Trabajo Práctico N° 4 - Laboratorio de Computación II
Ledesma Leonel Franco - DIV 2E

Descripción del Programa:

El programa es un sistema para la atención al cliente de un Centro Médico. Donde se puede dar de alta pacientes, profesionales, crear turnos y buscar estas entidades, eliminarlas o modificarlas.

Funcionamiento del programa:

En la sección
Configuración:

- Se puede habilitar la opción de autoguardado: Se puede guardar en base de datos y/o en archivo local.
- La dirección donde se quiere guardar los datos del Centro Médico.
- Se puede recuperar los datos desde archivo local o base de datos.
- Se va a poder elegir el nombre del Centro Médico.
- Se debe cargar las especialidades que va a poder registrar a cada profesional a la hora de darle el alta.
- Se puede modificar el horario de atención

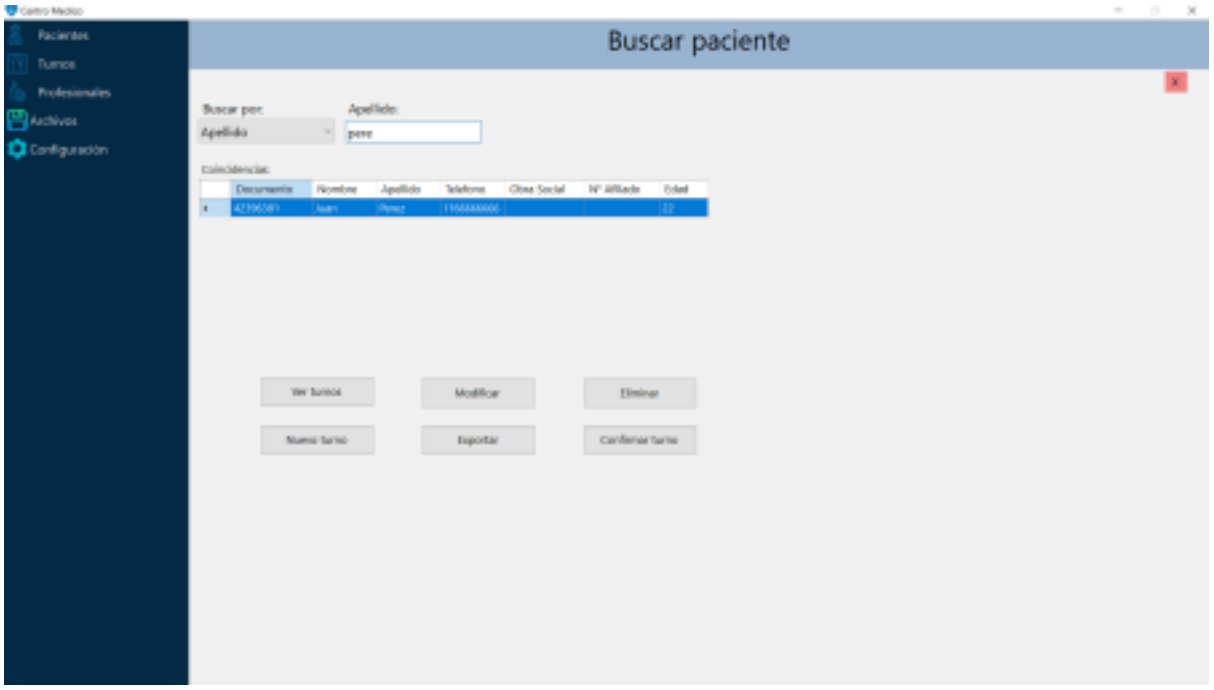
Pacientes:

● Nuevo: Se podrá dar de alta un nuevo paciente (1)

● Ver/Buscar/Modificar: Se podrá buscar los pacientes por apellido, documento, nombre, teléfono o todos. Una vez seleccionado un paciente, se puede crear un turno, modificar (haciendo doble click en la tabla o a través del botón), eliminar, exportar, ver turnos o confirmar turno. (2)

(1)

(2)

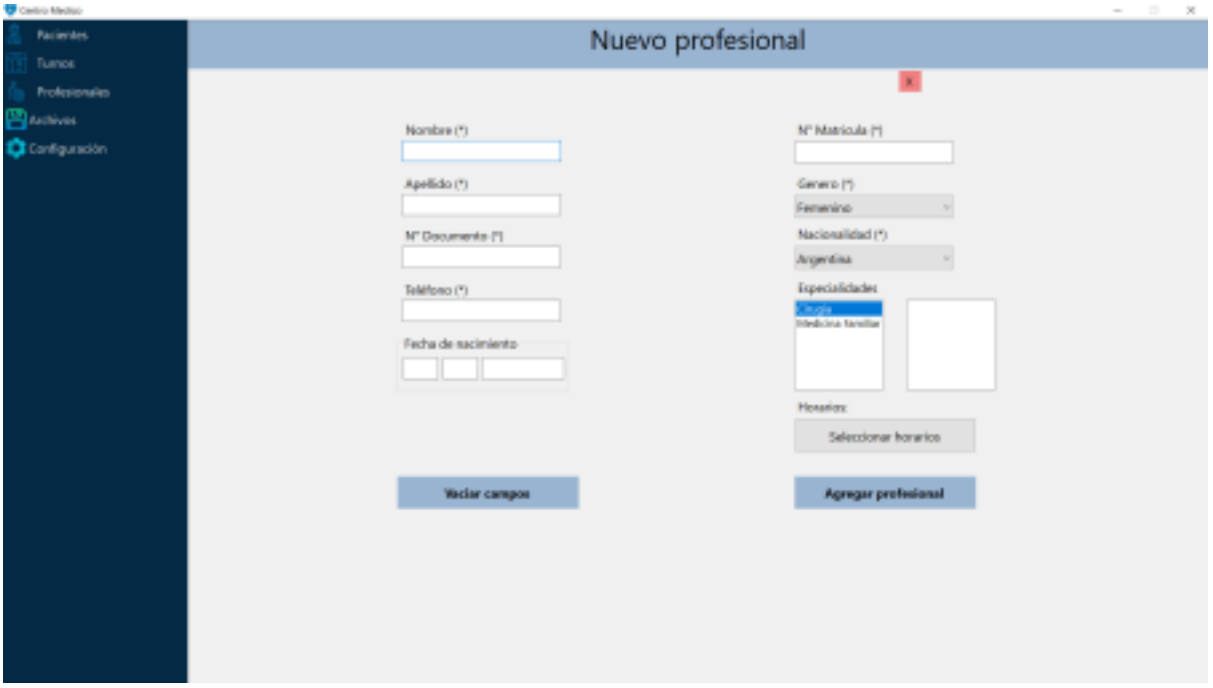


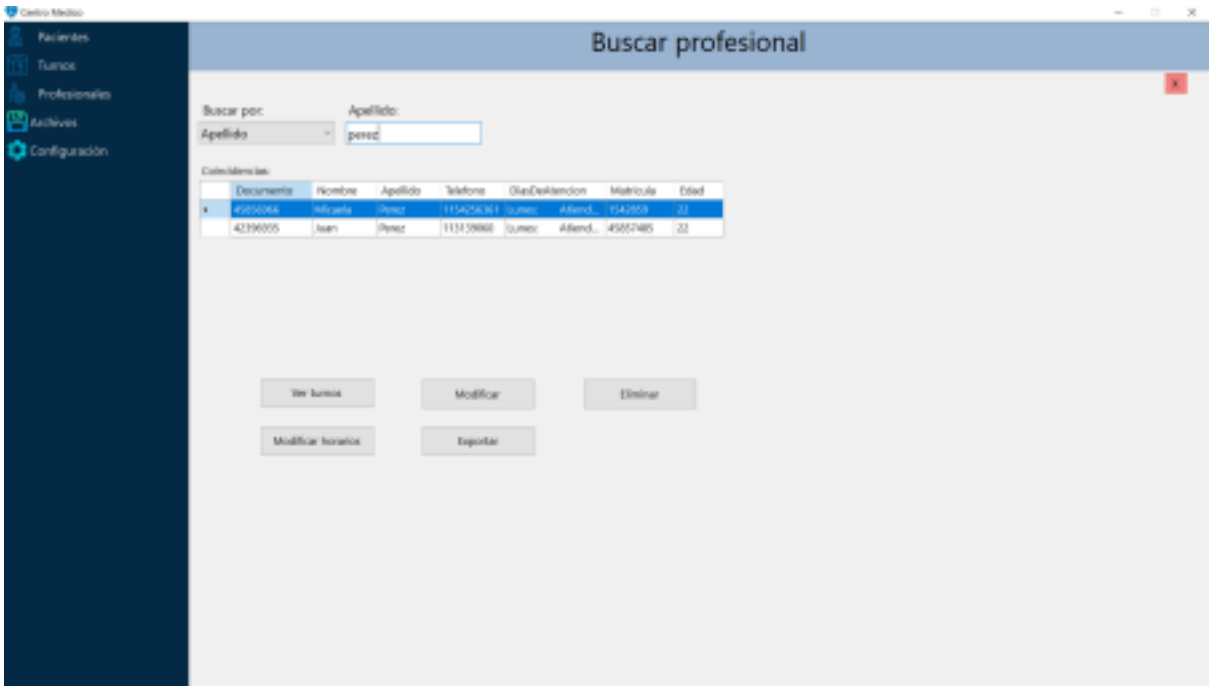
Profesionales:

● Nuevo: Se podrá dar de alta un nuevo profesional, seleccionando sus especialidades haciendo doble click en la especialidad y horarios de atención, donde abre en pantalla el formulario para la selección de horarios. (1)

● Ver/Buscar/Modificar: Se puede ver turnos, modificar (haciendo doble click en la tabla o a través del botón), modificar horarios, exportar o eliminar. (2)

(1)





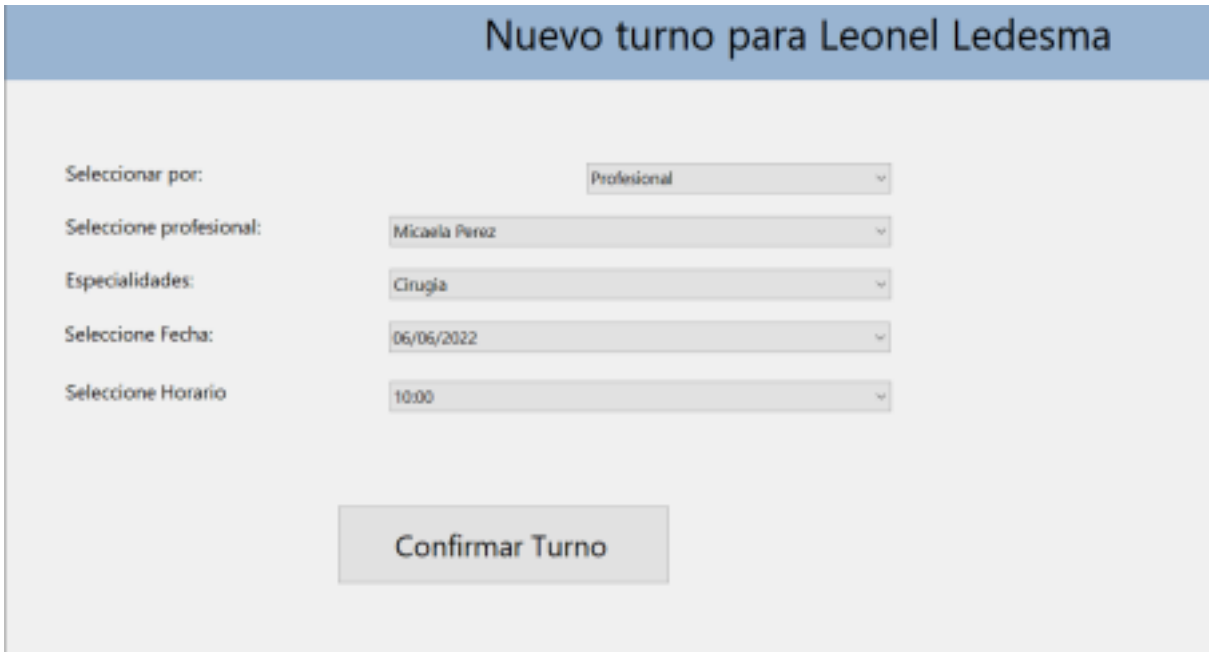
(2)

Turnos:

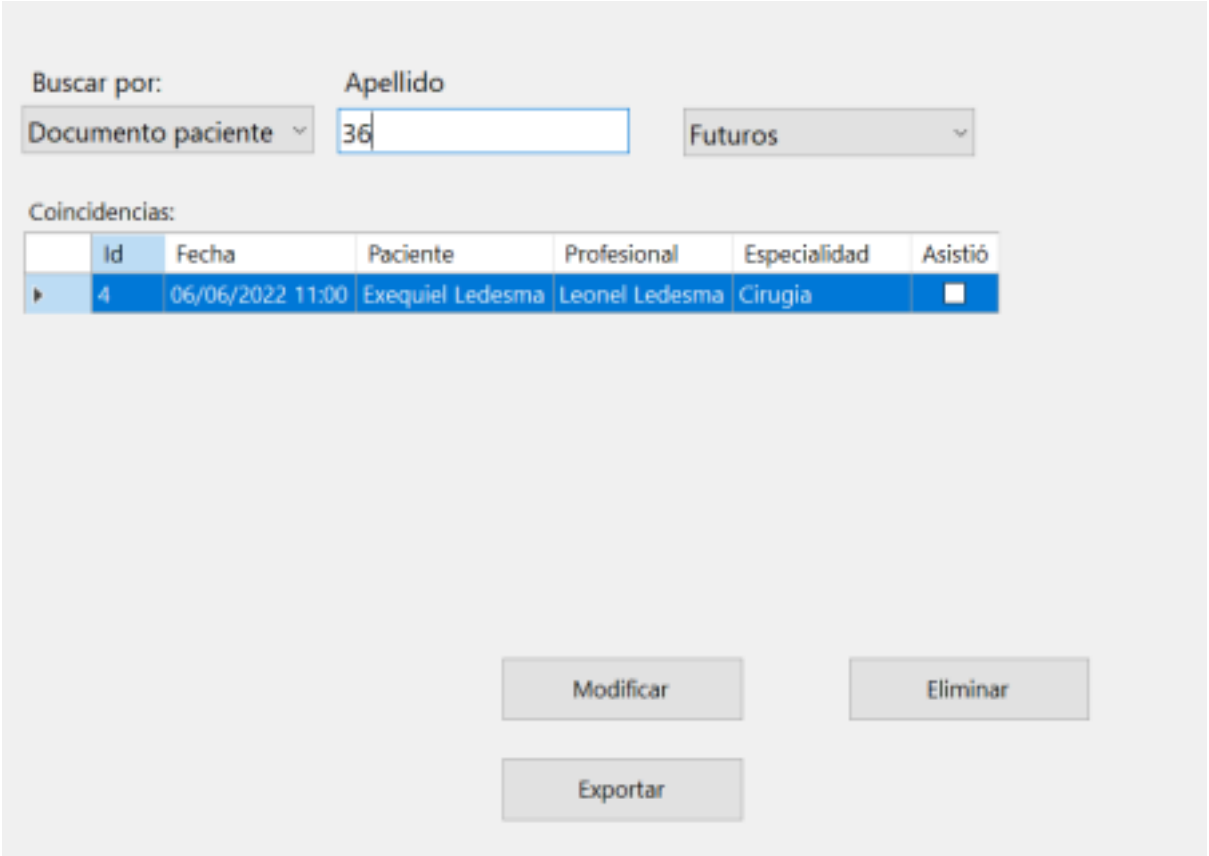
● **Nuevo turno:** Abre el formulario de búsqueda de pacientes, para hacer click en el botón de Nuevo Turno. Una vez seleccionado el paciente y hecho clic en Nuevo turno, se puede seleccionar por Especialidad o Profesional, abriendo así diferentes menús para la elección del turno, teniendo en cuenta los días que atiende el centro, y los días que atiende el profesional seleccionado.(1)

● **Buscar turno:** Abre el formulario de búsqueda de turnos, donde se puede buscar por documento del paciente, apellido del profesional, id del turno, o todos. Al mismo tiempo que se pueden buscar futuros, pasados o todos. Donde se puede modificar, exportar o eliminar dicho turno. (2)

● **Ver turnos:** Se pueden ver los turnos de una forma más enfocada, por fecha o todas las fechas, por profesional o todos los profesionales y por documento del paciente o todos los pacientes, en todas sus combinaciones. Pudiendo exportar todas las coincidencias. (3) **(1)**



(2)



(3)

Archivos:

Se pueden exportar las siguientes listas a .json o .xml. ● **Importar pacientes**

● **Importar profesionales**

● **importar centro**

● **Exportar pacientes**

● **Exportar profesionales**

● **Exportar centro**

Temas a aplicar:

Los temas a aplicar son:

1. Excepciones:

Fueron aplicadas varias veces en el proyecto para poder controlar los posibles errores en el tiempo de ejecución. Algunos de los ejemplos son:

```
public CentroMedico DeserializarFromXML(string path)
{
    StreamReader streamReader = null;
    try
    {
        streamReader = new StreamReader(path);

        XmlSerializer xml = new XmlSerializer(typeof(CentroMedicoData));
        CentroMedicoData centroMedico = xml.Deserialize(streamReader) as CentroMedicoData;
        return (CentroMedico)centroMedico;

    }
    catch (FileNotFoundException e)
    {
        throw new DeserializarException("No se ha encontrado el archivo del centro medico.", e);
    }
    catch (Exception e)
    {
        throw new DeserializarException("Error al deserializar del centro medico", e);
    }
    finally
    {
        if (streamReader != null)
            streamReader.Close();
    }
```

Donde se creó un nuevo tipo de excepción con el fin de controlar este tipo de errores al deserializar las entidades.

2. Pruebas unitarias:

Se implementaron para confirmar el correcto funcionamiento de algunas implementaciones.

```
public class TestCentroMedico
{
    [TestMethod]
    0 referencias
    public void EliminarPaciente_CuandoNoExisteEnLaListaDePacientes_DeberiaRetornarFalse()
    {
        // Arrange
        CentroMedico centroMedico = CentroMedico.Instanciar("centro");
        Paciente paciente = new Paciente("Leonel", "Ledesma", new System.DateTime(2000, 02, 10),
            Persona.eGenero.Masculino, Persona.eNacionalidad.Argentina, "41458745", "1145858574", "", "", "");
        centroMedico.AgregarPersona(paciente);
        Paciente paciente2 = new Paciente("Juan", "Perez", new System.DateTime(1960, 10, 05),
            Persona.eGenero.Masculino, Persona.eNacionalidad.Argentina, "18458969", "458599*/", "", "", "");

        bool result;
        //act
        result = centroMedico.RemoverPersona(paciente2);

        //Assert
        Assert.IsFalse(result);
    }
```

3. Tipos genéricos:

Entre otros usos lo utilice para poder eliminar en listas donde contengan personas aquellos que estén en lista y en list, retornando una nueva lista sin estos duplicados.

```
namespace Entidades.Extension_Class
{
    0 referencias
    public static class ListPersonasExtendido
    {
        3 referencias
        public static List<T> EliminarCoincidencias<T>(this List<T> lista, List<T> list) where T : Persona
        {
            List<T> nueva = new List<T>();
            nueva.AddRange(lista);
            foreach (T a in lista)
            {
                foreach(T b in list)
                {
                    if (a.Documento == b.Documento)
                    {
                        nueva.Remove(a);
                    }
                }
            }

            return nueva;
        }

    }
}
```

4. Interfaces:

Utilice esta interfaz para identificar aquellas entidades que yo voy a querer deserializar.

```
public interface IDeserializable<T>
{
    /// <summary> Deserializa la entidad desde un archivo .XML, devolviendo un objet ...
    2 referencias
    public T DeserializarFromXML(string path);

    /// <summary> Deserializa la entidad desde un archivo .JSON, devolviendo un obje ...
    2 referencias
    public T DeserializarFromJson(string path);
}
```

5. Serialización:

Tanto las interfaces como tipos genéricos fueron utilizados en diferentes ámbitos, con el fin de facilitar ciertas implementaciones de la serialización. Creando así las interfaces ISerializable e IDeserializable.

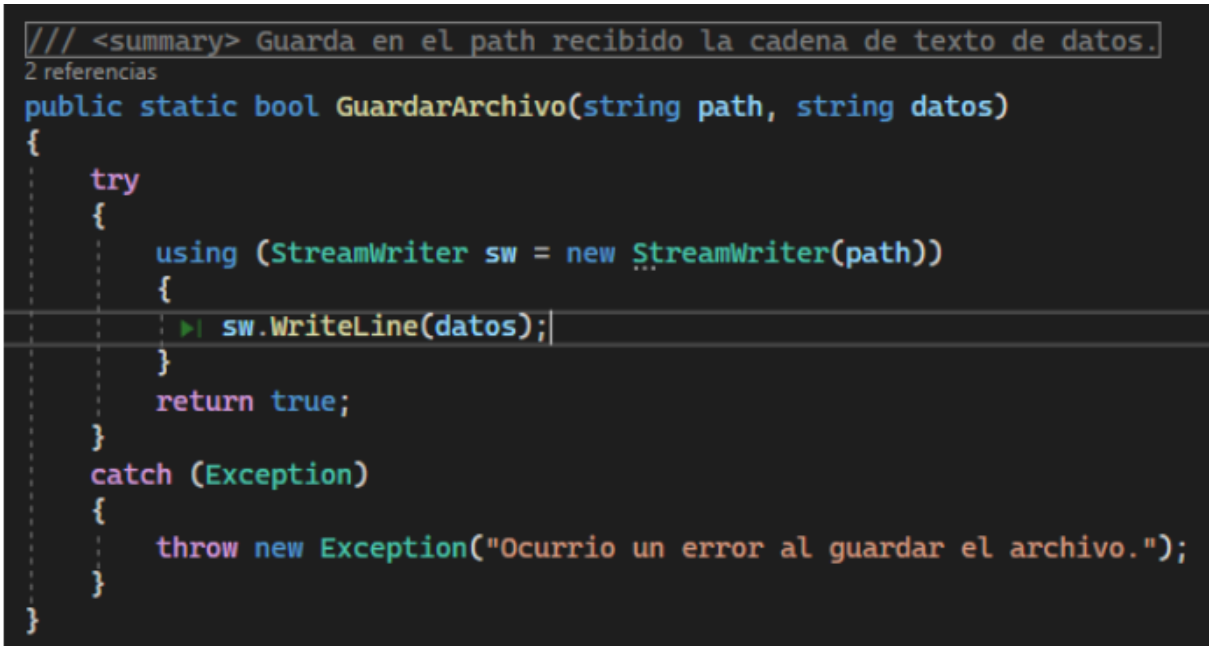
```
/// <summary> Serializa una lista del centro medico a un archivo XML.
3 referencias
public void SerializarListaToXML<T>(string fullPath, List<T> lista)
{
    StreamWriter streamWriter = null;
    try
    {
        streamWriter = new StreamWriter(fullPath);

        XmlSerializer xml = new XmlSerializer(typeof(List<T>));
        xml.Serialize(streamWriter, lista);
    }
    catch (Exception e)
    {
        throw new SerializarException("Error al serializar la lista", e);
    }
    finally
    {
        if (streamWriter != null)
            streamWriter.Close();
    }
}
```

Permitiendo así poder serializar cualquier tipo de lista del centro médico.

6. Archivos:

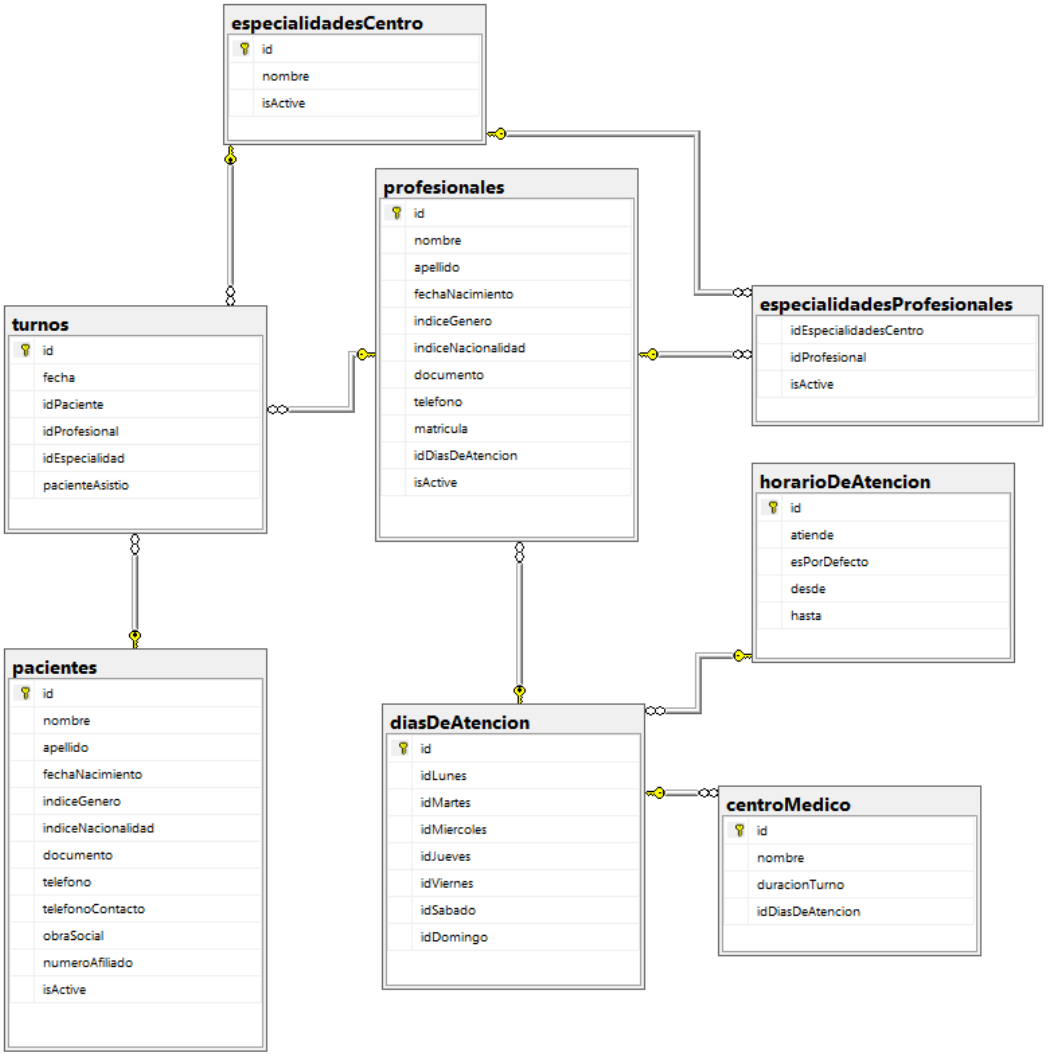
Además de utilizarse para la serialización y deserialización, lo utilizo para guardar la configuración de autoguardado.



7 y 8. SQL:

Para realizar la base de datos del centro médico, fui utilizando varias tablas relacionales, con sus primary key y foreign key de la manera que encontré más óptima.

Cree la clase GestorSQL para el manejo de toda la información relacionada a la base de datos. Algunos de los métodos que implemente fueron:




```
335         public static bool GuardarCentroMedico(CentroMedico centroMedico)
336         {
337             try
338             {
339                 string query;
340                 bool esActualizacion = false;
341                 if (BuscarCentroMedico())
342                 {
343                     query = "UPDATE centroMedico set nombre = @nombre, duracionTurno = @duracionTurno WHERE id = @id";
344                     esActualizacion = true;
345                 }
346                 else
347                 {
348                     query = "INSERT INTO centroMedico (nombre, duracionTurno, idDiasDeAtencion) VALUES (@nombre, @duracionTurno, @idDiasDeAtencion)";
349                 }
350 
351 
352                 using (SqlConnection connection = new SqlConnection(GestorSQL.cadenaConexion))
353                 {
354                     SqlCommand cmd = new SqlCommand(query, connection);
355                     cmd.Parameters.AddWithValue("@id", 1);
356                     cmd.Parameters.AddWithValue("@nombre", centroMedico.Nombre);
357                     cmd.Parameters.AddWithValue("@duracionTurno", (int)centroMedico.DuracionDeTurnos);
358                     if (esActualizacion)
359                     {
360                         GestorSQL.ActualizarDiasDeAtencion(centroMedico.DiasDeAtencion, GestorSQL.BuscarIdDiasDeAtencionCentro());
361                     }
362                     else
363                     {
364                         cmd.Parameters.AddWithValue("@idDiasDeAtencion", GestorSQL.GuardarDiasDeAtencion(centroMedico.DiasDeAtencion));
365                     }
366                     connection.Open();
367 
368                     cmd.ExecuteNonQuery();
```

```
/// <summary>
/// Elimina un turno.
/// </summary>
/// <param name="turno"></param>
/// <returns>True si pudo eliminarla, false si no</returns>
1 referencia
public static bool EliminarTurno(Turno turno)
{
    string query = "DELETE FROM turnos WHERE id = @id";

    using (SqlConnection connection = new SqlConnection(GestorSQL.cadenaConexion))
    {
        SqlCommand cmd = new SqlCommand(query, connection);
        cmd.Parameters.AddWithValue("@id", BuscarIdTurno(turno));

        connection.Open();
        if (cmd.ExecuteNonQuery() == 1)
            return true;

        return false;
    }
}
```

```
/// <summary>
/// Actualiza los dias de atencion que correspondan al id.
/// </summary>
/// <param name="diasDeAtencion"></param>
/// <param name="id"></param>
2 referencias
public static void ActualizarDiasDeAtencion(DiasDeAtencion diasDeAtencion, int id)
{
    string query = "SELECT idLunes, idMartes, idMiercoles, idJueves, idViernes, idSabado, idDomingo FROM diasDeAtencion WHERE id = @id";

    using (SqlConnection connection = new SqlConnection(GestorSQL.cadenaConexion))
    {
        SqlCommand cmd = new SqlCommand(query, connection);
        cmd.Parameters.AddWithValue("@id", id);
        connection.Open();

        List<int> idsHorarioDeAtencion = new List<int>();
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            idsHorarioDeAtencion.Add(reader.GetInt32(0));
            idsHorarioDeAtencion.Add(reader.GetInt32(1));
            idsHorarioDeAtencion.Add(reader.GetInt32(2));
            idsHorarioDeAtencion.Add(reader.GetInt32(3));
            idsHorarioDeAtencion.Add(reader.GetInt32(4));
            idsHorarioDeAtencion.Add(reader.GetInt32(5));
            idsHorarioDeAtencion.Add(reader.GetInt32(6));
        }

        int indice = 1;
        foreach (int idHorarioDeAtencion in idsHorarioDeAtencion)
        {
            GestorSQL.ActualizarHorarioDeAtencion(idHorarioDeAtencion, diasDeAtencion[indice]);
            indice++;
        }

    }
}
```

Utilice en el 100% consultas parametrizadas para tener certeza sobre la seguridad de los datos.

9. Delegados y Expresiones Lambda:

Se realizaron delegados propios en eventos para el orden del código y poder identificar a través del tipo de delegado a que se hace referencia, y poder avisar sobre las excepciones generadas en hilos secundarios.

```
public delegate void GuardarExceptionSQLHandler();
public delegate void RecuperarExceptionSQLHandler();

91 referencias
public class GestorSQL
{
    private static string cadenaConexion;
    public static string querys;
    public static event GuardarExceptionSQLHandler OnGuardarException;
    public static event RecuperarExceptionSQLHandler OnRecuperarException;
```

```
public delegate void ListaModificadaHandler();

50 referencias
public sealed class CentroMedico : ISerializable, IDeserializable<CentroMedico>
{
    6 referencias
    public enum EDuracionTurno
    {
        Corto = 10,
        Mediano = 15,
        Largo = 30
    }

    private static CentroMedico instancia;

    private string nombre;
    private List<Paciente> pacientes;
    private List<Profesional> profesionales;
    private List<Turno> turnos;
    private List<string> listaEspecialidades;
    private DiasDeAtencion diasDeAtencion;
    private EDuracionTurno duracionDeTurnos;

    public event ListaModificadaHandler OnCambioRealizado;
```

Las expresiones lambda en combinación con delegados predefinidos, dentro de los usos que le dí, fue para simplificar código en las búsquedas de turnos:

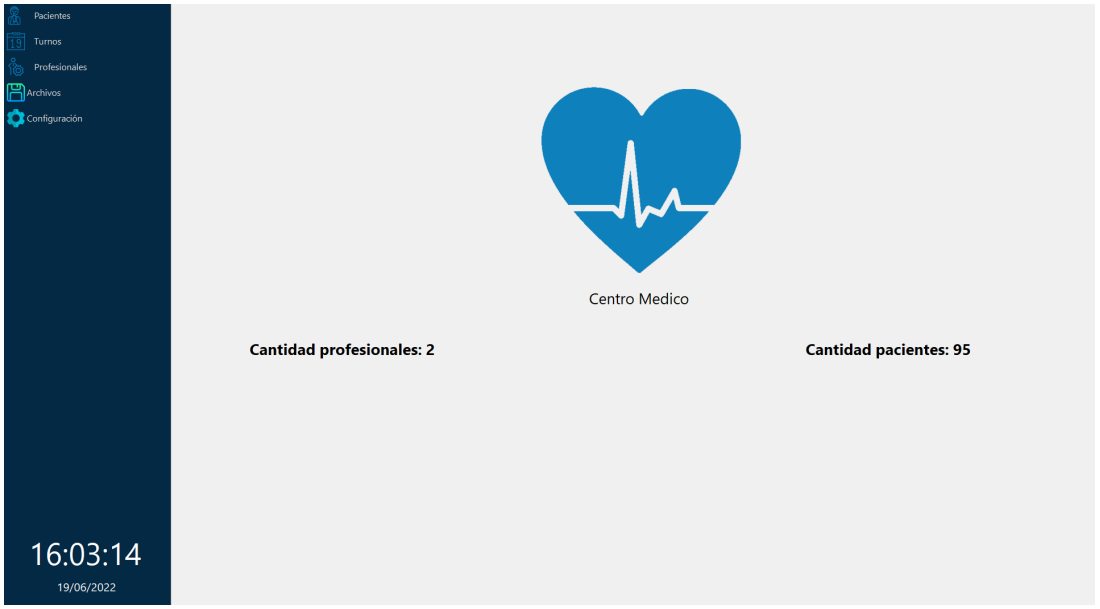
```
329         /// <summary>
330         /// Busca todos los turnos que tienen en comun un profesional y un paciente.
331         /// </summary>
332         /// <param name="profesional"></param>
333         /// <param name="paciente"></param>
334         /// <returns></returns>
335         /// <exception cref="BusquedaException"></exception>
336         public List<Turno> BuscarTurnos(Profesional profesional, Paciente paciente)
337         {
338             try
339             {
340                 return this.Turnos.FindAll(turno => turno.Profesional == profesional && turno.Paciente == paciente);
341             }
342             catch (Exception)
343             {
344                 throw new BusquedaException("Ocurrió un error al buscar los turnos.");
345             }
346         } //Hecho - 1
```

10. Programación MultiHilo y Concurrencia

Al estar pensada la aplicación para que funcione a “pantalla completa”, elegí poner un reloj en el menú lateral a través de la clase Reloj.

También así en el evento Load del FrmCentroSalud, donde a través de un hilo secundario recupero los datos de la base de datos.

Se ha utilizado en diferentes circunstancias la propiedad InvokeRequired para la modificación de controles del formulario.




```
public delegate void NotificarHorarioHandler(Reloj sender);

5 referencias
public class Reloj
{
    private DateTime tiempo;
    public event NotificarHorarioHandler OnNotificarCambio;

    1 referencia
    public void Iniciar()
    {
        Task.Run(() =>
        {
            while (true)
            {
                DateTime now = DateTime.Now;
                Thread.Sleep(100);

                if (now.Second != this.tiempo.Second)
                {
                    if (OnNotificarCambio is not null)
                        this.OnNotificarCambio.Invoke(this);
                }

                tiempo = now;
            }
        });
    }

    2 referencias
    public string ToString(bool fecha)
    {
        if (fecha)
            return $"{tiempo.ToString("d")}";
        else
            return $"{tiempo.ToString("T")}";
    }
}
```

```
public void AsignarTiempo(Reloj reloj)
{
    if (lblReloj.InvokeRequired)
    {
        Action<Reloj> delegado = AsignarTiempo;
        lblReloj.Invoke(delegado, reloj);
    }
    else
    {
        lblReloj.Text = reloj.ToString(false);
        label1.Text = reloj.ToString(true);
    }
}
```

11. Eventos:

Se ha utilizado eventos para poder informar cambios que se realizaron en las listas como pacientes, profesionales, utilizando esta invocación para modificar el menú principal donde se presentan las cantidades de pacientes y profesionales activos, al igual que para hacer cambios en el reloj también se utilizaron para informar posibles excepciones durante la conexión con SQL.

```
/// <summary>
/// Agrega un paciente a la lista de pacientes, siempre y cuando éste no se encuentre en la lista.
/// </summary>
/// <param name="paciente"></param>
/// <returns>True si fue agregado, false si no</returns>
/// <exception cref="Exception"></exception>
4 referencias
public bool AgregarPersona(Persona persona) //Hecho - 3
{
    bool retorno = this + persona;
    this.OnCambioRealizado?.Invoke();
    return retorno;
}
```

```
/// <summary>
/// Recupera el centro medico desde la base de datos.
/// </summary>
1 referencia
private void RecuperarCentroMedicoSQL()
{
    GestorSQL.OnRecuperarException += this.RecuperarCentroMedicoException;
    Task.Run(() => GestorSQL.RecuperarCentroMedico(centroMedico));

}
```

```
1 referencia
public void Iniciar()
{
    Task.Run(() =>
    {
        while (true)
        {
            DateTime now = DateTime.Now;
            Thread.Sleep(100);

            if (now.Second != this.tiempo.Second)
            {
                if (OnNotificarCambio is not null)
                    this.OnNotificarCambio.Invoke(this);
            }

            tiempo = now;
        }
    });
}
```

12. Metodos de extension:

Como bien dije en el punto 3 se utilizaron para eliminar duplicados entre dos listas y retornar una nueva lista siendo T de tipo Persona.

```
namespace Entidades.Extension_Class
{
    0 referencias
    public static class ListPersonasExtendido
    {
        3 referencias
        public static List<T> EliminarCoincidencias<T>(this List<T> lista, List<T> list) where T : Persona
        {
            List<T> nueva = new List<T>();
            nueva.AddRange(lista);
            foreach (T a in lista)
            {
                foreach(T b in list)
                {
                    if (a.Documento == b.Documento)
                    {
                        nueva.Remove(a);
                    }
                }
            }

            return nueva;
        }

    }
}
```

También la utilice para poder poner en mayúscula cada primera letra cuando quería que los nombres o apellidos estén en mayúscula.

```
public static class StringExtendido
{
    /// <summary>
    /// Eleva a mayusculas la primer letra de cada palabra de la cadena de textos recibida.
    /// </summary>
    /// <param name="text"></param>
    /// <returns></returns>
    4 referencias
    public static string TotUpperFirstLetter(this String text)
    {
        string retorno = "";
        char[] chars = text.ToCharArray();
        chars[0] = char.ToUpper(chars[0]);
        retorno += chars[0];

        for (int i = 1; i < text.Length; i++)
        {
            if (chars[i - 1] == ' ')
            {
                chars[i] = char.ToUpper(chars[i]);
            }
            retorno += chars[i];
        }

        return retorno;
    }
}
```