

Universidad de Costa Rica

**Escuela de Ciencias de la Computación e
Informática**

Bases de Datos I

**Tutorial básico de interfaces
C# Windows Forms**

II Ciclo 2016

Elaborado por: Gaudy Blanco

Contenido

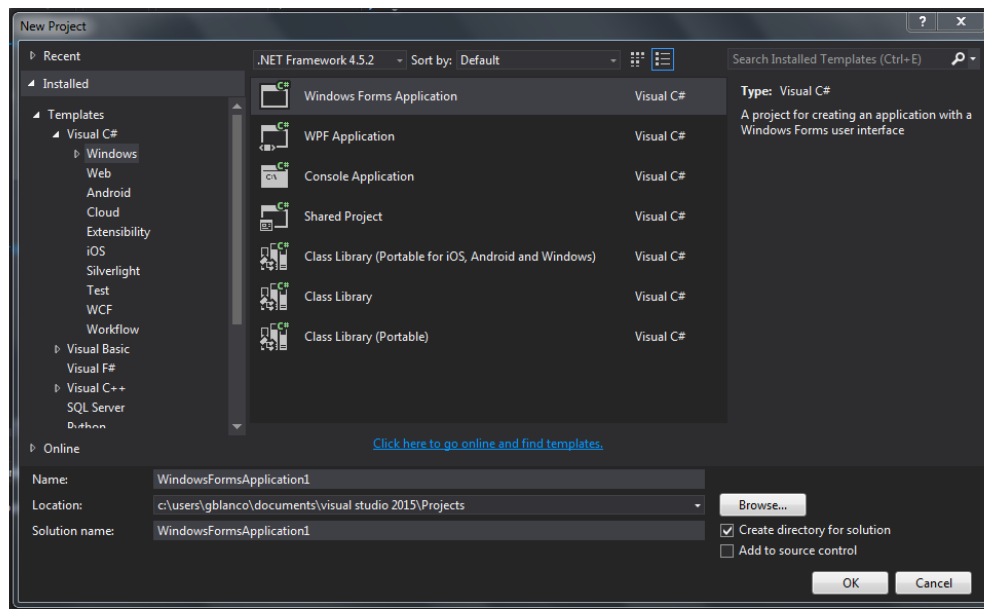
Introducción	3
1. Creación de un nuevo proyecto en Visual Studio 2013.....	3
2. Paneles básicos de Visual Studio 2013	4
3. Creación de interfaces.....	5
3a. Creación de la pantalla AgregarEstudiante	5
3b. Creación de la pantalla ListaEstudiantes	7
3c. Creación de la pantalla eliminarEstudiante	9
3d. Creación de la pantalla Login	9
4. Conexión a la base de datos de SQL Server.....	10
5. Creación de la tabla Usuarios para autenticación	11
6. Añadir procedimientos almacenados en su base de datos	11
7. Creación de una clase intermedia de comunicación entre la interfaz y la base de datos.....	15
8. Añadir código a la pantalla AgregarEstudiante	20
9. Añadir código a la pantalla ListaEstudiantes	24
10. Añadir código a la pantalla EliminarEstudiante.....	30
11. Añadir código en la pantalla Login	33
12. Ejecución de la aplicación	34

Introducción

El entorno de desarrollo integrado Visual Studio 2013 nos permite diseñar aplicaciones en distintos lenguajes de programación para distintas tecnologías (sitios web, aplicaciones de escritorio, aplicaciones de consola, etc...). Para el proyecto del curso se estará utilizando como lenguaje de programación C# y como tipo de aplicación Windows Forms Applications. A continuación se detallan todos los pasos necesarios para el diseño de una primera aplicación básica que permita insertar, consultar y eliminar registros de una base de datos. Además, esta guía también cubre de forma básica el uso de los controles más comunes de Visual Studio así como sus propiedades, el manejo de clases intermedias para una mejor separación del código y la interacción entre la base de datos y las interfaces gráficas.

1. Creación de un nuevo proyecto en Visual Studio 2013

Al acceder a Visual Studio 2013 nos dirigimos al menú File -> New -> Project, esta acción abrirá la siguiente pantalla:

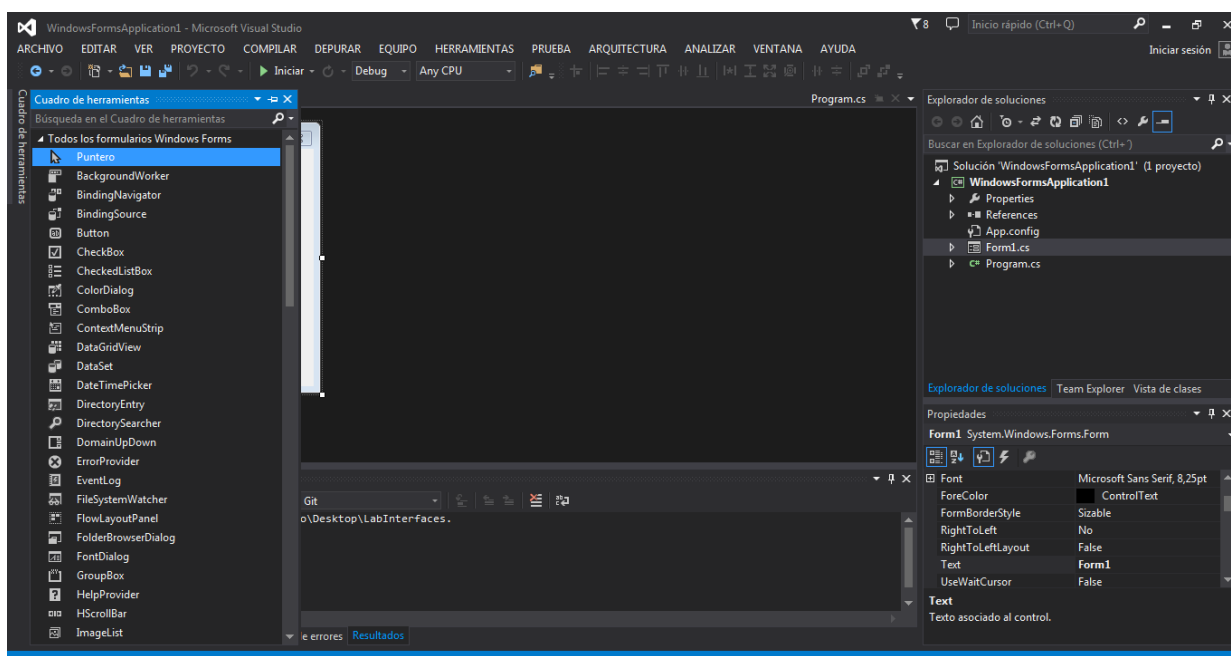


En esta pantalla se debe comprobar que en la parte izquierda este seleccionado Visual C# y la opción dentro de esta que corresponde a Windows. Al tener seleccionado esto en el centro de la pantalla se desplegará la opción Windows Forms Application, con la cual

trabajaremos a lo largo de la guía. Por último, podemos darle un nombre al proyecto así como una ubicación en la computadora.

2. Paneles básicos de Visual Studio 2013

Al crear un proyecto en Visual podemos acceder al entorno general de herramientas que nos provee (figura 2). Normalmente del lado izquierdo podemos encontrar los distintos controles (cajas de texto, labels, botones, datetimepickers, checkbox, etc...) que podemos añadir a un form (interfaz gráfica) de la aplicación. En el lado derecho en la parte superior se ubica el explorador de la solución de Visual (conjunto de archivos del proyecto con su jerarquía) y en la parte inferior el panel de propiedades que cargará propiedades diferentes dependiendo de qué control tengamos seleccionado en el form. Además, con el menú de la parte superior de la pantalla podemos acceder a distintos paneles que pueden ser necesarios en algún momento pero no para esta guía.



3. Creación de interfaces

Para este laboratorio vamos a crear una aplicación que permite insertar, consultar y eliminar registros de la tabla Estudiante, que fue creada en la base de datos de cada uno de los estudiantes en el laboratorio 1.

3a. Creación de la pantalla AgregarEstudiante

Primero, vamos a diseñar la pantalla para insertar estudiantes en dicha tabla. Utilizando el cuadro de herramientas de Visual, desplegado en el lado izquierdo de la pantalla, se debe diseñar una pantalla como la siguiente, sobre el form por defecto que carga el proyecto cuando se crea (se debe cambiar el nombre del form en la propiedad Name del panel de propiedades, para que se llame AgregarEstudiante):

The screenshot shows a Windows application window titled 'Agregar estudiante'. Inside, there's a form with the title 'Laboratorio #3'. At the top right, there are two links: 'Ir a Lista de estudiantes' and 'Eliminar estudiante'. The form is divided into two columns. The left column contains a group box labeled 'Datos del estudiante' (1), followed by labels and text boxes for 'Cédula:' (2), 'Carné:', 'Nombre:', 'Apellido 1:', 'Apellido 2:', 'Fecha nacimiento:' (with a date picker showing '22/08/2016' and a calendar icon) (5), and 'Dirección:'. The right column contains labels and text boxes for 'Teléfono:', 'Email:', 'Sexo:' (with radio buttons for 'Femenino', 'Masculino' (4), and 'Otro'), 'Usuario:', and 'Contraseña:'. At the bottom right, there is a 'Guardar' button (6). A red number 7 is placed near the top right links.

Los nombres de los controles de la pantalla anterior son:

1. Groupbox (Es un contenedor o panel que permite agrupar elementos, como en este caso, que agrupa todos los datos del estudiante).
2. Label (Permite mostrar texto en pantalla).
3. Textbox (Permite recibir texto del usuario).

4. Radiobutton (Permite chequear valores en la interfaz, este control permite marcar solo un radiobutton a la vez. **Para los que implementaron el sexo en la tabla como un booleano solo añadan 2 radiobutton en lugar de 3**).
5. Datetimepicker (Control para seleccionar fechas).
6. Button
7. LinkLabel (Realiza una acción semejante al button, ya que ambos pueden ser presionados y generar una acción).

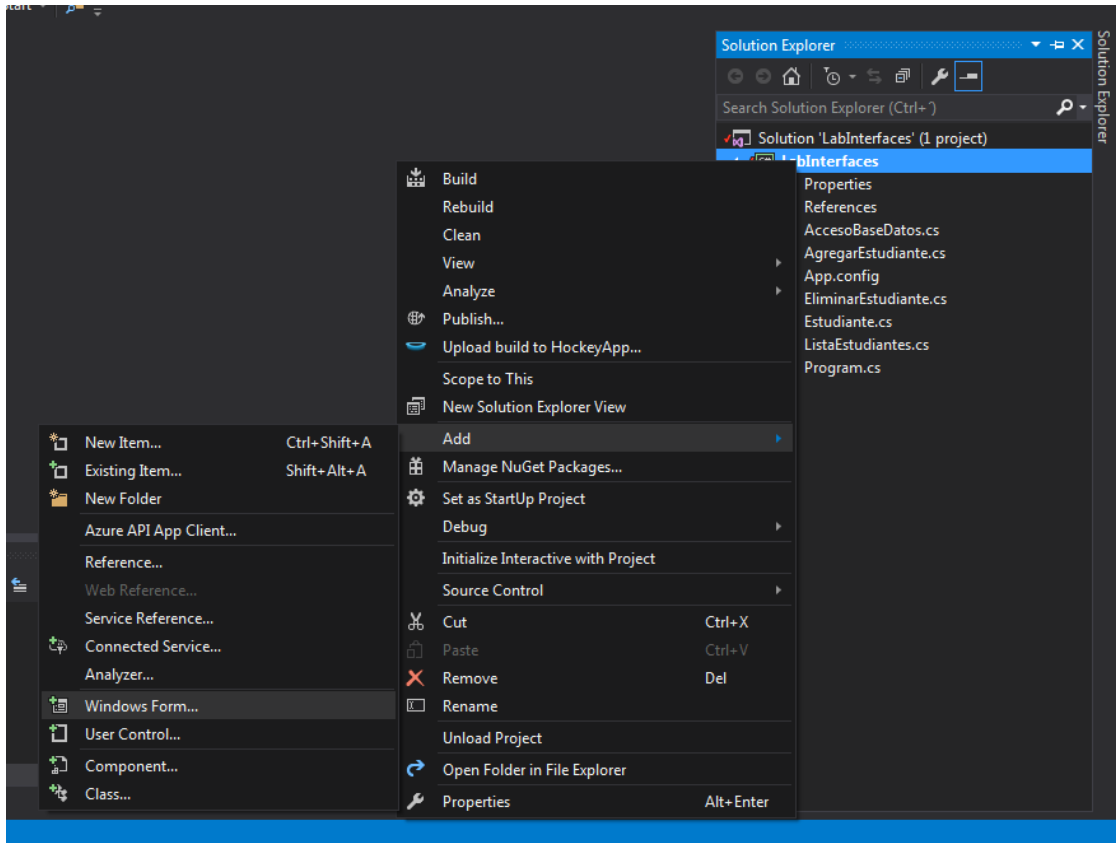
Todos estos controles tienen, cuando están seleccionados, un conjunto de propiedades que se carga en el panel de la parte inferior derecha de Visual Studio. Entre las propiedades que son importantes que conozcan se encuentran:

1. **Name:** permite darle al control un nombre con el que se puede acceder a él desde el código de la aplicación, por ejemplo: txtCedula para un textbox que recibe un número de cédula, lblCedula para un label, dtpFecha para un datetimepicker, nótese que se trata de identificar los controles con un prefijo en el nombre.
2. **Text:** En el caso de los labels o linklabels permite asociar el texto que este control va a tener cargado en la interfaz.
3. **PasswordChar:** Al asignarle un carácter a esta propiedad de los textbox podemos ocultar lo que se escribe en dicho textbox (útil para contraseñas).

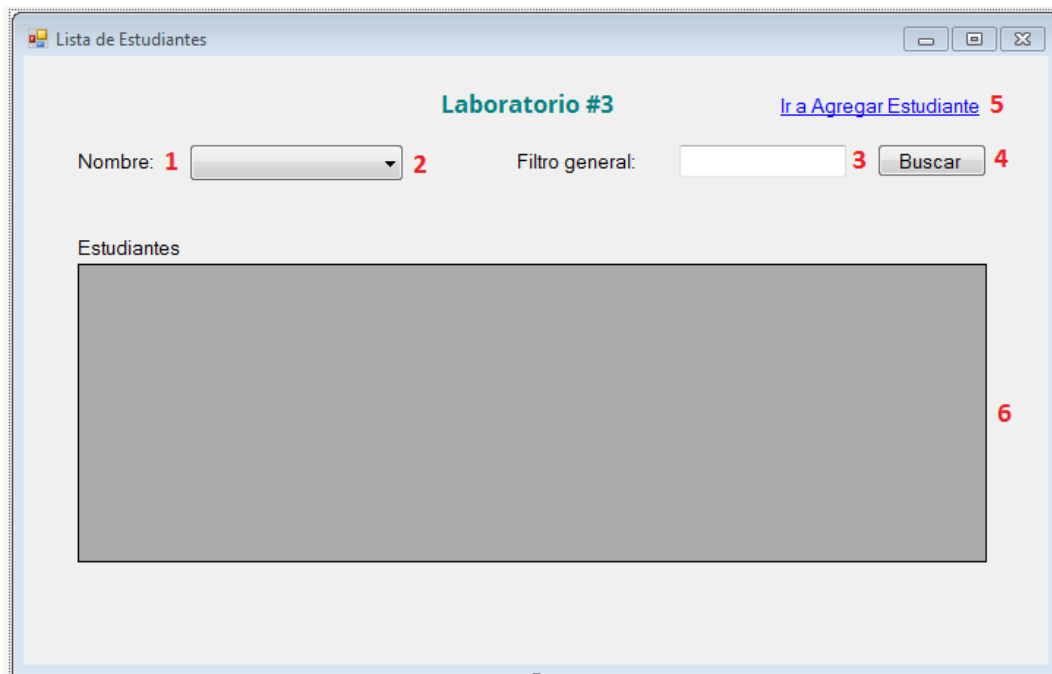
Hints: Algunas propiedades interesantes que pueden investigar para utilizar en sus proyectos, pero que no son necesarias para este laboratorio son Font, Forecolor, Enabled, MaxLength, Multiline, ReadOnly, PasswordChar, TabIndex, Backcolor, StartPosition, entre otras.

3b. Creación de la pantalla ListaEstudiantes

Ahora vamos a diseñar una pantalla que nos permita consultar estudiantes ya sea mediante su nombre o mediante un filtro de búsqueda introducido por el usuario. Para esto se debe crear un nuevo form de la siguiente forma:



En la ventana que se despliega deben ponerle un nombre al form, en este caso ListaEstudiantes. Ya con el form creado deben diseñar una pantalla como la siguiente:



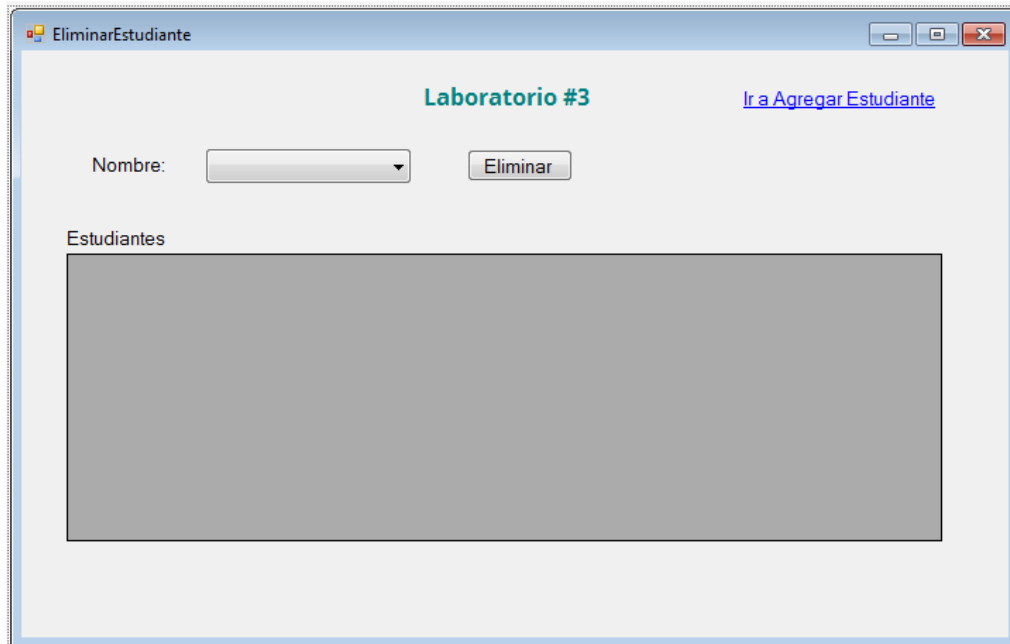
Los nombres de los controles de la pantalla anterior son:

1. Label
2. Combobox (Para que sea de solo lectura su propiedad DropDownStyle del panel de propiedades debe estar en DropDownList).
3. Textbox.
4. Button.
5. LinkLabel.
6. DataGridView (Control que permite desplegar conjuntos de tuplas de la base de datos).

Hint: Recuerden colocar nombres a los controles para poder acceder a ellos más fácilmente más adelante.

3c. Creación de la pantalla eliminarEstudiante

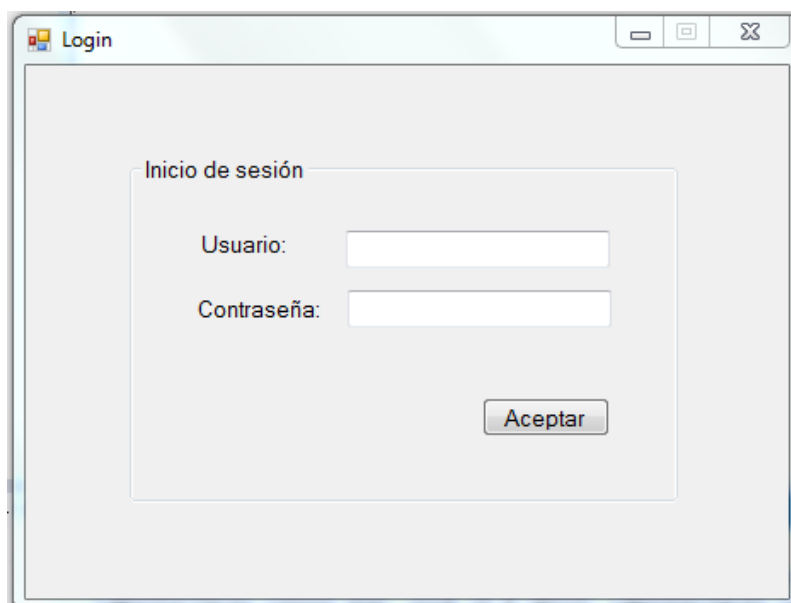
Ahora crearemos una pantalla que nos permite eliminar estudiantes de la base de datos de acuerdo a su nombre. Para esto crearemos una pantalla como la siguiente:



The screenshot shows a window titled "EliminarEstudiante". At the top center, it says "Laboratorio #3". To the right of this, there is a blue hyperlink that says "Ir a Agregar Estudiante". Below this, there is a label "Nombre:" followed by a dropdown menu and a button labeled "Eliminar". Underneath, there is a label "Estudiantes" followed by a large, empty rectangular box, likely intended for a list of students.

3d. Creación de la pantalla Login

Por último, crearemos una pantalla que nos permita autenticar usuarios en el sistema con los datos de la cuenta. Para esto crearemos la siguiente pantalla:



The screenshot shows a window titled "Login". Inside the window, there is a section titled "Inicio de sesión". Below this title, there are two labels: "Usuario:" and "Contraseña:", each followed by a text input field. At the bottom right of the input fields, there is a button labeled "Aceptar".

4. Conexión a la base de datos de SQL Server

Ahora se debe añadir una clase de conexión a la base de datos que se encuentra entre los recursos de Mediación Virtual. La clase se llama "AccesoBaseDatos.cs" y permite la interacción con la base de datos. Esta se debe colocar en la carpeta del proyecto de Visual, donde se encuentran ubicados los demás archivos. Luego de copiarla en esta ubicación se debe dar click derecho sobre el proyecto en el Explorador de Soluciones (lado derecho de Visual Studio), seleccionar Add -> Existing Item y añadir el archivo previamente copiado.

En esta clase lo único que se debe modificar es el campo Initial Catalog, ya que deben colocar el nombre de su base de datos BD_xxxxxx y probablemente el nombre del namespace para que se llame igual que en su proyecto. Esta clase provee métodos para consultar tuplas de la tabla Estudiante, un método genérico que permite ejecutar inserts, update y delete, y por último, un método para utilizar un procedimiento almacenado que borre estudiantes.

```
/*En Initial Catalog se agrega la base de datos propia. Integrated Security es para utilizar Windows Authentication*/  
    String conexion = "Data Source=10.1.4.55; Initial Catalog=gaudyblanco;  
Integrated Security=SSPI";
```

Los campos de dicho string de conexión son Data Source (nombre o IP del servidor de SQL Server), Initial Catalog (nombre de la base con la cual se desea conectar) y Integrated Security (modo de seguridad para la conexión, SSPI significa que es Windows Authentication).

En caso que se desee utilizar la base de datos accediendo a esta mediante una VPN debemos realizar la conexión utilizando SQL Server Authentication. El string de conexión en este caso varía un poco:

```
/*En Initial Catalog se agrega la base de datos propia. Integrated Security = false es para utilizar SQL SERVER Authentication*/  
    string conexion = "Data Source=10.1.4.55;User ID=Gaudy;Password=xxxxx;  
Initial Catalog=gaudyblanco; Integrated Security=false";
```

A este string le añadimos los campos User ID (usuario de SQL Server Authentication), Password (colocan aquí su contraseña de SQL Server Authentication) e Integrated Security (se cambia el valor a false, para que utilice SQL Server Authentication).

5. Creación de la tabla Usuarios para autenticación

Como podrán haber notado en la interfaz de AgregarEstudiante, existen dos campos que no tienen en su tabla Estudiante, los cuales corresponden a usuario y contraseña. Esto se hizo con la intención de que aprendan a realizar autenticación de usuarios en una aplicación. El primer paso para realizarlo es crear la siguiente tabla en su base de datos:

```
CREATE TABLE dbo.[Usuarios]
(
    cedulaUsuario varchar(9) NOT NULL,
    nombreUsuario NVARCHAR(40) NOT NULL PRIMARY KEY,
    PasswordHash BINARY(64) NOT NULL,
    salt UNIQUEIDENTIFIER,
    FOREIGN KEY (cedulaUsuario) REFERENCES Estudiante (cedula)
)
```

En esta tabla, **el atributo cedulaUsuario debe ser del mismo tipo y tamaño que la cédula del estudiante en su tabla Estudiante** ya que, como pueden ver, necesitamos que esta corresponda a una llave foránea de la tabla Estudiante. También necesitamos un atributo nombreUsuario para guardar el usuario asociado al estudiante, un atributo PasswordHash para almacenar la contraseña encriptada y un atributo salt para guardar uno de los atributos de encriptación del password.

6. Añadir procedimientos almacenados en su base de datos

En su base de datos BD_XXXXXX vamos a añadir diferentes procedimientos almacenados para funciones de la aplicación. El primer procedimiento almacenado servirá para eliminar estudiantes de la base de datos:

```

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE eliminarEstudiante @nombre varchar(20)

AS
select distinct Estudiante.Cedula
into #temp
from Estudiante join Usuarios on Estudiante.Cedula = Usuarios.cedulaUsuario
WHERE Estudiante.nombre = @nombre

delete Usuarios
where cedulaUsuario in (select * from #temp)

delete Estudiante
where Cedula in (select * from #temp)

drop table #temp
GO

```

Este procedimiento lo que nos permite es que al elegir un nombre de estudiante que deseemos eliminar, poder eliminar el usuario asociado a esos estudiantes (en caso que se seleccione un nombre que corresponda a más de un estudiante en la base de datos). También vamos a necesitar dos procedimientos almacenados para la autenticación de usuarios:

```

CREATE PROCEDURE dbo.agregarUsuario
/*Parámetros: pLogin donde se recibe el nombre de usuario,
pPassword donde se recibe la contraseña,
cedula donde se recibe la cédula del estudiante asociado a dicho usuario
y se declara el parámetro de salida estado que devuelve 1 si el usuario
se pudo guardar en la base de datos y cualquier otro número que corresponde
al ERROR_MESSAGE() si no se pudo guardar*/
    @pLogin NVARCHAR(50),
    @pPassword NVARCHAR(50),
    @cedula varchar(9),
    @estado bit OUTPUT
AS
BEGIN
    SET NOCOUNT ON

    /*Se genera un salt, el cual corresponde a una llave de encriptación del
password*/
    DECLARE @salt UNIQUEIDENTIFIER=NEWID()
    BEGIN TRY

        /*Se inserta en la tabla Usuarios los datos de un nuevo usuario, se
encripta la contraseña
        con un HASHBYTES con el algoritmo SHA2_512 con la unión del password
digitado y el salt (notese que
        este salt es único para cada usuario sin importar que tengan la misma
contraseña, este se almacena
        diferente para cada uno)*/
        INSERT INTO dbo.[Usuarios] (cedulaUsuario, nombreUsuario, PasswordHash,
Salt)
        VALUES(@cedula, @pLogin, HASHBYTES('SHA2_512', @pPassword+CAST(@salt AS
NVARCHAR(36))), @salt)

        /*si la inserción se pudo realizar se devuelve un 1*/
        SET @estado=1

    END TRY
    BEGIN CATCH
        /*En cualquier otro caso se devuelve el mensaje de error*/
        SET @estado=ERROR_MESSAGE()
    END CATCH
END

```

Este procedimiento agregarUsuario recibe como parámetro una cédula, un nombre de usuario y un password, y devuelve un booleano con un 1 si se pudo agregar el usuario y otro número distinto si la operación no se pudo realizar. En términos generales, el procedimiento se encarga de guardar la contraseña encriptada mediante la función HASHBYTES utilizando el algoritmo SHA2_512 con un casteo del password digitado junto con el salt generado para ese usuario en específico.

Ahora el procedimiento almacenado para el login es el siguiente:

```
CREATE PROCEDURE dbo.Login
/*Parámetros: pLoginName donde se recibe el nombre de usuario,
pPassword donde se recibe la contraseña,
y se declara el parámetro de salida isInDB que devuelve 1 si el usuario
si está en la BD o 0 si no está*/
    @pLoginName NVARCHAR(254),
    @pPassword NVARCHAR(50),
    @isInDB bit=0 OUTPUT
AS
BEGIN

    SET NOCOUNT ON

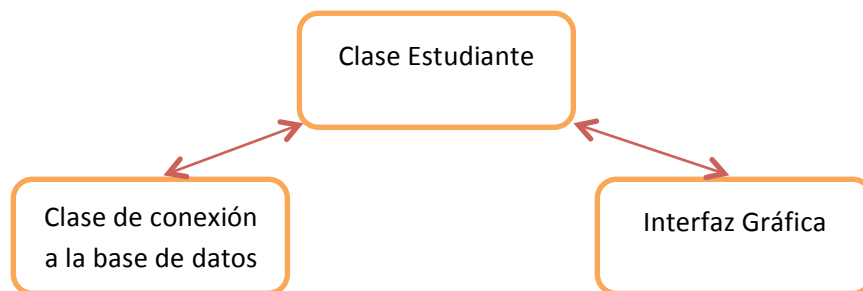
    /*Se declara variable para buscar el usuario*/
    DECLARE @userID INT

    /*Se pregunta si existe una cedulaUsuario que tenga en nombreUsuario lo reci-
bido en pLoginName*/
    IF EXISTS (SELECT TOP 1 cedulaUsuario FROM [dbo].[Usuarios] WHERE nombreUsu-
ario=@pLoginName)
    BEGIN
        /*Si si existe una cédula con este nombreUsuario se pregunta si el
Password de dicho usuario
corresponde al recibido por parámetro junto con el salt de esa tupla*/
        SET @userID=(SELECT cedulaUsuario FROM [dbo].[Usuarios] WHERE nombreUsu-
ario=@pLoginName AND PasswordHash=HASHBYTES('SHA2_512', @pPassword+CAST(Salt AS
NVARCHAR(36))))

        /*si al final de ambas consultas userID es null se retorna 0*/
        IF(@userID IS NULL)
            SET @isInDB=0
        /*Si al final de ambas consultas userID no es null se retorna 1*/
        ELSE
            SET @isInDB=1
        END
        /*Si no existe ninguna cédula asociada a ese nombre de usuario se retorna 0*/
        ELSE
            SET @isInDB=0
    END
END
```

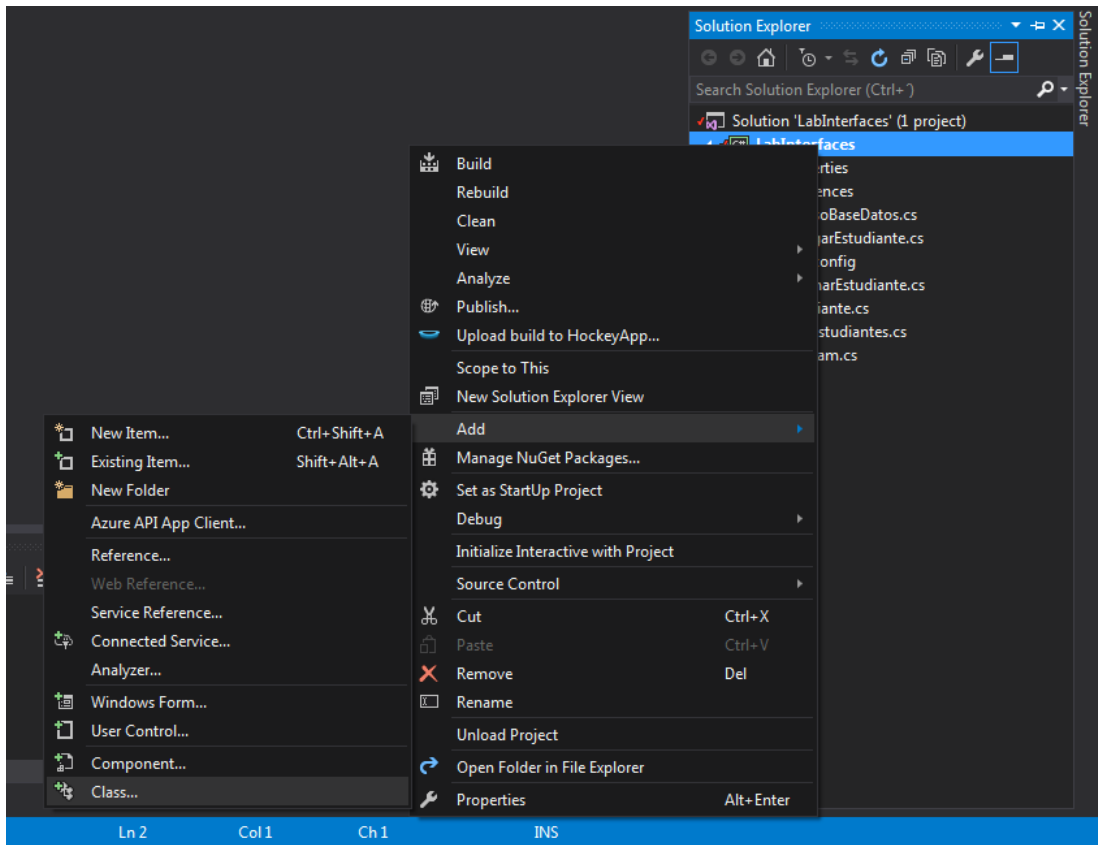
7. Creación de una clase intermedia de comunicación entre la interfaz y la base de datos

Una buena práctica de programación es el no sobrecargar de responsabilidades a las clases más allá de las tareas que originalmente les toca administrar. Comúnmente las clases que se encargan de comunicarse con el usuario (en este caso los forms) solo deben preocuparse por recibir las solicitudes del usuario y dar respuesta al usuario después de procesar la solicitud. Por esto motivo no es correcto que las interfaces tengan comunicación directa con las clases encargadas de la conexión con la base de datos, además que es algo inseguro. Existen muchos patrones y estrategias para resolver dicho problema, pero para mayor facilidad en este laboratorio y en el proyecto del curso manejaremos una estructura de clases como la siguiente:



En este diagrama podemos ver que por un lado está la interfaz gráfica, la cual se comunica con el usuario. Esta interfaz se comunica únicamente con la clase Estudiante, la cual se encarga de crear las sentencias SQL para manipular los datos de la tabla Estudiante. Por último, la clase Estudiante se encarga de la comunicación con la clase de conexión a la base de datos para que se ejecuten las sentencias SQL que creó en sus métodos.

Para esto vamos a crear la clase Estudiante en el proyecto de Visual Studio, dando click derecho en el proyecto, en el menú que se despliega seleccionamos Add -> Class y en la ventana emergente colocamos como nombre de la clase Estudiante.cs:



En esta clase agregaremos, en la parte superior, 2 directivas using para poder utilizar objetos SQL:

```
using System.Data;
using System.Data.SqlClient;
```

También debemos añadir un objeto tipo AccesoBaseDatos para acceder a los métodos de la clase de conexión a la base de datos y debemos inicializarlo en el constructor de la clase:

```
AccesoBaseDatos bd;

/*Constructor de la clase estudiante*/
public Estudiante()
{
    //Se inicializa el objeto que realiza la conexión con la base de datos
    bd = new AccesoBaseDatos();
}
```


Ahora añadiremos un método para agregar estudiantes a la base de datos, **aquí su código puede variar de acuerdo al nombre de los atributos de la tabla Estudiante que crearon en su base de datos:**

```
/*Método para agregar un nuevo estudiante a la base de datos
   Recibe: Los datos del nuevo estudiante
   Modifica: inserta en la base de datos el nuevo estudiante
   Retorna: el tipo de error que generó la inserción o cero si la inserción
   fue exitosa*/
public int agregarEstudiante(string cedula, string carne, string nombre,
string ape1, string ape2, string email, char genero, string fechaNac, string
direccion, string telefono, int estado)
{
    String insertar = "INSERT INTO Estudiante (Cedula, carne, nombre,
apellido1, apellido2, email, sexo, fechaNac, direccion, telefono, estado ) VALUES ("
+ cedula + "','" + carne + "','" + nombre + "','" + ape1 + "','" + ape2 + "','" +
email + "','" + genero + "','" + fechaNac + "','" + direccion + "','" + telefono +
"', '" + 1 + "' )";
    return bd.actualizarDatos(insertar);
}
```

También agregaremos un método para obtener una lista con los nombres de todos los estudiantes de la base de datos:

```
/*Método para obtener los nombres de estudiantes de la base de datos
   Recibe: Nada
   Modifica: Realiza la selección de los nombres de estudiantes y lo carga en
un dataReader
   Retorna: el dataReader con los datos*/
public SqlDataReader obtenerListaNombresEstudiantes()
{
    SqlDataReader datos = null;
    try
    {
        datos = bd.ejecutarConsulta("Select distinct nombre from
Estudiante");
    }
    catch (SQLException ex)
    {
    }

    return datos;
}
```

Después añadiremos un método para obtener una lista con todos los datos de los estudiantes de la base de datos y, de acuerdo a los valores que reciba por parámetro, también se pueden realizar filtros en las tuplas de la base de datos:

```

/*Método para obtener los estudiantes de la base de datos
    Recibe: dos tipos de filtros por los cuales se pueden filtrar las tuplas
    Modifica: Realiza la selección de los estudiantes y los carga en un
dataTable
    Retorna: el dataTable con los datos*/
public DataTable obtenerEstudiantes(string filtroNombre, string
filtroGeneral)
{
    DataTable tabla = null;
    try
    {
        //Si los filtros son nulos se cargan todos los estudiantes de la
base de datos
        if (filtroGeneral == null && filtroNombre == null)
        {
            tabla = bd.ejecutarConsultaTabla("Select * from estudiante");
        }
        //Si el filtro de nombre no es nulo carga los estudiantes cuyo
nombre sea el que tiene el filtro
        else if(filtroNombre != null)
        {
            tabla = bd.ejecutarConsultaTabla("Select * from estudiante where
nombre ='" + filtroNombre+ "'");
        }
        //Si el filtro general no es nulo cargan los estudiantes con
atributos que contengan ese filtro como parte del atributo (like)
        else if(filtroGeneral != null)
        {
            tabla = bd.ejecutarConsultaTabla("Select * from estudiante where
nombre like '%" + filtroGeneral + "%' OR apellido1 like '%" + filtroGeneral + "%' OR
apellido2 like '%" + filtroGeneral + "%' OR cedula like '%" + filtroGeneral + "%' OR
carne like '%" + filtroGeneral + "%'");
        }
        //Si ninguno de los filtros es nulo carga los estudiantes que
coincidan con ambos filtros
        else if(filtroGeneral != null && filtroNombre != null)
        {
            tabla = bd.ejecutarConsultaTabla("Select * from estudiante where
nombre ='" + filtroNombre + "' && nombre like '%" + filtroGeneral + "%' OR
apellido1 like '%" + filtroGeneral + "%' OR apellido2 like '%" + filtroGeneral + "%'
OR cedula like '%" + filtroGeneral + "%' OR carne like '%" + filtroGeneral + "%'");
        }

    }
    catch (SQLException ex)
    {
    }

    return tabla;
}

```

También, añadiremos un método para poder invocar al procedimiento almacenado para eliminar estudiantes por nombre que ya se cargó en la base de datos anteriormente:

```

/*Método para eliminar un estudiante mediante el procedimiento almacenado
  Recibe: El nombre de los estudiantes o estudiante a eliminar
  Modifica: Llama al método que elimina el estudiante mediante el nombre
  Retorna: el tipo de error que generó el eliminar o cero si el eliminar fue
exitoso*/
public int eliminarEstudiante(string nombre)
{
    return bd.eliminarEstudiante(nombre);
}

```

Además, debemos añadir un método para poder llamar al procedimiento almacenado que nos permite agregar un nuevo usuario:

```

/*Método para agregar un usuario mediante el procedimiento almacenado
  Recibe: el nombre de usuario, contraseña y cédula del estudiante al cual
queremos crearle el nuevo usuario
  Modifica: llama al método agregarUsuario de la base de datos para agregar
el nuevo usuario
  Retorna: si se pudo agregar el nuevo usuario devuelve true, sino false*/
public bool agregarUsuario(string nombre, string password, string cedula)
{
    int resultado = bd.agregarUsuario(nombre, password, cedula);

    //si el procedimiento almacenado devuelve un 1 es porque agregó un nuevo
usuario, por lo que se convierte en true
    if (resultado == 1)
    {
        return true;
    }

    //Cualquier valor distinto a 1 que el procedimiento almacenado
significa que no agregó un nuevo usuario
    else
    {
        return false;
    }
}

```

Por último, añadimos un método para comprobar si un usuario y contraseña se encuentran en la base de datos:

```

        /*Método para verificar que dado un usuario y una contraseña el usuario si
se encuentra en la base de datos
    Recibe: el nombre y la contraseña que queremos verificar que existe en la
base de datos
    Modifica: obtiene si ese usuario y contraseña están en la bd
    Retorna: true si se encuentra en la bd, false si no lo encuentra */
public bool login(string nombre, string password)
{
    return bd.login(nombre, password);
}

```

8. Añadir código a la pantalla AgregarEstudiante

Todo form de Visual tiene también un archivo de código asociado a la parte gráfica. Podemos acceder a este al dar click derecho sobre el form agregarEstudiante y seleccionar Ver código. Aquí debemos añadir toda la funcionalidad detrás de los controles, por lo que lo primero que vamos a añadir es un objeto de tipo Estudiante para poder utilizar los métodos implementados en esta clase anteriormente.

Para esto añadimos el siguiente código, el cual declara un objeto de tipo Estudiante y lo inicializa en el constructor de la interfaz junto con los componentes gráficos (el constructor ya se encuentra generado en el código):

```

Estudiante estudiante;

/*Constructor de la clase*/
public AgregarEstudiante()
{
    InitializeComponent();
    estudiante = new Estudiante();
}

```

Ahora vamos a añadir la acción del botón Guardar. Para esto damos doble click sobre el botón en el form y esto automáticamente nos genera el evento del click en el código:

```

/*Método que se activa al dar click en el botón guardar para guardar un nuevo
estudiante en la base de datos*/
private void btnGuardar_Click(object sender, EventArgs e)
{
}

```

En este botón se realizarán todas las acciones para insertar un nuevo estudiante en la base de datos. Primero, debemos obtener que género seleccionó el usuario (**esto varía si**

en la base de datos declararon este campo como un booleano, ya que la variable genero sería de este tipo de dato y no char como en el ejemplo). Con la propiedad **Checked** del radiobutton podemos obtener si este es el radiobutton marcado por el usuario:

```
/*Método que se activa al dar click en el botón guardar para guardar un nuevo
estudiante en la base de datos*/
private void btnGuardar_Click(object sender, EventArgs e)
{
    //obtengo el genero seleccionado en la pantalla
    char genero = ' ';

    //El control radiobutton tiene la propiedad Checked en true si este se
    encuentra seleccionado
    if(rbFem.Checked)
    {
        genero = 'F';
    }
    else if(rbMasc.Checked)
    {
        genero = 'M';
    }
    else if (rbOtro.Checked)
    {
        genero = 'O';
    }
}
```

Después de las sentencias del if llamamos al método para agregar un nuevo estudiante que nos provee la clase Estudiante. Como pueden ver en el siguiente código, es importante el uso de nombres significativos para componentes como los textboxes, ya que podemos acceder a lo que el usuario digitó en ellos mediante la sentencia **nombreControl.text**.

```
/*Mediante el objeto de tipo Estudiante podemos agregar un nuevo estudiante con el
método agregarEstudiante, el cual recibe por parámetro todos los valores de la tabla
Estudiante*/
int resultado = estudiante.agregarEstudiante(txtCedula.Text,
txtCarne.Text, txtNombre.Text, txtApe1.Text, txtApe2.Text, txtEmail.Text, genero,
dtpFecha.Value.ToString("yyyy-MM-dd"), txtDireccion.Text ,txtTelefono.Text , 1);
```

Como el método agregarEstudiante de la clase Estudiante devuelve un entero que corresponde a un número de error SQL, en caso que la inserción fue errónea, o un valor cero en caso que la inserción fue exitosa, podemos agregar el nuevo usuario en la tabla Usuarios ya que comprobamos que se guardó el estudiante. Para esto añadimos el siguiente código justo después del llamado al método agregarEstudiante:

```

        //Si la inserción devuelve un 0 la inserción fue exitosa, por lo que se
        trata de insertar el usuario
        if(resultado == 0)
        {
            //se inserta el usuario mediante el llamado al método agregarUsuario
            en la clase Estudiante
            bool resultado1 = estudiante.agregarUsuario(txtUsuario.Text, txt-
            Password.Text, txtCedula.Text);

            //si se agregó el nuevo usuario se muestra un mensaje de éxito
            if (resultado1 == true)
            {
                MessageBox.Show("El estudiante ha sido agregado exitosamente",
                "Resultados", MessageBoxButtons.OK, MessageBoxIcon.None);
                //Se limpian las cajas de texto para permitir al usuario añadir
                un nuevo estudiante cuando lo desee
                txtCarne.Clear();
                txtCedula.Clear();
                txtNombre.Clear();
                txtApe1.Clear();
                txtApe2.Clear();
                txtEmail.Clear();
                txtDireccion.Clear();
                txtTelefono.Clear();
                txtUsuario.Clear();
                txtPassword.Clear();
            }
            else
            {
                /*Aquí se podría validar con distintos mensajes de error de
                acuerdo al número de error recibido
                y además, si no se pudo agregar el usuario para el estudiante
                lo ideal sería que se eliminara
                el estudiante que se acaba de crear, ya que hay una inconsis-
                tencia entre que si se guardara una
                tupla en Estudiante pero en Usuarios no*/
                MessageBox.Show("Ya existe un estudiante asociado a este usuario
                en el sistema", "Resultados", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        //si la inserción devuelve un código de error se puede validar con un
        mensaje de error personalizado
        else if (resultado == 2627)
        {
            MessageBox.Show("Ya existe un estudiante asociado a este numero de
            cedula en el sistema", "Resultados", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

```

Por último, en esta clase debemos agregar los métodos para el click sobre los linkLabels que añadimos en esta pantalla anteriormente. Estos linkLabels nos permiten trasladarlos hacia las otras pantallas que diseñamos. Para esto, igual que con el botón, debemos dar

click sobre cada uno de los linklabels para generar la acción del click en el código. En la acción del linkLabel “Ir a lista de Estudiantes” se añade el siguiente código:

```
//Creación de la interfaz ListaEstudiantes y se muestra, desaparece la
interfaz actual
ListaEstudiantes lista = new ListaEstudiantes();
lista.Show();
this.Hide();
```

Este método debe verse como el siguiente código:

```
/*Método que se activa cuando se da click en el link de Lista de
estudiantes*/
private void lkLista_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    //Creación de la interfaz ListaEstudiantes y se muestra, desaparece la
    interfaz actual
    ListaEstudiantes lista = new ListaEstudiantes();
    lista.Show();
    this.Hide();
}
```

Ahora, en la acción del linkLabel “Eliminar estudiante” se debe añadir el siguiente código:

```
//Creación de la interfaz EliminarEstudiante y se muestra, desaparece la interfaz
actual
EliminarEstudiante eliminar = new EliminarEstudiante();
eliminar.Show();
this.Hide();
```

Este método debe verse como el siguiente código:

```
/*Método que se activa cuando se da click en el link de eliminar Estudiante*/
private void lkEliminar_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    //Creación de la interfaz EliminarEstudiante y se muestra, desaparece la
    interfaz actual
    EliminarEstudiante eliminar = new EliminarEstudiante();
    eliminar.Show();
    this.Hide();
}
```

9. Añadir código a la pantalla ListaEstudiantes

De la misma forma que con la pantalla AgregarEstudiante, debemos añadir un objeto de tipo Estudiante para acceder a sus métodos:

```
Estudiante estudiante;

/*Constructor de la clase*/
public ListaEstudiantes()
{
    InitializeComponent();
    estudiante = new Estudiante();
}
```

Ahora, vamos a añadir un método que permite cargar todos los nombres de estudiantes en el combobox de la interfaz. Para esto añadimos el siguiente código:

```
/*Método para llenar un combobox con datos específicos
   Recibe: Un objeto combobox que va a llenar con una consulta específica
   Modifica: Llena el combobox que recibe por parámetro con el nombre de todos
los estudiantes que se encuentran en la bd
   Retorna: Ningún valor*/
private void llenarCombobox(ComboBox combobox)
{
    //Se obtiene un dataReader con todos los nombres de los estudiantes de
la base de datos
    SqlDataReader datos = estudiante.obtenerListaNombresEstudiantes();

    /*Si existen datos en la base de datos se carga como primer elemento del
combobox un dato "Seleccione"
y luego se cargan todos los datos de la base de datos*/
    if (datos != null)
    {
        combobox.Items.Add("Seleccione");
        while (datos.Read())
        {
            combobox.Items.Add(datos.GetValue(0));
        }
    }
    /*Si no hay tuplas en la base de datos se limpia el combobox y se carga
unicamente el valor "Seleccione"*/
    else
    {
        combobox.Items.Clear();
        combobox.Items.Add("Seleccione");
    }

    //Se pone por defecto la primera entrada del combobox seleccionada
    combobox.SelectedIndex = 0;
}
```


También debemos declarar un método para llenar la tabla de datos (dataGridView) que añadimos a la interfaz. Este método es un poco más general, ya que nos permite añadir distintos filtros a la búsqueda de tuplas en la base de datos:

```
/*Método para llenar un datagridview con datos específicos
   Recibe: Un control datagridview que va a cargar con datos, una string en
   caso que se quiera filtrar por el valor
   * del combobox y un string de filtroGeneral en caso que se quiera filtrar
   por el texto introducido por el usuario
   Modifica: carga los datos en el datagridview
   Retorna: ningún valor*/
private void llenarTabla(DataGridView dataGridView, string filtroCombobox,
string filtroGeneral)
{
    /*obtiene un dataTable con todos los estudiantes que se encuentran en la
    base de datos que cumplan las condiciones
    de los dos filtros que el método recibe por parámetro*/
    DataTable tabla = estudiante.obtenerEstudiantes(filtroCombobox,
filtroGeneral);

    //Se inicializa el source para cargar el datagridview y se le asigna el
    dataTable obtenido
    BindingSource bindingSource = new BindingSource();
    bindingSource.DataSource = tabla;

    dataGridView.AutoSizeColumnsMode(DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)
    ;
    dataGridView.DataSource = bindingSource;

    //Ciclo para darle un ancho a cada columna del datagridview
    proporcionado
    for (int i = 0; i < dgvEstudiantes.ColumnCount; i++)
    {
        dataGridView.Columns[i].Width = 100;
    }
}
```

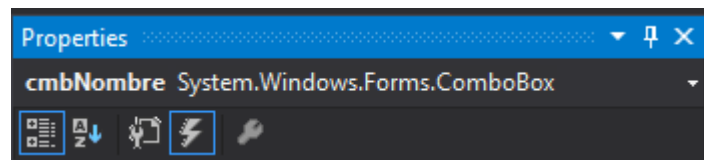
Ahora debemos determinar donde llamar a estos métodos para que la interfaz nos pueda mostrar los datos de los estudiantes en todo momento. En este tipo de pantallas de consulta normalmente queremos que en el momento que accedemos a ella tenga todas las tuplas cargadas en el dataGridView y todos los nombres cargados en el combobox. Para esto debemos solicitar este tipo de datos en el Load del form, este es el método de carga de la pantalla cuando accedemos a ella. Para acceder a él debemos dar doble click sobre el form (genera el código del método) y colocar el siguiente código dentro del método que acabamos de generar:

```
//Llena el combobox de nombres de estudiante
    llenarCombobox(cmbNombre);
    //Llena el datagridview de estudiantes con todas las tuplas de
estudiante de la interfaz
    llenarTabla(dgvEstudiantes, null, null);
```

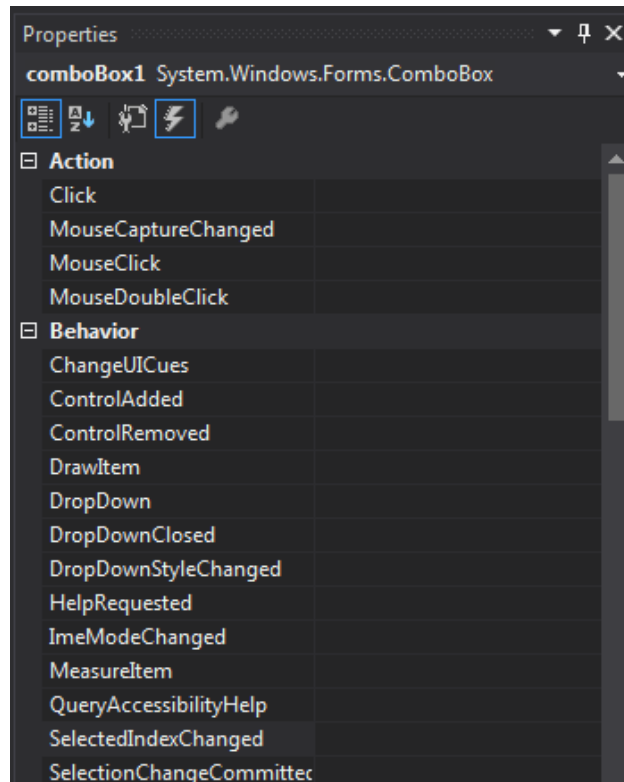
Este método del Load debería verse así después de añadir el código anterior:

```
/*Método load, este método es llamado cuando se carga la pantalla */
private void ListaEstudiantes_Load(object sender, EventArgs e)
{
    //Llena el combobox de nombres de estudiante
    llenarCombobox(cmbNombre);
    //Llena el datagridview de estudiantes con todas las tuplas de
estudiante de la interfaz
    llenarTabla(dgvEstudiantes, null, null);
}
```

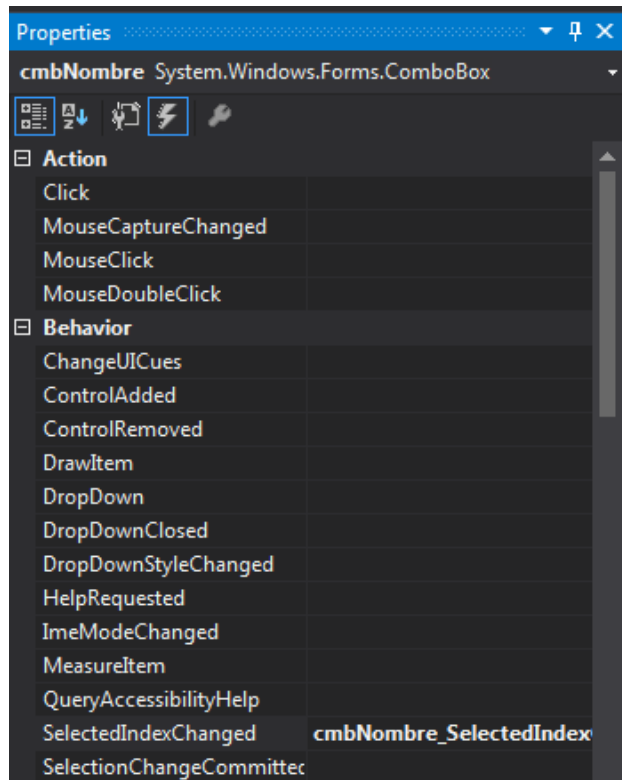
Por último, sería interesante que la pantalla pueda filtrar las tuplas del dataGridView mediante el nombre seleccionado en el combobox sin necesidad de dar click en el botón Buscar. Esto es posible ya que cada control tiene una serie de eventos específicos. Para acceder a los eventos disponibles para un control seleccionamos el control en el form y vamos al panel de propiedades de la parte inferior derecha de Visual. Este panel tiene un conjunto de iconos como el siguiente:



El icono de un rayo corresponde a los eventos disponibles para el control. En el caso del combobox se muestran eventos como los de la siguiente imagen:



Entre todos estos eventos necesitamos uno que realice una acción cuando la selección del combobox cambie, es decir, si el usuario selecciona un nombre del combobox que cargue en el dataGridView las tuplas cuyo nombre coincida con el seleccionado en el combobox. Si el usuario vuelve a seleccionar otro elemento del combobox debería cargar otras tuplas y así sucesivamente. Este tipo de acción se llama `SelectedIndexChanged` y si damos doble click sobre el evento en el panel nos genera un nuevo método en el código:



En este método únicamente debemos llamar al método `llenarTabla` que creamos anteriormente. La única diferencia con la forma en la que lo llamamos en el `Load` es que aprovecharemos el segundo parámetro del método, el cual corresponde al filtro del combobox:

```
//Se carga el datagridview con estudiantes que tengan como nombre el seleccionado en
el combobox nombre
llenarTabla(dgvEstudiantes, cmbNombre.Text, null);
```

El método para el evento debe verse como el siguiente código:

```
/*Método para el evento de cambio de la selección del combobox nombre
Cuando en la interfaz el usuario cambia la selección este metodo se
activa*/
private void cmbNombre_SelectedIndexChanged(object sender, EventArgs e)
{
    //Se carga el datagridview con estudiantes que tengan como nombre el
    seleccionado en el combobox nombre
    llenarTabla(dgvEstudiantes, cmbNombre.Text, null);
}
```

Como se puede ver en la interfaz, queremos implementar dos filtros diferentes. Ya implementamos un filtro por nombre utilizando el combobox y ahora implementaremos un filtro más general utilizando el textbox creado en la interfaz. En este caso el usuario puede filtrar tuplas que en alguno de sus atributos contenga el string introducido en el textbox. Este filtro se hará cuando el usuario presione el botón Buscar, por lo que debemos generar el código de la acción del botón como lo hicimos anteriormente (doble click sobre el control en el form) y añadir el siguiente código:

```
//Llena el datagridview con los estudiantes que contengan en alguno de sus campos el
texto del textbox txtBuscar
llenarTabla(dgvEstudiantes, null, txtBuscar.Text);
```

Este método para el botón debe verse como el siguiente código:

```
/*Método que se activa al dar click en el botón Buscar*/
private void btnBuscar_Click(object sender, EventArgs e)
{
    //Llena el datagridview con los estudiantes que contengan en alguno de
sus campos el texto del textbox txtBuscar
    llenarTabla(dgvEstudiantes, null, txtBuscar.Text);
}
```

Por último, añadimos la acción del LinkLabel “Ir a agregar estudiante” que diseñamos en la interfaz:

```
/*Método que se activa al dar click en el link de agregar estudiante*/
private void lkAgregar_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    //Crea la interfaz AgregarEstudiante y la muestra, desaparece la
interfaz actual
    AgregarEstudiante agregar = new AgregarEstudiante();
    agregar.Show();
    this.Hide();
}
```

10. Añadir código a la pantalla EliminarEstudiante

Esta pantalla también debe tener declarado un objeto de tipo Estudiante:

```
Estudiante estudiante;
```

```
/*Constructor de la clase*/
public EliminarEstudiante()
{
    InitializeComponent();
    estudiante = new Estudiante();
}
```

Además, debemos tener un método para llenar el combobox idéntico al de la pantalla anterior:

```
/*Método para llenar un combobox con datos específicos
Recibe: Un objeto combobox que va a llenar con una consulta específica
Modifica: Llena el combobox que recibe por parámetro con el nombre de todos
los estudiantes que se encuentran en la bd
Retorna: Ningún valor*/
private void llenarCombobox(ComboBox combobox)
{
    //Se obtiene un dataReader con todos los nombres de los estudiantes de
    la base de datos
    SqlDataReader datos = estudiante.obtenerListaNombresEstudiantes();

    /*Si existen datos en la base de datos se carga como primer elemento del
    combobox un dato "Seleccione"
    y luego se cargan todos los datos de la base de datos*/
    if (datos != null)
    {
        combobox.Items.Add("Seleccione");
        while (datos.Read())
        {
            combobox.Items.Add(datos.GetValue(0));
        }
    }
    /*Si no hay tuplas en la base de datos se limpia el combobox y se carga
    unicamente el valor "Seleccione"*/
    else
    {
        combobox.Items.Clear();
        combobox.Items.Add("Seleccione");
    }

    //Se pone por defecto la primera entrada del combobox seleccionada
    combobox.SelectedIndex = 0;
}
```

También necesitamos un método para cargar el dataGridView, pero podemos modificarlo un poco para que no tenga filtros ya que, a diferencia de la pantalla anterior, aquí no necesitamos realizar búsquedas:

```
/*Método para llenar un datagridview con datos específicos
  Recibe: Un control datagridview que va a cargar con datos
  Modifica: carga los datos en el datagridview
  Retorna: ningún valor*/
private void llenarTabla(DataGridView dataGridView)
{
    /*obtiene un dataTable con todos los estudiantes que se encuentran en la
    base de datos (null, null) es para
    vengan todas las tuplas sin ningún filtro*/
    DataTable tabla = estudiante.obtenerEstudiantes(null, null);

    //Se inicializa el source para cargar el datagridview y se le asigna el
    dataTable obtenido
    BindingSource bindingSource = new BindingSource();
    bindingSource.DataSource = tabla;

    dataGridView.AutoSizeColumnsMode(DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)
    ;

    dataGridView.DataSource = bindingSource;

    //Ciclo para darle un ancho a cada columna del datagridview
    proporcionado
    for (int i = 0; i < dgvEstudiantes.ColumnCount; i++)
    {
        dataGridView.Columns[i].Width = 100;
    }
}
```

El método Load del form también debería llamar a ambos métodos, como en la pantalla anterior:

```
/*Método load, este método es llamado cuando se carga la pantalla */
private void EliminarEstudiante_Load(object sender, EventArgs e)
{
    //Llena el combobox de nombres de estudiante
    llenarCombobox(cmbNombre);
    //Llena el datagridview de estudiantes con todas las tuplas de
    estudiante de la interfaz
    llenarTabla(dgvEstudiantes);
}
```

El usuario elimina estudiantes dando click en el botón Eliminar. Así como hemos declarado acciones para botones anteriormente, declararemos la acción del botón eliminar con el siguiente código dentro:

```

/*Llama al procedimiento almacenado mediante el método de la clase estudiante,
elimina por nombre*/
    estudiante.eliminarEstudiante(cmbNombre.Text);
    //Vuelvo a llenar el combobox y el datagridview con los datos
actualizados
    llenarCombobox(cmbNombre);
    llenarTabla(dgvEstudiantes);

```

El método debe verse como el siguiente código:

```

/*Método que se activa al dar click en el botón Buscar*/
private void btnBuscar_Click(object sender, EventArgs e)
{
    /*Llama al procedimiento almacenado mediante el método de la clase
estudiante, elimina por nombre*/
    estudiante.eliminarEstudiante(cmbNombre.Text);
    //Vuelvo a llenar el combobox y el datagridview con los datos
actualizados
    llenarCombobox(cmbNombre);
    llenarTabla(dgvEstudiantes);
}

```

Por último, se agrega la acción del linkLabel “Ir a Agregar Estudiante”:

```

/*Método que se activa al dar click en el link de agregar estudiante*/
private void lkAgregar_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    //Crea la interfaz AgregarEstudiante y la muestra, desaparece la
interfaz actual
    AgregarEstudiante agregar = new AgregarEstudiante();
    agregar.Show();
    this.Hide();
}

```


11. Añadir código en la pantalla Login

Esta pantalla también debe tener declarado un objeto de tipo Estudiante:

```
Estudiante estudiante;

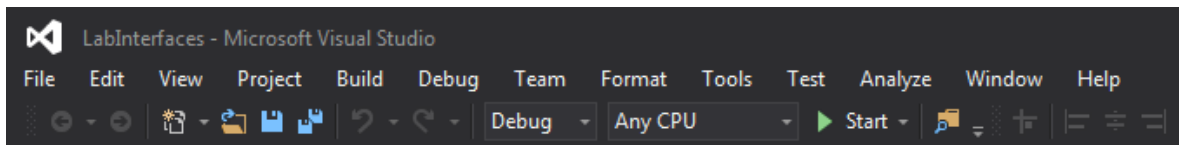
public Login()
{
    InitializeComponent();
    estudiante = new Estudiante();
}
```

Además, se debe validar en el botón Aceptar el usuario y la contraseña, esto se realiza con el siguiente código:

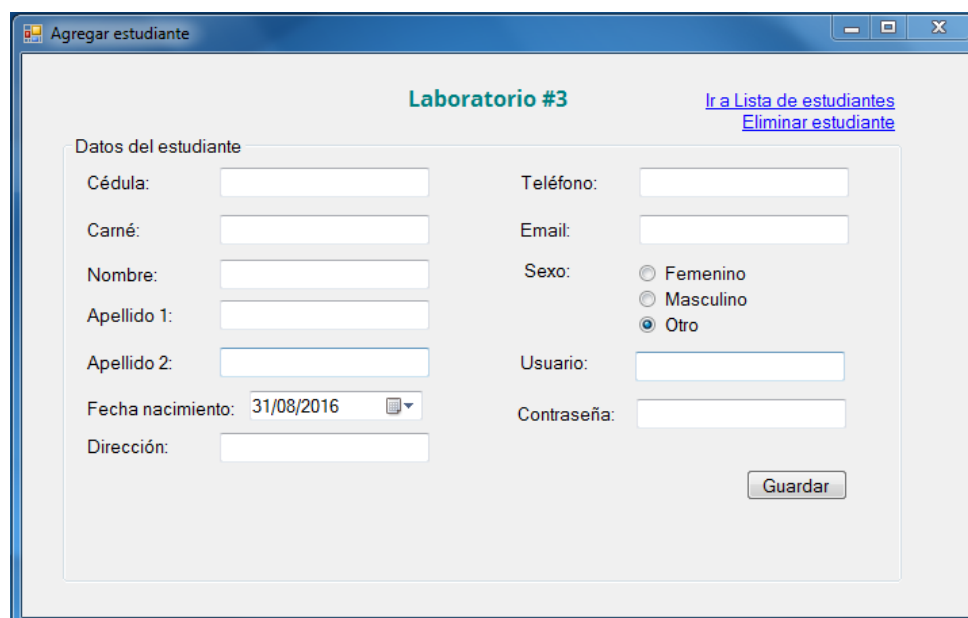
```
/*Método que se activa al dar click en el botón aceptar y valida el usuario
que quiere iniciar sesión*/
private void btnAceptar_Click(object sender, EventArgs e)
{
    //Si los dos campos tienen datos se valida si no muestra un mensaje de
error
    if (txtUsuario.Text != "" && txtPassword.Text != "")
    {
        //si el usuario si existe pasa a la pantalla de Agregar usuario
        if (estudiante.login(txtUsuario.Text, txtPassword.Text) == true)
        {
            //Crea la interfaz AgregarEstudiante y la muestra, desaparece la
interfaz actual
            AgregarEstudiante agregar = new AgregarEstudiante();
            agregar.Show();
            this.Hide();
        }
        //Si el usuario no existe muestra un mensaje de error
        else
        {
            MessageBox.Show("Usuario y/o incorrecto, por favor intente de
nuevo", "Login", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    else
    {
        MessageBox.Show("Por favor introduzca todos los datos para el inicio
de sesión", "Login", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

12. Ejecución de la aplicación

La aplicación se puede ejecutar dando click en el icono Start del menú de la parte superior de Visual:



La pantalla inicial es Agregar Estudiante y se ve de la siguiente forma:

A screenshot of a web application window titled 'Agregar estudiante'. The window has a blue header bar. Below the header, the text 'Laboratorio #3' is displayed in green. To the right of this text are two blue links: 'Ir a Lista de estudiantes' and 'Eliminar estudiante'. The main content area is titled 'Datos del estudiante' and contains two columns of input fields. The left column includes fields for 'Cédula:', 'Carné:', 'Nombre:', 'Apellido 1:', 'Apellido 2:', 'Fecha nacimiento:' (with a date picker showing '31/08/2016'), and 'Dirección:'. The right column includes fields for 'Teléfono:', 'Email:', 'Sexo:' (with radio buttons for 'Femenino', 'Masculino', and 'Otro', where 'Otro' is selected), 'Usuario:', and 'Contraseña:'. A 'Guardar' button is located at the bottom right of the form.

Si realizaron el tutorial en orden verán que la pantalla inicial de la aplicación es Agregar Estudiante, pero lo normal en una aplicación es que de primero se despliegue el Login. Esto podemos modificarlo si nos vamos al archivo Program.cs, aquí nos encontraremos un código como el siguiente:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LabInterfaces
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new AgregarEstudiante());
        }
    }
}

```

Si queremos que se ejecute primero el Login, modificamos la última línea del archivo. Así debería quedar:

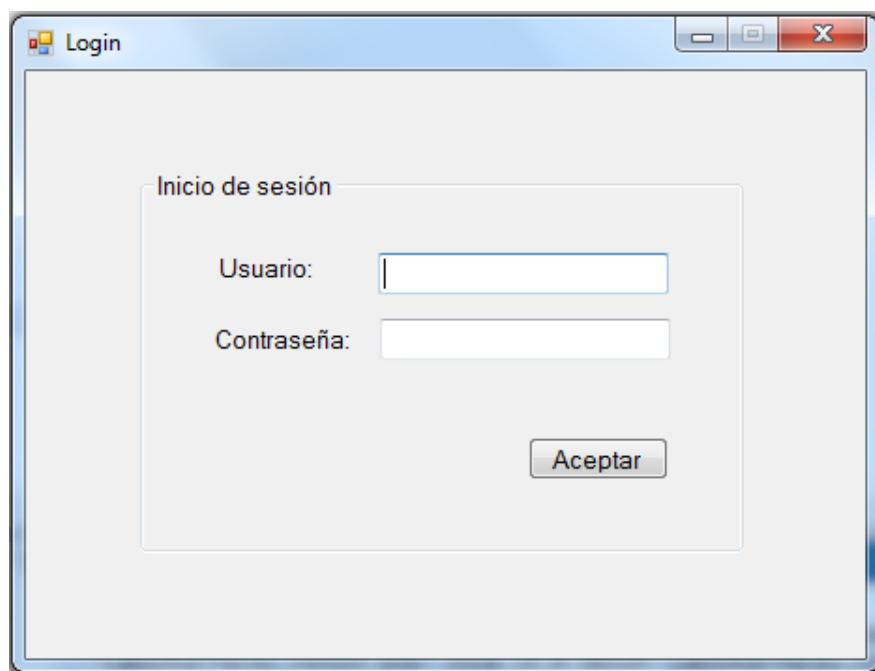
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LabInterfaces
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Login());
        }
    }
}

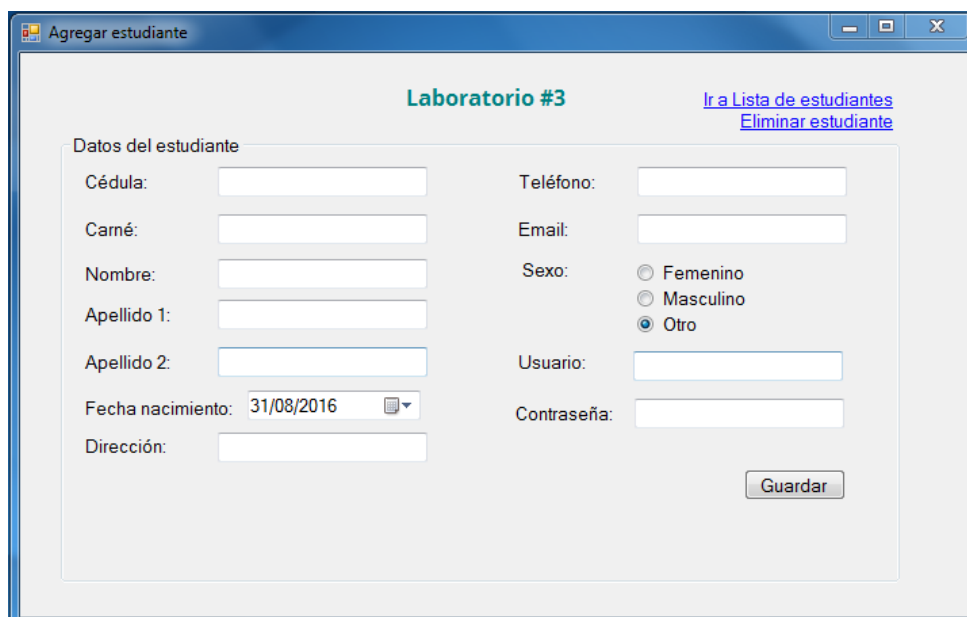
```

Ahora, si volvemos a ejecutar la aplicación se mostrará la siguiente pantalla al inicio:



A screenshot of a Windows-style application window titled "Login". The window has a standard title bar with minimize, maximize, and close buttons. The main content area is light gray and contains a smaller, rounded rectangular box with a light blue border. Inside this box, the text "Inicio de sesión" is at the top. Below it, there are two labels: "Usuario:" followed by a text input field, and "Contraseña:" followed by a text input field. At the bottom right of the box is a button labeled "Aceptar".

Si introducimos un usuario y contraseña correcta, la aplicación nos llevará nuevamente a la siguiente pantalla:



A screenshot of a Windows-style application window titled "Agregar estudiante". The window has a standard title bar. The main content area is light gray and contains a form titled "Laboratorio #3" in green text. To the right of the title are two blue links: "Ir a Lista de estudiantes" and "Eliminar estudiante". Below the title, the text "Datos del estudiante" is followed by a grid of input fields. The fields are: "Cédula:", "Teléfono:", "Carné:", "Email:", "Nombre:", "Sexo:" (with radio buttons for "Femenino", "Masculino", and "Otro", where "Otro" is selected), "Apellido 1:", "Usuario:", "Apellido 2:", "Fecha nacimiento:" (with a date picker showing "31/08/2016"), "Contraseña:", and "Dirección:". A "Guardar" button is located at the bottom right of the form.

Si damos click en el linkLabel "Ir a Lista de Estudiantes" se muestra la pantalla de la siguiente forma:

Lista de Estudiantes

Laboratorio #3 [Ir a Agregar Estudiante](#)

Nombre: Filtro general:

Estudiantes

	Cedula	came	nombre	apellido1	apellido2	email
▶	115650402	B31003	Gaudy	Blanco	Meneses	g@hotmail.com
	19784562	B00000	Alexandra	Martinez		ale@hotmail.com
	874512369	B26184	Luis	Leandro	Jimenez	luis@gmail.com
	987654321	B30675	Larissa	Arroyo	Castillo	lari@gmail.com
	99999999	B30233	Gaudy	Loria	Marin	gaudy@hotmail.com
*						

Si seleccionamos un nombre del combobox el dataGridView va a cargar tuplas filtradas por ese nombre:

Lista de Estudiantes

Laboratorio #3 [Ir a Agregar Estudiante](#)

Nombre: Filtro general:

Estudiantes

	Cedula	came	nombre	apellido1	apellido2	email
▶	19784562	B00000	Alexandra	Martinez		ale@hotmail.com
*						

Si introducimos texto en el textbox y damos click en el botón Buscar el dataGridView cargará las tuplas que coincidan con esa búsqueda:

Lista de Estudiantes

Laboratorio #3 [Ir a Agregar Estudiante](#)

Nombre: Filtro general:

Estudiantes

	Cedula	came	nombre	apellido1	apellido2	email
▶	115650402	B31003	Gaudy	Blanco	Meneses	g@hotmail.com
	99999999	B30233	Gaudy	Loria	Marin	gaudy@hotmail.
*						

Si vamos a la pantalla EliminarEstudiante se cargan todos los datos de la base de datos:

EliminarEstudiante

Laboratorio #3 [Ir a Agregar Estudiante](#)

Nombre:

Estudiantes

	Cedula	came	nombre	apellido1	apellido2	email
▶	115650402	B31003	Gaudy	Blanco	Meneses	g@hotmail.com
	19784562	B00000	Alexandra	Martinez		ale@hotmail.com
	874512369	B26184	Luis	Leandro	Jimenez	luis@gmail.com
	987654321	B30675	Larissa	Arroyo	Castillo	lari@gmail.com
	99999999	B30233	Gaudy	Loria	Marin	gaudy@hotmail.
*						

Seleccionamos un nombre del combobox, en este caso un nombre que está en dos tuplas diferentes:

EliminarEstudiante

Laboratorio #3

[Ir a Agregar Estudiante](#)

Nombre:

Gaudy

Buscar

Estudiantes

	Cedula	came	nombre	apellido1	apellido2	email
▶	115650402	B31003	Gaudy	Blanco	Meneses	g@hotmail.com
	19784562	B00000	Alexandra	Martinez		ale@hotmail.com
	874512369	B26184	Luis	Leandro	Jimenez	luis@gmail.com
	987654321	B30675	Larissa	Arroyo	Castillo	lari@gmail.com
	99999999	B30233	Gaudy	Loria	Marin	gaudy@hotmail.com
*						

<

>

Si damos click en Eliminar se eliminarán las tuplas que coinciden con ese nombre:

EliminarEstudiante

Laboratorio #3

[Ir a Agregar Estudiante](#)

Nombre:

Selecione

Buscar

Estudiantes

	Cedula	came	nombre	apellido1	apellido2	email
▶	19784562	B00000	Alexandra	Martinez		ale@hotmail.com
	874512369	B26184	Luis	Leandro	Jimenez	luis@gmail.com
	987654321	B30675	Larissa	Arroyo	Castillo	lari@gmail.com
*						

<

>