# BugTracker - ABM Usuarios

## 2. Temas

Iniciar la solución que se encuentra en la carpeta src/BugTracker.sln Se agregaron 2 formularios. - frmUsuario: Consulta los usuarios registrados con un filtro por "Nombre" y Perfil (que se elige de un combo) - frmABMUsuario: Permite hacer el Alta y Modificación de usuarios, que previamente se seleccionaron en frmUsuario.

**2.1. Enum (Enumeraciones)**

La palabra clave enum se utiliza para declarar una enumeración, un tipo distinto que consiste en un conjunto de constantes con nombre denominado lista de enumeradores. Normalmente suele ser recomendable definir una enumeración directamente dentro de un espacio de nombres para que todas las clases de dicho espacio puedan tener acceso a esta con la misma facilidad. Sin embargo, una enumeración también puede anidarse dentro de una clase o estructura. De manera predeterminada, el primer enumerador tiene el valor 0 y el valor de cada enumerador sucesivo se incrementa en 1. Por ejemplo:

enum TipoDocumento  
 {  
 DNI, // 0  
 NIE, // 1  
 Pasaporte, // 2  
 Otro = 99  
 };

**2.2. INSERT**

Ejemplo:

INSERT [Usuarios] ([id\_usuario], [id\_perfil], [usuario], [password], [email], [estado])   
 VALUES (1, 1, 'ADMIN', '123', 'admin@bt.com', 'S')

**2.3. UPDATE**

Ejemplo:

UPDATE Usuarios  
 SET usuario = ADMIN,  
 password = 123,   
 email = admin@bt.com,   
 id\_perfil = 1  
 WHERE id\_usuario = 1

**2.4. Manejo de Excepciones**

Cuando se inicia una excepción, el entorno runtime comprueba la instrucción actual para ver si se encuentra dentro de un bloque try. Si es así, se comprueban los bloques catch asociados al bloque try para ver si pueden detectar la excepción. Los bloques Catch suelen especificar tipos de excepción; si el tipo del bloque catch es el mismo de la excepción, o una clase base de la excepción, el bloque catch puede controlar el método. Por ejemplo:

static void TestCatch()  
{  
 try  
 {  
 TestThrow();  
 }  
 catch (CustomException ex)  
 {  
 System.Console.WriteLine(ex.ToString());  
 }  
}

En general con tres de las propiedades de la excepción suele haber suficiente información como para saber la causa del error y poder corregirlo: - ***MessageError***: proporciona una explicación de lo que ha ocurrido.

* ***InnerException***: proporciona información sobre la excepción interna.
* ***StackTrace***: proporciona información de la pila de llamadas antes de la excepción.

Una instrucción try puede contener más de un bloque catch. Se ejecuta la primera instrucción catch que pueda controlar la excepción; las instrucciones catch siguientes se omiten, aunque sean compatibles. Por consiguiente, los bloques catch deben ordenarse siempre de más específico (o más derivado) a menos específico. Por ejemplo:

using System;  
using System.IO;  
  
public class ExceptionExample  
{  
 static void Main()  
 {  
 try  
 {  
 using (var sw = new StreamWriter(@"C:\test\test.txt"))  
 {  
 sw.WriteLine("Hello");  
 }   
 }  
 // Put the more specific exceptions first.  
 catch (DirectoryNotFoundException ex)  
 {  
 Console.WriteLine(ex);   
 }  
 catch (FileNotFoundException ex)  
 {  
 Console.WriteLine(ex);   
 }  
 // Put the least specific exception last.  
 catch (IOException ex)  
 {  
 Console.WriteLine(ex);   
 }  
  
 Console.WriteLine("Done");   
 }  
}

Para que el bloque catch se ejecute, el entorno runtime busca bloques finally. Los bloques Finally permiten al programador limpiar cualquier estado ambiguo que pudiera haber quedado tras la anulación de un bloque try o liberar los recursos externos (como identificadores de gráficos, conexiones de base de datos o flujos de archivos) sin tener que esperar a que el recolector de elementos no utilizados del entorno runtime finalice los objetos. Por ejemplo:

static void TestFinally()  
{  
 System.IO.FileStream file = null;  
 //Change the path to something that works on your machine.  
 System.IO.FileInfo fileInfo = new System.IO.FileInfo(@"C:\file.txt");  
  
 try  
 {  
 file = fileInfo.OpenWrite();  
 file.WriteByte(0xF);  
 }  
 finally  
 {  
 // Closing the file allows you to reopen it immediately - otherwise IOException is thrown.  
 if (file != null)  
 {  
 file.Close();  
 }  
 }  
  
 try  
 {  
 file = fileInfo.OpenWrite();  
 System.Console.WriteLine("OpenWrite() succeeded");  
 }  
 catch (System.IO.IOException)  
 {  
 System.Console.WriteLine("OpenWrite() failed");  
 }  
}

Las excepciones están representadas por clases derivadas de [Exception](https://docs.microsoft.com/es-es/dotnet/api/system.exception). Esta clase identifica el tipo de excepción y contiene propiedades que tienen los detalles sobre la excepción. Iniciar una excepción implica crear una instancia de una clase derivada de excepción, configurar opcionalmente las propiedades de la excepción y luego producir el objeto con la palabra clave throw. Por ejemplo:

static void TestCatch()  
{  
 try  
 {  
 TestThrow();  
 }  
 catch (CustomException ex)  
 {  
 System.Console.WriteLine(ex.ToString());  
 }  
}

## 3. Actividad

Sobre el formulario frmABMUsuario realizar las siguientes actividades: **3.1. Validar Campos Obligatorios.** En el método ValidarCampos agregar las validaciones que faltan sobre los textbox: \* txtPassword \* txtConfirmarPass \* cboPerfil

**3.2. Actualizar Usuario.** Implementar lógica en BusinessLayer y DataAccessLayer para actualizar los datos de un usuario.

**3.3. Habilitar/Deshabilitar Usuario.** Idem punto anterior pero solo para actualizar el estado del Usuario:

* Estado = "S"-> Habilitado .
* Estado = "N" -> Deshabilitado