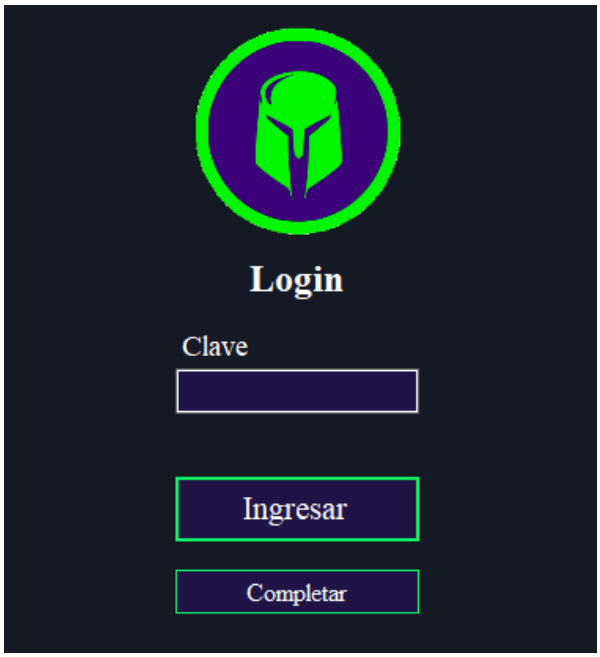
TP_N°4

GIMNASIO MAXIMUS

Matías Ferrari | Laboratorio II | 30 de julio de 2022

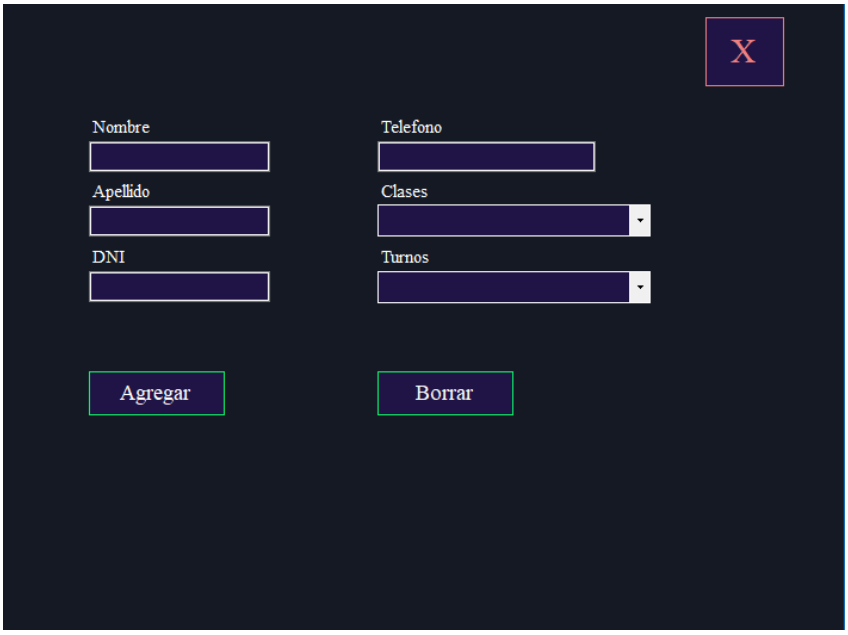
MAXIMUS

Gimnasio Maximus es una aplicación que trata de facilitar el manejo de clientes del establecimiento. Se podra cargar datos de nuevos clientes y asignarles las clases y sus horarios correspondientes.



Para poder acceder al sistema se deberá ingresar la contraseña “maximus123” o apretar el botón completar para que autocomplete solo (Esta función solo estaría disponible para facilitar la corrección).

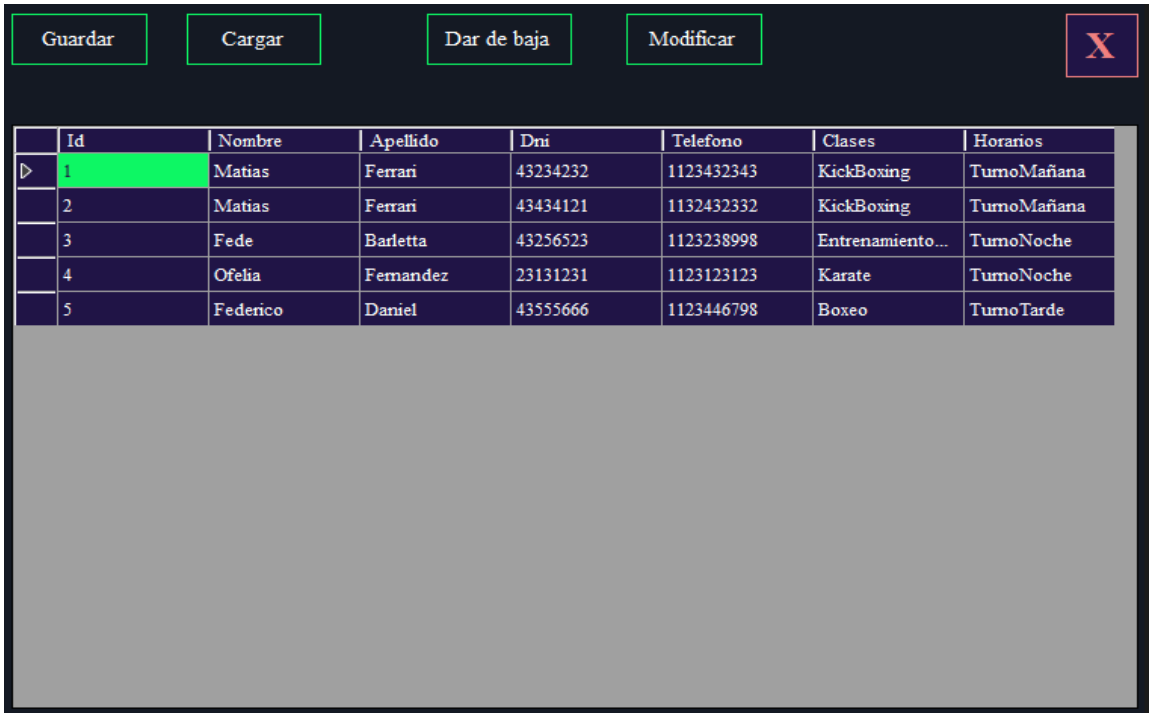
Al hacer click en “Agregar Clientes” se abrirá otra pestaña donde se podrá registrar un nuevo cliente. Y al hacer click en el botón “Lista de Clientes” se podrán ver todos los clientes registrados y de ser el caso poder modificarlos y eliminarlos.



Una vez presionado el botón “Agregar Clientes”

Aparecerá este formulario con todos los campos

Que se necesitaran llenar para poder agregar.



Id	Nombre	Apellido	Dni	Telefono	Clases	Horarios
1	Matias	Ferrari	43234232	1123432343	KickBoxing	TurnoMañana
2	Matias	Ferrari	43434121	1132432332	KickBoxing	TurnoMañana
3	Fede	Barletta	43256523	1123238998	Entrenamiento...	TurnoNoche
4	Ofelia	Fernandez	23131231	1123123123	Karate	TurnoNoche
5	Federico	Daniel	43555666	1123446798	Boxeo	TurnoTarde

- Una vez presionado el botón “Lista de Clientes” en el menú principal se abrirá este formulario, que contiene un data grid que muestra toda la lista de clientes con su información separados por sus campos.

- También posee botones de “Guardar” que lo que hará exportar la información de la lista a un archivo XML o JSON. Para después poder ser levantados al apretar el botón “Cargar” que lo que hará es levantar esa información del archivo elegido.

- Los botones “Dar de baja” y “Modificar” hacen una baja o modificación de un cliente en la base de datos y luego actualizara el data grid para poder ver los cambios.

Si se selecciona una fila y se presiona el botón “Modificar” en el formulario lista de clientes, se abrirá este form que es igual al form de agregar cliente, con la diferencia que aparecerán todos los campos del cliente ya cargados y listos para ser modificados.

TEMAS USADOS:

1-Excepciones

2-Pruebas unitarias

3-Generics

4-Interfaces

5-Archivos y serialización

6-Base de datos

7-Delegados

8-Hilos

9-Eventos

1- EXCEPCIONES: Utilizado para validar algunos datos

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Library
{
    3 referencias
    public class DatosNoValidosException : Exception
    {
        2 referencias
        public DatosNoValidosException(string mensaje) : base(mensaje)
        {

        }
    }
}
```

2- **PRUEBAS UNITARIAS: Se usa para verificar si la extensión del archivo es correcta y también para ver si los métodos devuelven lo que se pide.**

```
[TestMethod]
0 referencias
public void TestMethod_TestInsertar()
{
    Assert.IsTrue(SQLManagment.Insertar(new(100,"asd","dsa",43234323,1123233223,EClases.KickBoxing,EHorarios.TurnoMañana))==true);
}

[ExpectedException(typeof(DatosNoValidosException))]
[TestMethod]
0 referencias
public void TestMethod_TestVerficarDatos()
{
    //Assert
    bool expected = true;
    //Act
    bool actual = Negocio.VerificarDatos(4324332, 123433443);// tira la excepcion que se expecta

    //Arrange
    Assert.AreEqual(expected , actual);
}
```

```
[TestClass]
0 referencias
public class UnitTestExtension
{
    [TestMethod]
    0 referencias
    public void TestMethod_ExtensionJson()
    {
        //Assert
        string expected = ".json";

        //Act
        ExtJson<Cliente> ext = new ExtJson<Cliente>();
        string actual = ext.Extension;

        //Arrange
        Assert.AreEqual(expected, actual);

    }

    [TestMethod]
    0 referencias
    public void TestMethod_ExtensionXml()
    {
        //Assert
        string expected = ".xml";

        //Act
        ExtXml<Cliente> ext = new ExtXml<Cliente>();
        string actual = ext.Extension;

        //Arrange
        Assert.AreEqual(expected, actual);

    }
```

3- GENERICS: Implementado para las clases que manejan archivos y serializan a XML y JSON.

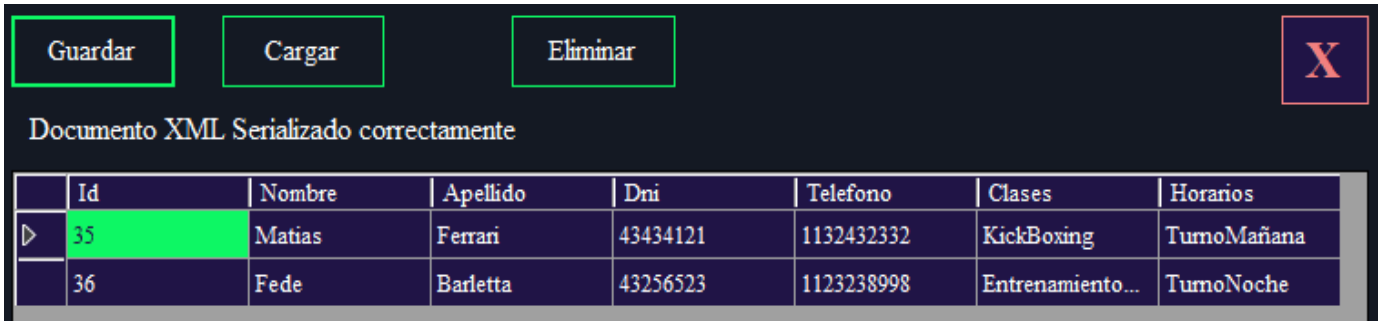
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace Library
{
    2 referencias
    public class ExtJson<T> : ISerializacion<T> where T : class
    {
        3 referencias
        public string Extension { get { return ".json"; } }

        2 referencias
        public void Escribir(string path, T texto, Action<string> mostrarMensaje)
        {
            if (ValidacionArchivo(path) && ValidacionExtension(path))
            {
                Serializar(path, texto, mostrarMensaje);
            }
        }
```

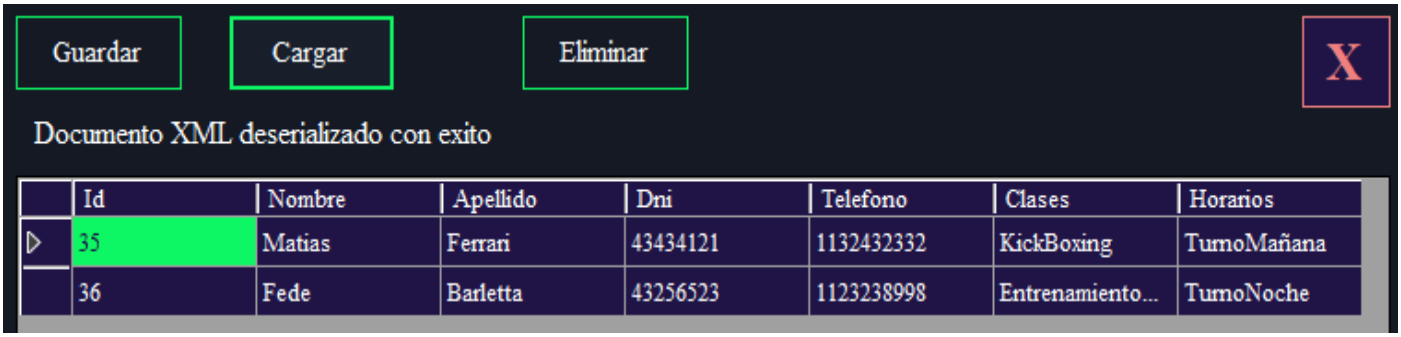
4- INTERFACES: Se utilizan para las clases ExtJson y ExtXml.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Library
{
    2 referencias
    public interface ISerializacion<T>
    {
        6 referencias
        public string Extension { get; }
        4 referencias
        public void Escribir(string path, T contenido, Action<string> mostrarMensaje);
        4 referencias
        public T Leer(string path, Action<string> mostrarMensaje);
        5 referencias
        public bool ValidacionArchivo(string path);
        7 referencias
        public bool ValidacionExtension(string path);


    }
}
```

5- SERIALIZACIÓN: Serializar y deserializar archivos XML y JSON.

```
/// <summary>
/// Recibe por parametros el path, el contenido y un delegado para mostrar un mensaje si el serializado fue exitoso
/// </summary>
/// <param name="path"></param>
/// <param name="texto"></param>
/// <param name="mostrarMensaje"></param>
2 referencias
private void Serializar(string path, T texto, Action<string> mostrarMensaje)
{
    using (StreamWriter stW = new StreamWriter(path))
    {
        string json = JsonSerializer.Serialize(texto);
        stW.WriteLine(json);
        if(mostrarMensaje is not null)
        {
            mostrarMensaje.Invoke("Documento JSON Serializado correctamente");
        }
    }
}
/// <summary>
/// recibe por parametros el path y un delegado para mostrar en un label que la operacion de deserializar fue exitosa
/// </summary>
/// <param name="path"></param>
/// <param name="mostrarMensaje"></param>
/// <returns></returns>
2 referencias
public T Leer(string path, Action<string> mostrarMensaje)
{
    if (ValidacionArchivo(path) && ValidacionExtension(path))
    {
        using (StreamReader stR = new StreamReader(path))
        {
            string json = stR.ReadToEnd();
            if(mostrarMensaje is not null)
            {
                mostrarMensaje.Invoke("Documento JSON deserializado con exito");
            }
            return JsonSerializer.Deserialize<T>(json);
        }
    }
    return null;
}
```





6- BASE DE DATOS: Al pulsar el botón “Lista de Clientes” se abrirá un form con un data grid que contendrá la información que hay en la base de datos. Se podrán hacer inserciones, modificaciones y eliminaciones de clientes. Se adjuntará un script en la carpeta del TP4.

```
5 referencias
public static class SQLManagment
{
    static string connectionString;
    static SqlCommand command;
    static SqlConnection connection;

    0 referencias
    static SQLManagment()
    {
        connectionString = "Data Source=.;Initial Catalog=TP4_DB;Integrated Security=True";
        connection = new SqlConnection(connectionString);
        command = new SqlCommand();
        command.CommandType = System.Data.CommandType.Text;
        command.Connection = connection;
    }
    /// <summary>
    /// Inserta un cliente en la base de datos
    /// </summary>
    /// <param name="cliente"></param>
    1 referencia
    public static void Insertar(Cliente cliente)
    {
        try
        {
            command.Parameters.Clear();
            connection.Open();

            command.CommandText = "INSERT INTO CLIENTES VALUES (@Nombre,@Apellido,@DNI,@Telefono,@Clases,@Turno)";
            command.Parameters.AddWithValue("@Nombre", cliente.Nombre);
            command.Parameters.AddWithValue("@Apellido", cliente.Apellido);
            command.Parameters.AddWithValue("@DNI", cliente.Dni);
            command.Parameters.AddWithValue("@Telefono", cliente.Telefono);
            command.Parameters.AddWithValue("@Clases", cliente.Clases.ToString());
            command.Parameters.AddWithValue("@Turno", cliente.Horarios.ToString());

            command.ExecuteNonQuery();
        }
        catch (Exception)
        {
            throw;
        }
        finally
        {
            connection.Close();
        }
    }
```

7- **DELEGADOS: Utilizado al momento de deserializar o serializar un archivo, se le muestra por un label si la operación fue exitosa.**

```
public delegate void DelegadoMensajeExito(string mensaje);
```

```
public void Escribir(string path, T texto, Action<string> mostrarMensaje)
{
    if (ValidacionArchivo(path) && ValidacionExtension(path))
    {
        Serializar(path, texto, mostrarMensaje);
    }
}
```

```
try
{
    switch (Path.GetExtension(LastFile))
    {
        case ".json":
            this.extJson.Escribir(LastFile, Negocio.Clientes, ActualizarComponenetesFormulario);
            break;
        case ".xml":
            this.extXml.Escribir(LastFile, Negocio.Clientes, ActualizarComponenetesFormulario);
            break;
    }
}
catch (Exception e)
{
    MessageBox.Show(e.Message);
}
```

8- **HILOS: Se utiliza para hacer un reloj digital con la clase Timer (No se me ocurrió que otro uso darle).**

```
1 referencia
private void MenuPrincipal_Load(object sender, EventArgs e)
{
    System.Timers.Timer t = new System.Timers.Timer();
    t.Interval = 1000;
    t.Elapsed += Timer_Elapsed;
    Task.Run(() => { t.Start(); });
}

1 referencia
private void Timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
{
    Invoke(new MethodInvoker(delegate ()
    {
        lblRelog.Text = DateTime.Now.ToString("HH:mm:ss");
    }));
}
```

9- **EVENTOS: Se utiliza cuando la lista no tiene ningún valor y lo que hace es invocar a un método que desactivara todos los botones del formulario “Lista de Clientes” excepto el botón de salir.**

```
public delegate void DelegadoDesactivarBoton();
public event DelegadoDesactivarBoton EventoDesactivar;
```

```
EventoDesactivar += DesactivarBotones;
```

```
/// <summary>
/// Actualiza el datagrid cada vez que hay un cambio
/// </summary>
3 referencias
private void RefrescarDataGrid()
{
    List<Cliente> listaClientes = Negocio.Clientes;
    listaClientes = SQLManagment.Leer();

    if(listaClientes.Count>0)
    {
        dgvListaClientes.DataSource = null;
        dgvListaClientes.DataSource = new List<Cliente>(listaClientes);
    }else
    {
        EventoDesactivar.Invoke();
        MessageBox.Show("No hay datos para mostrar.", "Información", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```