 --Lo primero que vamos a hacer es modificar la tabla para que el campo Estado tenga valor 1 en su creación. Esto es una condición.
 --La instrucción ALTER TABLE table_name permite modificar/añadir columnas y restricciones/condiciones a columnas de una tabla

--Consulta columnas y propiedades de una tabla
 SELECT * FROM Information_Schema.Columns WHERE TABLE_NAME = 'USUARIO' ORDER BY ORDINAL_POSITION;

--añadimos esta condición a la tabla USUARIO
 ALTER TABLE USUARIO ADD CONSTRAINT df_Usuario DEFAULT 1 FOR Estado;

--Creamos un usuario omitiendo los datos autorellenables

INSERT INTO USUARIO(Documento, NombreCompleto, Clave, IdRol)
 VALUES ('101010', 'Usuario1', '12345', 1);

--Al ejecutar nos da error porque no existe IdRol

--Creamos un Rol Administrador

INSERT INTO ROL(Descripcion) VALUES ('ADMINISTRADOR');

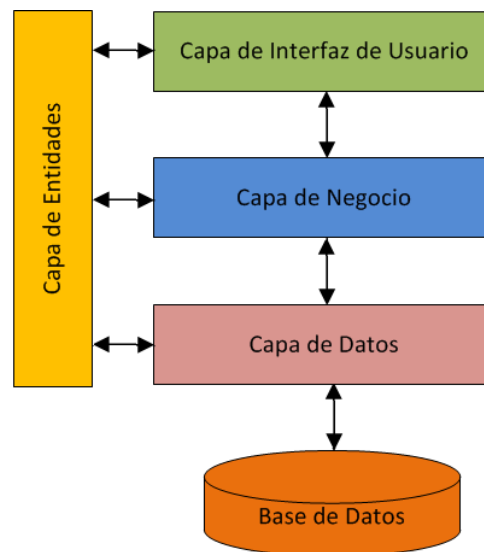
--Comprobamos que se ha creado el Rol y que tiene IdRol 1
 SELECT * FROM ROL;

--Volvemos a intentar crear un usuario omitiendo los datos autorellenables en la creación
 INSERT INTO USUARIO(Documento, NombreCompleto, Clave, IdRol)
 VALUES ('101010', 'Usuario1', '12345', 1
);

--Consultamos el usuario creado
 SELECT * FROM USUARIO;

Para esta aplicación estamos utilizando una arquitectura en capas con una comunicación entre capas definida:

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	--



Cada capa tiene una función específica:

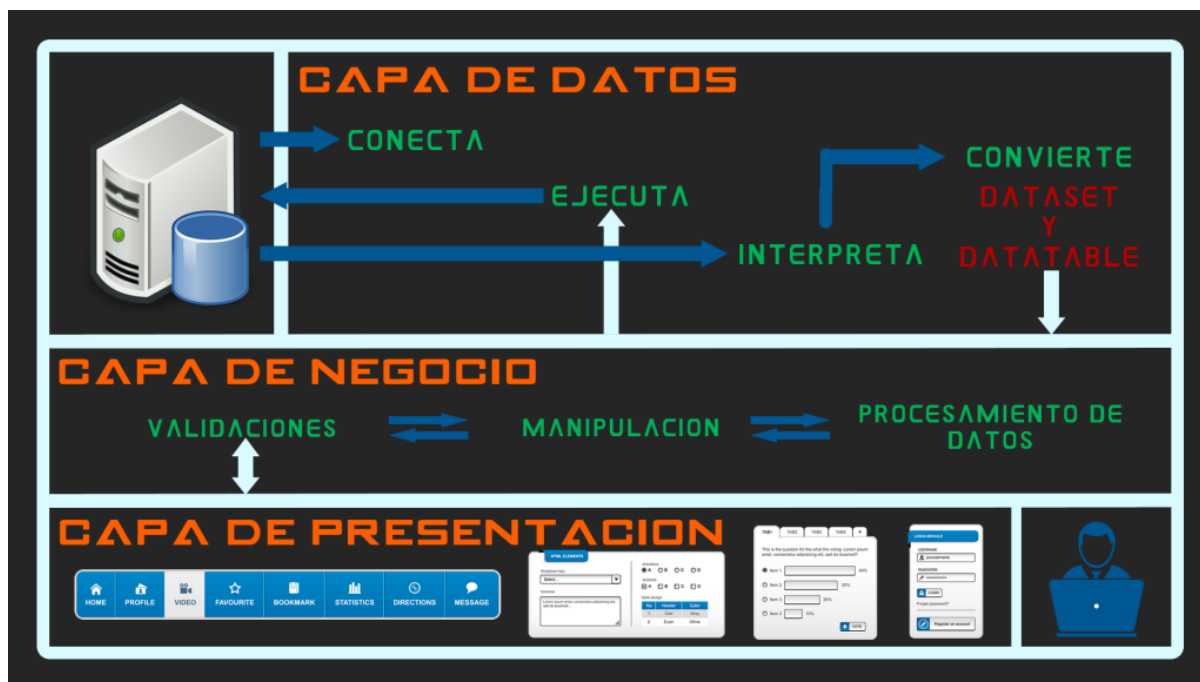
Capa Entidad: tendrá las clases con una estructura similar a las tablas de nuestra base de datos. Se comunica con todas las demás capas.

Capa de Datos: es la encargada de la comunicación con la base de datos. Aquí se ejecuta las operaciones CRUD (Create Read Update Delete). sobre nuestra BBDD. Se comunica con la capa de datos y la capa de entidades.

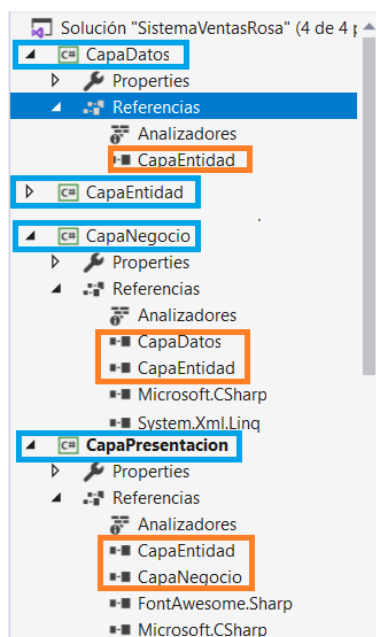
Capa de Negocio: Es la encargada de recibir las peticiones solicitadas por el usuario desde la capa de presentación (Interfaz de Usuario), Se comunica con la capa de presentación o de interfaz de usuario y la capa de entidades.

Capa de Presentación o de Interfaz de Usuario: es la encargada de interactuar con el usuario, formularios, ventanas, cuadros de diálogo que el usuario final utiliza. Se comunica con la capa de negocio y la capa entidad

	UT 2 – Práctica Sistema de Ventas Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---



Para establecer el acceso entre capas utilizaremos de las referencias de proyectos. La Capa de Entidad es la encargada de la comunicación entre todas las capas, es la encargada de enviar todos los datos a través de las clases que hemos creado por lo que añadiremos esta referencia a todas las capas. Agregamos referencias pinchando sobre el proyecto con el botón derecho->Agregar->referencia->proyecto y seleccionando.



La Capa Negocio se comunica además de con la capa entidad con la capa de datos por lo que añadimos esta referencia a esta capa.

La Capa Presentación se comunica además de con la capa entidad con la capa de negocio por lo que añadimos esta capa.

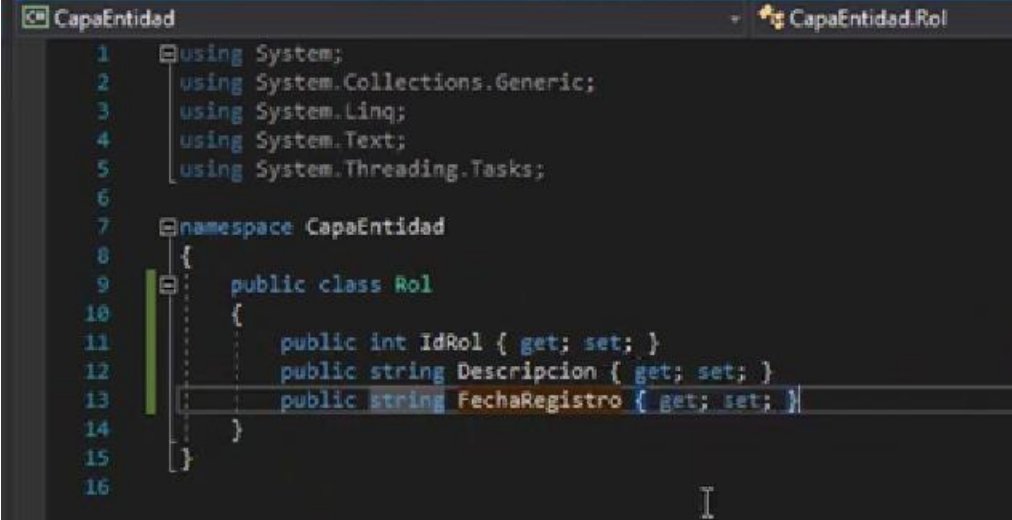
Comenzaremos creando en la capa Entidad las clases con una estructura similar a las tablas de nuestra base de datos.

Nos situamos en el proyecto CapaEntidad, por defecto se crea una clase que no vamos a utilizar por lo que podemos eliminarla.

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGs Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

Agregamos una nueva **clase Rol**. Tiene que ser **pública** para poder compartir esta clase entre las distintas capas. Recordemos que la capa entidad se comunica con todas las capas.

Añadimos a esta clase **los atributos** que van a ser las **equivalentes a las columnas** que tenemos en la tabla **con las propiedades get y set** (https://www.w3schools.com/cs/cs_properties.php). Get puede leer información de un campo y devolverla, set puede escribir información. Los campos se pueden hacer de solo lectura (si solo se usa el método get) o de solo escritura (si solo usa el método set):

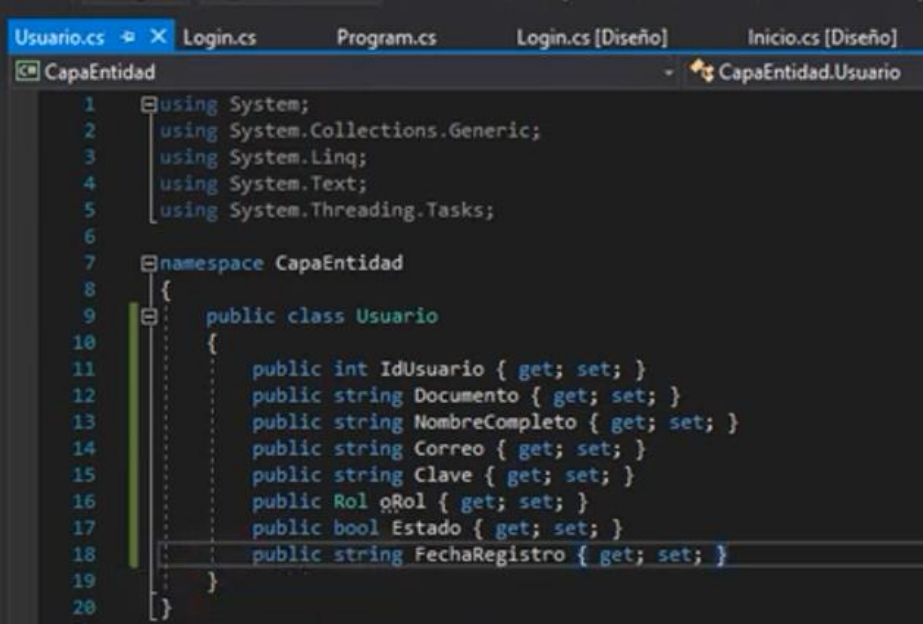


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Rol
10     {
11         public int IdRol { get; set; }
12         public string Descripcion { get; set; }
13         public string FechaRegistro { get; set; }
14     }
15 }
16

```

Agregamos una nueva **clase pública usuario** con sus respectivos atributos. En nuestra BBDD el campo usuario.IdRol hace referencia al campo Rol.IdRol. En nuestra clase vamos a mantener una relación similar indicando que la propiedad oRol es un objeto de tipo Rol.



```

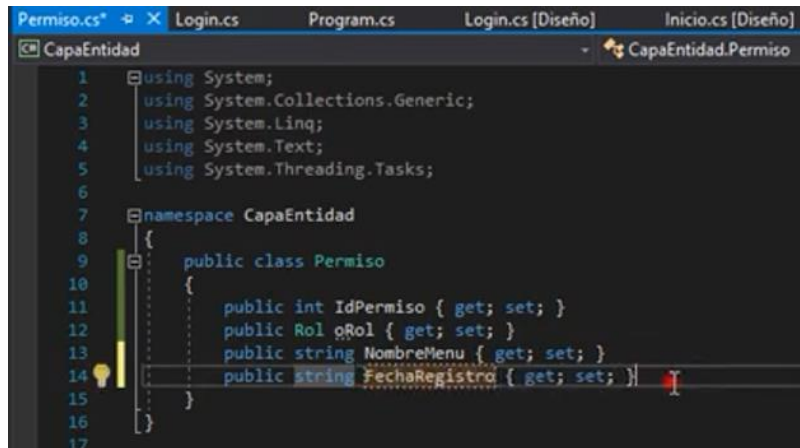
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Usuario
10     {
11         public int IdUsuario { get; set; }
12         public string Documento { get; set; }
13         public string NombreCompleto { get; set; }
14         public string Correo { get; set; }
15         public string Clave { get; set; }
16         public Rol oRol { get; set; }
17         public bool Estado { get; set; }
18         public string FechaRegistro { get; set; }
19     }
20 }
21

```


	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

Con estas dos clases ya podríamos continuar con nuestra funcionalidad de logeo pero podemos aprovechar para crear el resto de clases:

Clase pública Permiso:

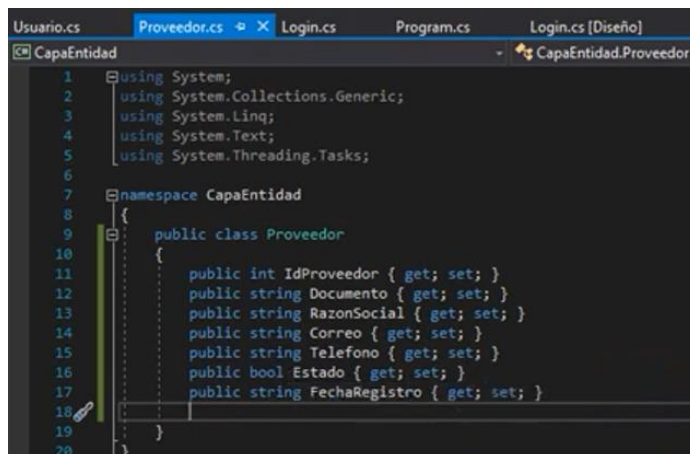


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Permiso
10     {
11         public int IdPermiso { get; set; }
12         public Rol rol { get; set; }
13         public string NombreMenu { get; set; }
14         public string FechaRegistro { get; set; }
15     }
16 }

```

Clase pública Proveedor:

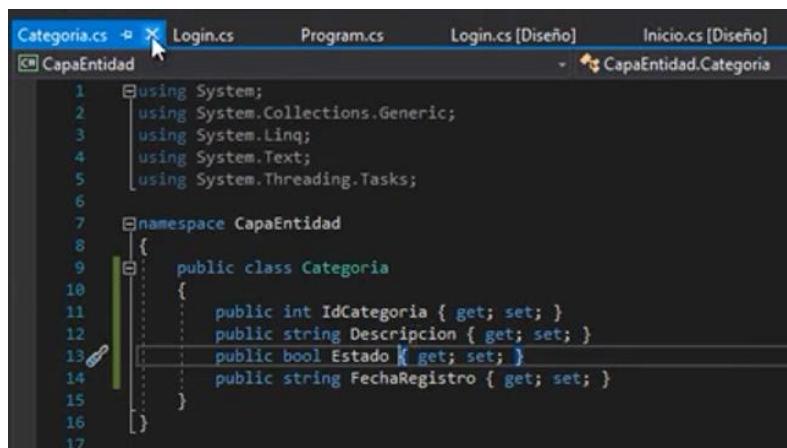


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Proveedor
10     {
11         public int IdProveedor { get; set; }
12         public string Documento { get; set; }
13         public string RazonSocial { get; set; }
14         public string Correo { get; set; }
15         public string Telefono { get; set; }
16         public bool Estado { get; set; }
17         public string FechaRegistro { get; set; }
18     }
19 }

```

Clase pública Categoría:



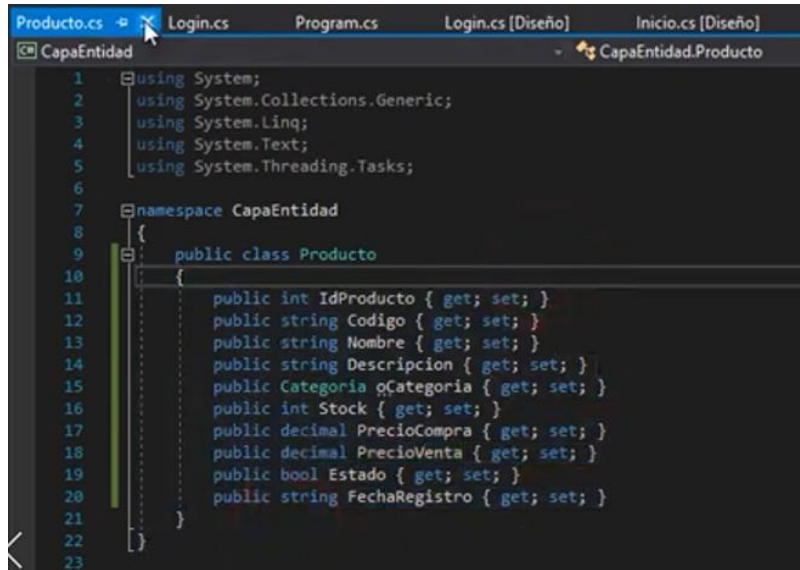
```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Categoría
10     {
11         public int IdCategoría { get; set; }
12         public string Descripción { get; set; }
13         public bool Estado { get; set; }
14         public string FechaRegistro { get; set; }
15     }
16 }

```

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

Clase pública Producto:

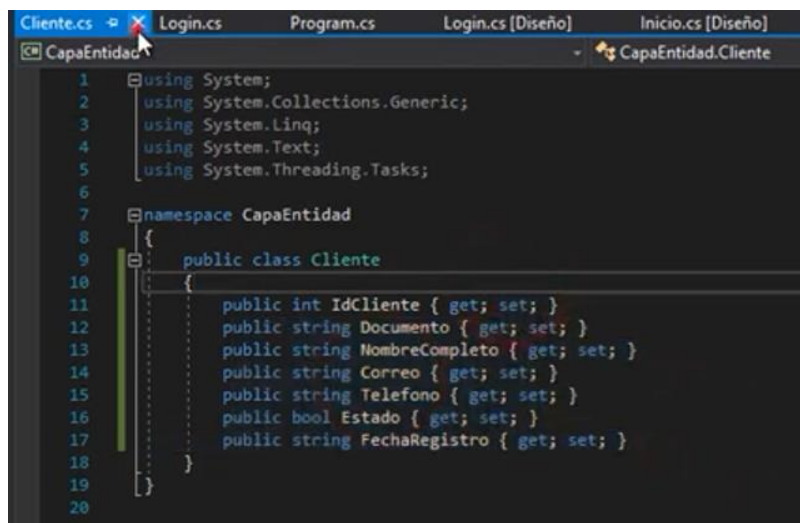


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Producto
10     {
11         public int IdProducto { get; set; }
12         public string Codigo { get; set; }
13         public string Nombre { get; set; }
14         public string Descripcion { get; set; }
15         public Categoria oCategoria { get; set; }
16         public int Stock { get; set; }
17         public decimal PrecioCompra { get; set; }
18         public decimal PrecioVenta { get; set; }
19         public bool Estado { get; set; }
20         public string FechaRegistro { get; set; }
21     }
22 }

```

Clase pública Cliente:



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Cliente
10     {
11         public int IdCliente { get; set; }
12         public string Documento { get; set; }
13         public string NombreCompleto { get; set; }
14         public string Correo { get; set; }
15         public string Telefono { get; set; }
16         public bool Estado { get; set; }
17         public string FechaRegistro { get; set; }
18     }
19 }

```

Clase pública Compra:

La tabla Compra tiene una relación 1-n con la tabla Detalle_Compra por ello incluiremos una referencia al detalle compra dentro de la clase. Una compra puede englobar varios ítems, incluimos en esta clase un objeto de tipo lista que va a contener todos los id de detalle_compra relacionados con de una compra.

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Compra
10     {
11         public int IdCompra { get; set; }
12         public Usuario qUsuario { get; set; }
13         public Proveedor qProveedor { get; set; }
14         public string TipoDocumento { get; set; }
15         public string NumeroDocumento { get; set; }
16         public decimal MontoTotal { get; set; }
17         public List<Detalle_Compra> qDetalleCompra { get; set; }
18         public string FechaRegistro { get; set; }
19     }
20 }

```

Clase pública Detalle_Compra:

La relación entre la compra y el detalle de la compra ha quedado definida en la anterior clase por lo que en la clase Detalle_Compra, podemos omitir el atributo el IdCompra:

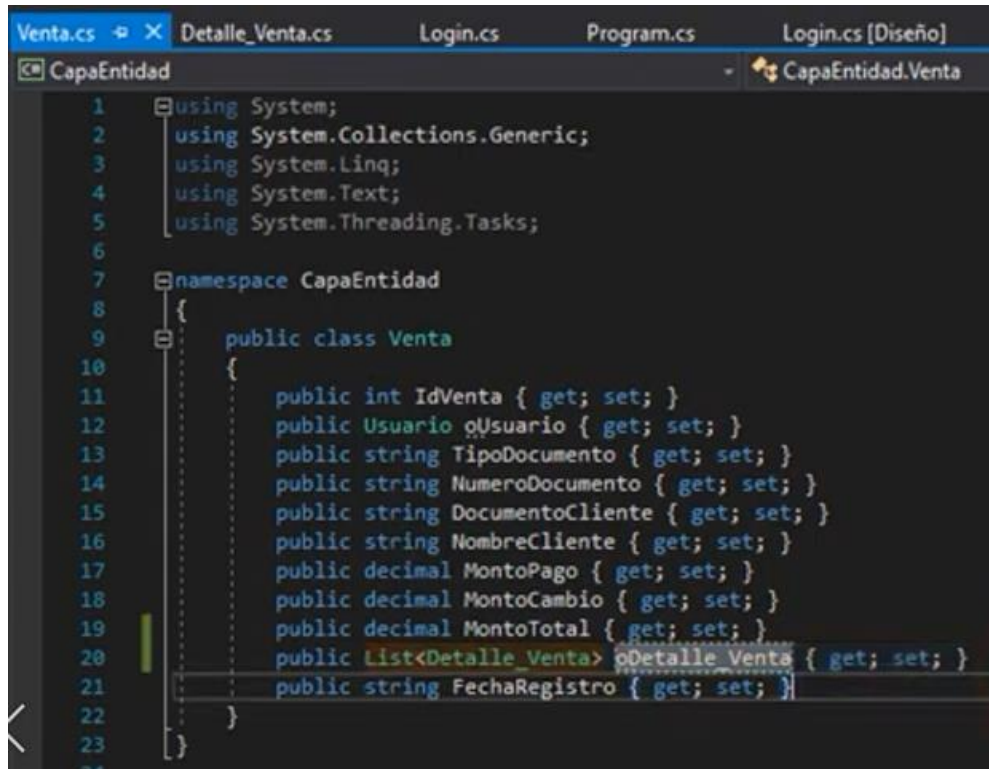
```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace CapaEntidad
8  {
9      public class Detalle_Compra
10     {
11         public int IdDetalleCompra { get; set; }
12         public Producto qProducto { get; set; }
13         public decimal PrecioCompra { get; set; }
14         public decimal PrecioVenta { get; set; }
15         public int Cantidad { get; set; }
16         public decimal MontoTotal { get; set; }
17         public string FechaRegistro { get; set; }
18     }
19 }

```

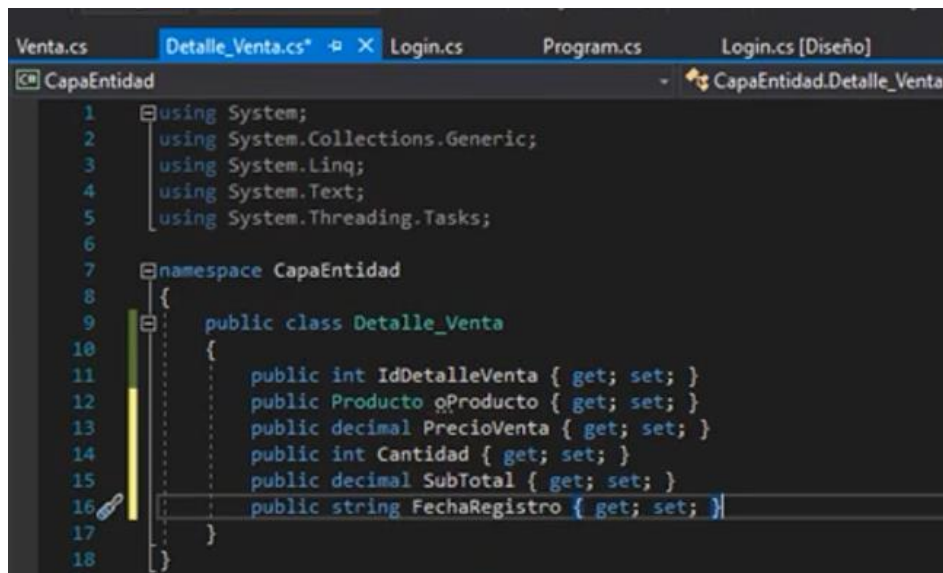
Clase pública Ventas:

Como el caso de la compra la relación entre venta y detalle venta es 1 a n por lo que seguiremos el mismo criterio que en la relación de compra incluyendo en la clase un atributo lista que nos servirá de referencia con detalle ventas



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace CapaEntidad
8 {
9     public class Venta
10     {
11         public int IdVenta { get; set; }
12         public Usuario oUsuario { get; set; }
13         public string TipoDocumento { get; set; }
14         public string NumeroDocumento { get; set; }
15         public string DocumentoCliente { get; set; }
16         public string NombreCliente { get; set; }
17         public decimal MontoPago { get; set; }
18         public decimal MontoCambio { get; set; }
19         public decimal MontoTotal { get; set; }
20         public List<Detalle_Venta> oDetalle_Venta { get; set; }
21         public string FechaRegistro { get; set; }
22     }
23 }
```

Clase pública Detalle_Ventas:



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace CapaEntidad
8 {
9     public class Detalle_Venta
10     {
11         public int IdDetalleVenta { get; set; }
12         public Producto oProducto { get; set; }
13         public decimal PrecioVenta { get; set; }
14         public int Cantidad { get; set; }
15         public decimal SubTotal { get; set; }
16         public string FechaRegistro { get; set; }
17     }
18 }
```

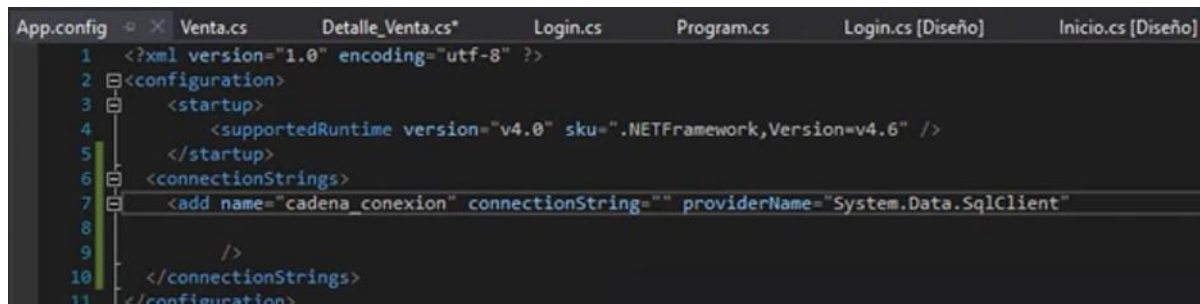
CONEXIÓN A LA BASE DE DATOS

Antes de ejecutar cualquier consulta es necesario abrir una conexión con el servidor de bases de datos. Esta conexión se crea mediante la clase `SqlConnection`. Tras la instanciación de una conexión es necesario inicializar ciertos valores relativos a la base de datos. Estos se especifican mediante la cadena de conexión.

La conexión con la base de datos se indica en el fichero `App.config` de la `CapaPresentación`.

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

Dentro de este archivo vamos a añadir una nueva etiqueta <connectionStrings> y dentro de ella añadiremos nuestra conexión:



```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <configuration>
3   <startup>
4     <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
5   </startup>
6   <connectionStrings>
7     <add name="cadena_conexion" connectionString="" providerName="System.Data.SqlClient" />
8   </connectionStrings>
9 </configuration>

```

Una cadena de conexión es una cadena de caracteres que contiene la información necesaria para establecer una conexión con una base de datos. Esta información se especifica en forma de parejas de campos **clave/valor separados por puntos y coma**, las claves son nombres de parámetro reconocidos por el proveedor de datos.

Cuando se asigna a la propiedad **ConnectionString** de un objeto SqlConnection, se realiza un análisis de la cadena de conexión. Tras este análisis, se extraen los distintos valores contenidos en ella y se asignan a las distintas propiedades de la conexión. En caso de error durante esta etapa de análisis, se produce una excepción y las propiedades de la conexión permanecen sin cambios.

Las siguientes **palabras clave** están disponibles para configurar una cadena de conexión:

Data Source

Nombre o dirección de red del servidor al que se desea establecer la conexión. Es posible especificar un número de puerto a continuación, separando los valores mediante una coma. El valor por defecto del número de puerto es el 1433.

Initial Catalog

Nombre de la base de datos sobre la que se desea realizar la conexión.

User Id

Nombre del usuario de SQL Server que se desea utilizar para establecer la conexión. Cualquier consulta ejecutada a través de la conexión se someterá a los permisos de acceso correspondientes a esta cuenta de usuario.

Password

Contraseña asociada a la cuenta de usuario de SQL Server utilizada.

Integrated Security

Valor booleano que ofrece la posibilidad de utilizar la autenticación de Windows en curso para la conexión.

	UT 2 – Práctica Sistema de Ventas Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	---	--

Connect Timeout

Tiempo (en segundos) durante el cual la aplicación espera a que se establezca la conexión. En caso de superarse este intervalo de tiempo, se produce una excepción. Por defecto, este intervalo es de 15 segundos.

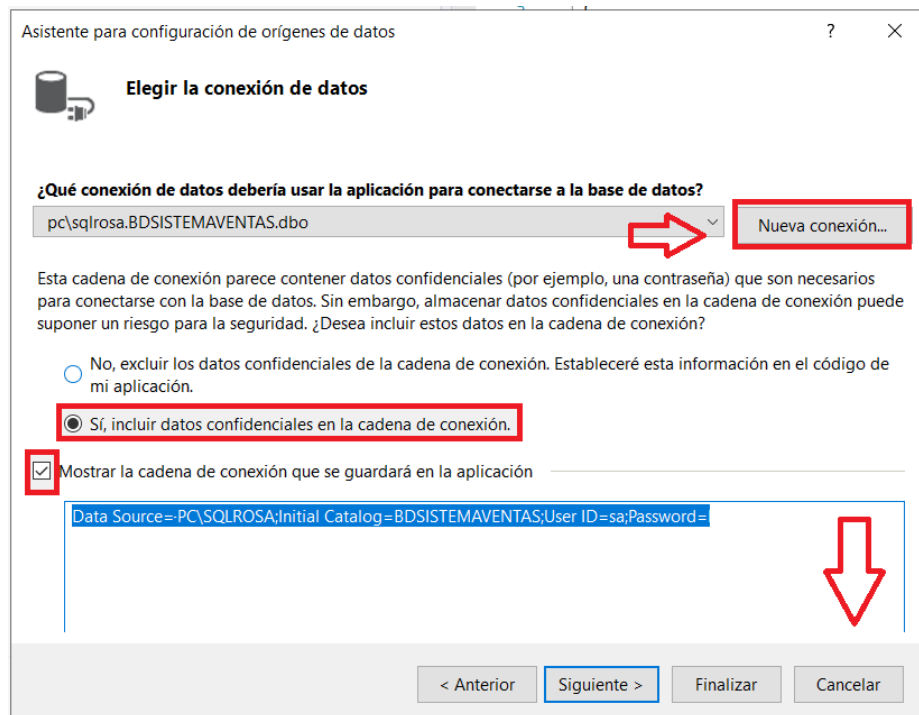
Min Pool Size

Número mínimo de conexiones que se desea conservar en el pool de conexiones.

Max Pool Size

Número máximo de conexiones que se desea conservar en el pool de conexiones.

Una forma sencilla de obtener nuestra cadena de conexión sin cometer errores es a través de la opción Proyecto->Agregar nuevo origen de datos del menú de Visual Studio



Al mostrar la cadena de conexión podremos copiarla a en nuestro connectionStrings:

```
connectionStrings="Data Source=PC\SQLROSA;Initial Catalog=BDSISTEMAVENTAS;User ID=sa;Password=12345"
```

Recordemos que **la Capa de Datos es la encargada de comunicarse con la base de datos**. Nos situamos en el proyecto CapaDatos, igual que en la CapaEntidad por defecto se crea una clase que no utilizaremos por lo que podemos eliminarla.

La clase ConfigurationManager permite tener acceso a la información de configuración del equipo, la una aplicación y del usuario. Incluye miembros que le permiten realizar leer una sección de un

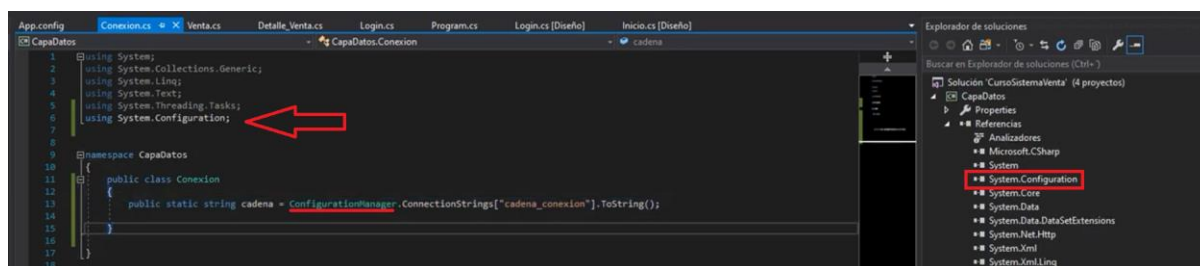
	UT 2 – Práctica Sistema de Ventas Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	---	---

archivo de configuración. Es una clase estática y por ello todos sus miembros también lo son lo que permite realizar de forma sencilla la operación de lectura de configuraciones como AppSettings o ConnectionStrings. Para usar la clase ConfigurationManager, el proyecto debe hacer referencia al **System.Configuration** (recordemos que una referencia es una entrada de un archivo de proyecto que contiene la información que Visual Studio necesita para localizar el componente o el servicio). System.Configuration **es un espacio de nombres** que contiene tipos que nos permiten leer datos de archivos .config (en nuestro caso App.config).

Comprobamos si nuestro proyecto CapaDatos tiene como referencia **System.Configuration** Si no lo tiene agregamos al proyecto una referencia. En las opciones de ensamblado buscamos una el nombre del componente **System.Configuration** y lo seleccionamos y aceptamos.

Recordemos que las declaraciones de espacio de nombres con ámbito de archivo permiten declarar que todos los tipos contenidos en ese archivo están en un único espacio de nombres. Para permitir usar los tipos definidos en un espacio de nombres hay que utilizar la directiva Using, al principio de un archivo de código fuente, antes de las declaraciones de espacio de nombres o de tipo.

Dentro de CapaDatos agregamos una nueva **clase Conexión pública** que se va a encargar de enviar a las demás capas la cadena_conexión que acabamos de configura en App.config, añadimos la directiva using SystemConfiguration antes de la declaración del espacio de nombres y almacenamos en una variable cadena nuestra cadena de conexión.



LECTURA DATOS TABLA BASE DE DATOS

En nuestra CapaDatos Comenzaremos agregando un método público llamado **CD_Usuario** con la que vamos a hacer las consultas de usuarios en nuestra BBDD y nos va a devolver una lista de todos los usuarios que tenemos en nuestra base de datos.

Agregamos dos referencias. Una a System.Datos y otra a System.Data.SqlClient.

System.Datos es un espacio de nombres que contiene clases compartidas por los proveedores de datos de .NET. Ofrece acceso a clases que representan la arquitectura de ADO.NET. ADO.NET

	UT 2 – <i>Práctica Sistema de Ventas</i> Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	--	---

permite crear componentes que administran datos de varios orígenes de datos con eficacia.
<https://docs.microsoft.com/es-es/dotnet/api/system.data.common?view=net-6.0>

El espacio de nombres **System.Data.SqlClient** es el proveedor de datos .NET para SQL Server. Entre sus clases encontramos SqlConnection que utilizaremos para realizar la conexión a BBDD SQL Server, la clase SqlDataReader que utilizaremos para leer un conjunto de filas desde una BBDD SQL Server y la clase SqlCommand para ejecutar instrucciones Transact-SQL.
<https://docs.microsoft.com/es-es/dotnet/api/system.data.sqlclient?view=dotnet-plat-ext-6.0>

Tenemos que declarar la conexión a la base de datos y realizar operaciones con esta conexión:

- Almacenar la query a realizar en una variable.
- Ejecutar la query con SqlCommand
- Recuperar datos mediante un objeto SqlDataReader que almacenaremos en nuestra lista de objetos usuario

Para declarar una conexión tenemos esta sintaxis:

```
using (SqlConnection <nombre conexión> = new
SqlConnection(<cadena de conexión>))
{
    // Realizar operaciones con la conexión
}
```

Añadimos las referencias que necesitamos.

Creamos un método listar que nos va a devolver un lista de usuarios.

Definimos una variable lista de tipo list de una lista de usuario.

Nos contamos a la base de datos utilizando using y como cadena de conexión que tenemos almacenada en la clase conexion creada anteriormente dentro de la variable cadena.

La instrucción Try-cath nos sirve para capturar los errores (<https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/try-catch>). La utilización del argumento de cath Exception permite filtrar las excepciones. Las excepciones en C# proporcionan una forma de controlar las condiciones de error del nivel de sistema y de la aplicación.

Dentro del try almacenamos en una cadena la consulta a realizar y utilizaremos SqlCommand para

	UT 2 – Práctica Sistema de Ventas Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	---	---

ejecutar el comando indicando la query y la conexión que hemos abierto (<https://docs.microsoft.com/es-es/dotnet/api/system.data.sqlclient.sqlcommand?view=dotnet-plat-ext-6.0>), abrimos nuestra cadena de conexión para que se pueda ejecutar el comando y leemos el resultado de nuestro comando con DataReader que vamos a almacenar en nuestra lista mediante un while que se ejecutará mientras esté leyendo registros, de modo que cada vez que lea un registro va a almacenar en nuestra lista un objeto de tipo Usuario.

// Set the connection, command, and then execute the command with query and return the reader.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  using System.Data;
8  using System.Data.SqlClient;
9  using CapaEntidad;
10
11 namespace CapaDatos
12 {
13     public class CD_Usuario
14     {
15
16         public List<Usuario> Listar() {
17             List<Usuario> lista = new List<Usuario>();
18
19             using (SqlConnection oconexion = new SqlConnection(Conexion.cadena)) {
20
21                 try
22                 {
23                     string query = "select IdUsuario,Documento,NombreCompleto,Correo,Clave,Estado from usuario";
24                     SqlCommand cmd = new SqlCommand(query, oconexion);
25                     cmd.CommandType = CommandType.Text;
26
27                     oconexion.Open();
28
29                     using (SqlDataReader dr = cmd.ExecuteReader()) {
30
31                         while (dr.Read()) {
32
33                             lista.Add(new Usuario()
34                             {
35                                 IdUsuario = Convert.ToInt32(dr["IdUsuario"]),
36                                 Documento = dr["Documento"].ToString(),
37                                 NombreCompleto = dr["NombreCompleto"].ToString(),
38                                 Correo = dr["Correo"].ToString(),
39                                 Clave = dr["Clave"].ToString(),
40                                 Estado = Convert.ToBoolean(dr["Estado"])
41                             });
42                         }
43                     }
44                 }
45                 catch (Exception ex) {
46                     lista = new List<Usuario>();
47                 }
48             }
49
50             return lista;
51         }
52     }
53 }

```

La capa de Datos se comunica con la capa de Negocio por lo que este método CD_usuario que hemos creado lo tiene que llamar nuestra capa de Negocio.

Creamos una nueva clase pública CN_Usuario en CapaNegocio que va a devolver el método de

	UT 2 – Práctica Sistema de Ventas Parte II – LOGIN	CFGS Desarrollo de Aplicaciones Multiplataforma Módulo: DDI
--	---	---

listar de la capa de Datos. Tiene que tener como referencia CapaDatos y la CapaEntidad.

Hacemos una instancia a nuestra clase CD_Usuario de la capa Datos que va a devolver la misma lista que tiene CD_Usuario.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  using CapaDatos;
8  using CapaEntidad;
9  namespace CapaNegocio
10 {
11     public class CN_Usuario
12     {
13     }
14     private CD_Usuario objcd_usuario = new CD_Usuario();
15
16
17     public List<Usuario> Listar()
18     {
19         return objcd_usuario.Listar();
20     }
21 }
22
23
24
25

```

Desde la capa de Negocio ya hay comunicación con la capa de Presentación.

Regresemos a nuestro formulario de Login y hacemos referencia a CapaNegocio y a la CapaEntidades.

Utilizando expresiones lambda (**input-parameters**) => { **<sequence-of-statements>** } compararemos si existe en la lista un usuario cuyo Documento y Contraseña coincida con el contenido de los textbox txtdocumento y txtclave y pidiendo que solo nos devuelva el primero o si no encuentra que devuelva el valor nulo (default), almacenando en ousuario (<https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/operators/lambda-expressions>)

Por último cuando cerremos el formulario de inicio borraremos el contenido de Usuario y Clave introducido.

```

33 private void btnInicio_Click(object sender, EventArgs e)
34 {
35     List<Usuario> TEST = new CD_Usuario().Listar();
36
37     Usuario ousuario = new CN_Usuario().Listar().Where(u => u.Documento == txtUsuario.Text && u.Clave == txtClave.Text).FirstOrDefault();
38
39     if (ousuario != null)
40     {
41         Inicio form = new Inicio();
42         form.Show();
43         this.Hide();
44
45         form.FormClosing += frm_closing;
46         //borrar contenido textbox usuario y clave
47         txtUsuario.Text = "";
48         txtClave.Text = "";
49     }
50     else
51     {
52         MessageBox.Show("usuario erróneo o no existente", "Mensaje", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
53     }
54 }

```