

Documentación de ASTRA

Jose Angel Villa Ramírez

Ámbar Azul Ronquillo López

Iker Samuel Requeses Hernández

Cristian Daniel Campa Padilla

Documentación general del desarrollo de la aplicación de escritorio
ASTRA (Agenda Saludable con Tecnología para la Regulación de
Atención medica).

Índice

Documentación del Proyecto ASTRA.....	4
1. Descripción del Proyecto.....	4
2. Objetivo del Proyecto	4
3. Requerimientos Funcionales (RF).....	4
4. Requerimientos No Funcionales (RNF)	5
5. Tecnologías Utilizadas	5
6. Arquitectura del Sistema	5
7. Diagramas UML	5
8. Modelado de Base de Datos.....	7
9. Indicadores del Proyecto.....	7
10. Indicadores de Desempeño de la Aplicación.....	7
11. Desarrollo Iterativo e Incremental	7
12. Mejora Continua.....	8
13. Evidencias Visuales y Capturas de Pantalla.....	8
Formulario de inicio de sesión.....	12
Código completo.....	12
Diseño.....	15
Iniciar sesión	15
Cancelar	17
Iniciar sesión aquí.....	17
Formulario de registro de usuarios.....	17
Código completo.....	17
Diseño.....	19
Registrar.....	20
Formulario general de los datos de los pacientes.....	21
Código completo.....	21
Diseño.....	26
Metodo de cargar pacientes	26
Agregar paciente	27

Eliminar paciente	28
Agendar y modificar cita	29
Eliminar cita	29
Expediente	30
Formulario agregar paciente	31
Código completo	31
Diseño	33
Agregar paciente	34
Formulario de agendar/modificar una cita	35
Código completo:	35
Diseño	37
Agendar/Modificar cita	37
Formulario de expediente	38
Código completo	38
Diseño	41
Expediente	41
Confirmar expediente	41
Base de datos	42
Localdb	42
Tablas	43

Documentación del Proyecto ASTRA

1. Descripción del Proyecto

El proyecto ASTRA consiste en el desarrollo de una aplicación de escritorio desarrollada con C# y Windows Forms, diseñada para la gestión de citas y pacientes en un centro comunitario. El sistema busca informatizar el proceso de control de pacientes y consultas, permitiendo una administración más eficiente, organizada y segura.

2. Objetivo del Proyecto

El objetivo principal de ASTRA es registrar pacientes y agendar citas médicas. Como objetivo secundario, se incorpora la funcionalidad de redactar y almacenar un expediente clínico correspondiente a cada cita registrada.

3. Requerimientos Funcionales (RF)

Código	Requerimiento Funcional	Descripción
RF-01	Registro de citas médicas	Crear nuevas citas con fecha, hora, médico y paciente.
RF-02	Visualización de agenda	Mostrar agenda diaria/semanal filtrable.
RF-03	Gestión de pacientes	Registrar, editar y consultar información de pacientes.
RF-04	Modificación/Cancelación de citas	Editar o cancelar citas con confirmación.
RF-05	Control de solapamiento	Validar que no haya citas en conflicto.
RF-06	Notificaciones	Alertas automáticas para próximas citas.
RF-07	Búsqueda de pacientes	Filtrado por nombre, ID o atributos.
RF-08	Autenticación segura	Ingreso con usuario y contraseña, con roles.
RF-09	Registro automático de fecha/hora	Guardado automático de timestamps.

4. Requerimientos No Funcionales (RNF)

Código Requerimiento No Funcional Descripción Detallada

RNF-01	Interfaz intuitiva	UX/UI sencillo, formularios cortos.
RNF-02	Rendimiento	Tiempo de respuesta < 2s.
RNF-03	Multiplataforma	Compatible con escritorio, web, móvil.
RNF-04	Funcionalidad offline	Agendar citas sin conexión y sincronizar luego.
RNF-05	Seguridad	Cifrado AES-256, autenticación JWT.
RNF-06	Compatibilidad	Ejecutable en equipos con recursos limitados.
RNF-07	Diseño profesional	Estética médica, tipografía clara.
RNF-08	Escalabilidad	Arquitectura modular.

5. Tecnologías Utilizadas

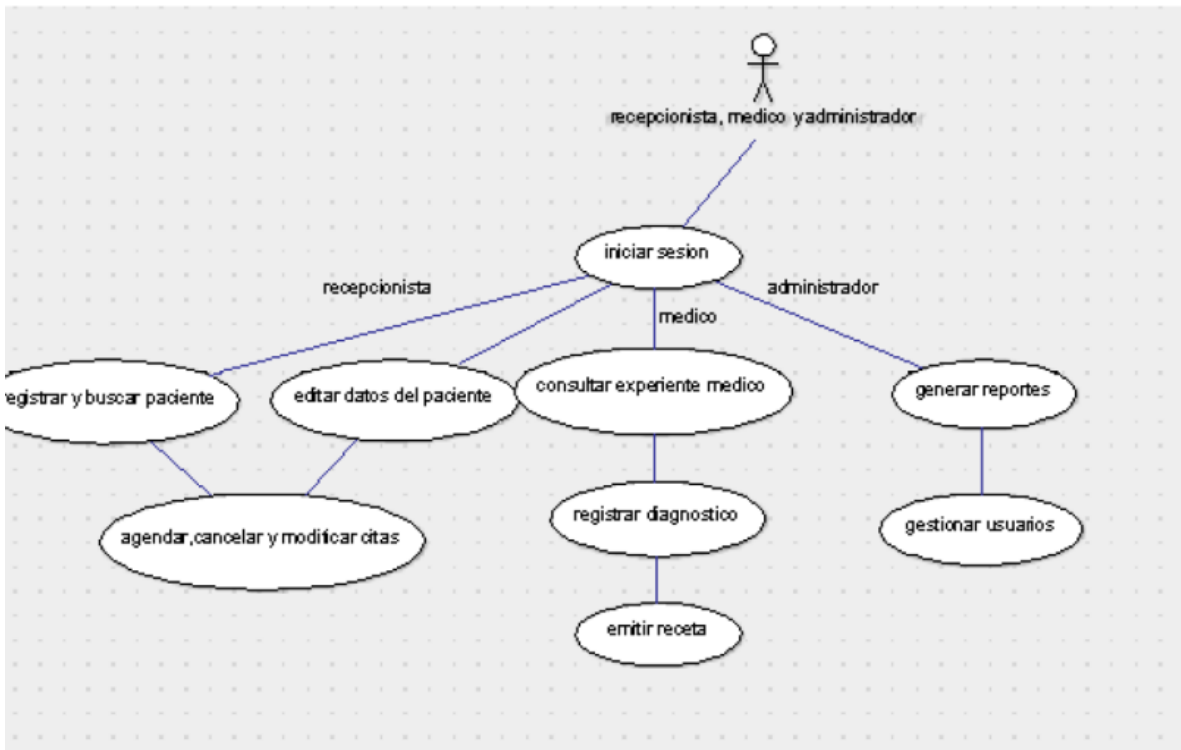
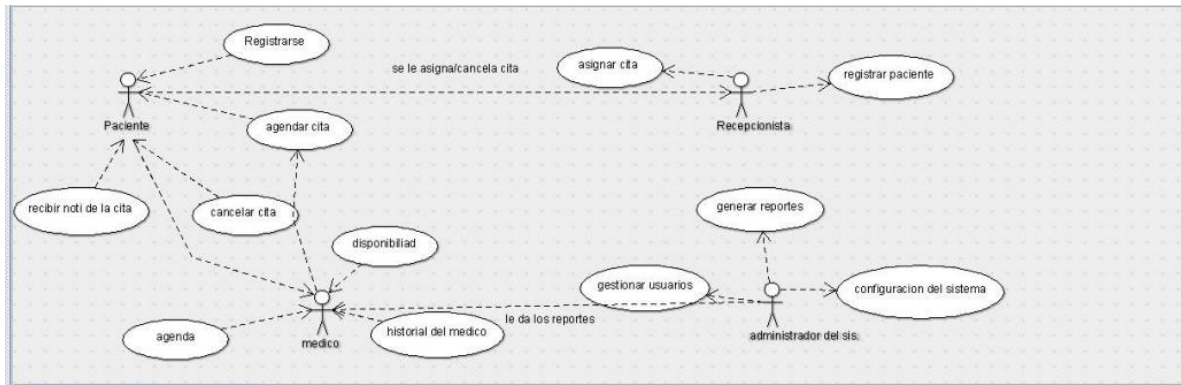
- Lenguaje: C#
 - Entorno: Windows Forms
 - Base de Datos: MySQL (LocalDB)
-

6. Arquitectura del Sistema

Se realizaron los ajustes de la aplicación para que sea funcional en arquitecturas x32 y x64 bits respectivamente.

7. Diagramas UML

Estos diagramas son meramente ilustrativos y pueden tener variaciones con respecto al producto final.



8. Modelado de Base de Datos



9. Indicadores del Proyecto

Tiempo de desarrollo: 3 meses

Costo estimado:+

Costos estimados por persona, no representan los costos totales del desarrollo.

- Equipo de cómputo: \$4000 MXN
- Licencias: \$1200 MXN
- Otros gastos: \$400 MXN (transporte, energía, internet, oficinas)

Alcance:

- Prototipo monousuario
- Base de datos local integrada
- Cumplimiento del objetivo de agendamiento de citas

10. Indicadores de Desempeño de la Aplicación

Espacio para insertar tablas de indicadores cualitativos y cuantitativos (ya proporcionados en imagen).

11. Desarrollo Iterativo e Incremental

El proyecto se desarrolló en fases:

1. Inicio de sesión y registro de usuarios

2. Diseño de interfaz principal con DataGridView para pacientes
3. Ventanas independientes para registro de pacientes y citas
4. Integración del expediente por paciente
5. Módulos de eliminación y edición

Cada iteración fue evaluada y validada antes de continuar con la siguiente, lo que permitió implementar mejoras constantes sobre el prototipo inicial.

12. Mejora Continua

Se implementó una metodología de retroalimentación continua:

- Validación funcional tras cada módulo
 - Ajustes según experiencia de usuario
 - Revisiones internas de código
 - Priorización de funcionalidades clave antes de implementar extras
-

13. Evidencias Visuales y Capturas de Pantalla



The screenshot shows a login window titled "Inicio de sesión". It has a dark blue background. There are two input fields: "Usuario" with the text "admin" and "Contraseña" with masked text "*****". Below the fields, there is a link "¿No está registrado?" and a link "Inicie sesión aquí". At the bottom, there are two buttons: "Iniciar sesión" and "Cancelar".

Registro

Usuario

admin

Contraseña

Registrar

Nuevo paciente


Paciente

Nombre	Apellidos	Edad
Angel	Villa	19
Altura	Peso	
1.65	70	
Alergias medicinales	Padecimientos	
Ninguna	ninguna	




Pacientes


	IdPaciente	Nombre	Apellido	Edad	Altura	Peso	Alergia	Padecimiento	Proxima_cita
▶	1	Angel	Villa	19	1.65	70	Ninguna	ninguna	
*									




Agregar Paciente




Eliminar Paciente




Agendar Cita



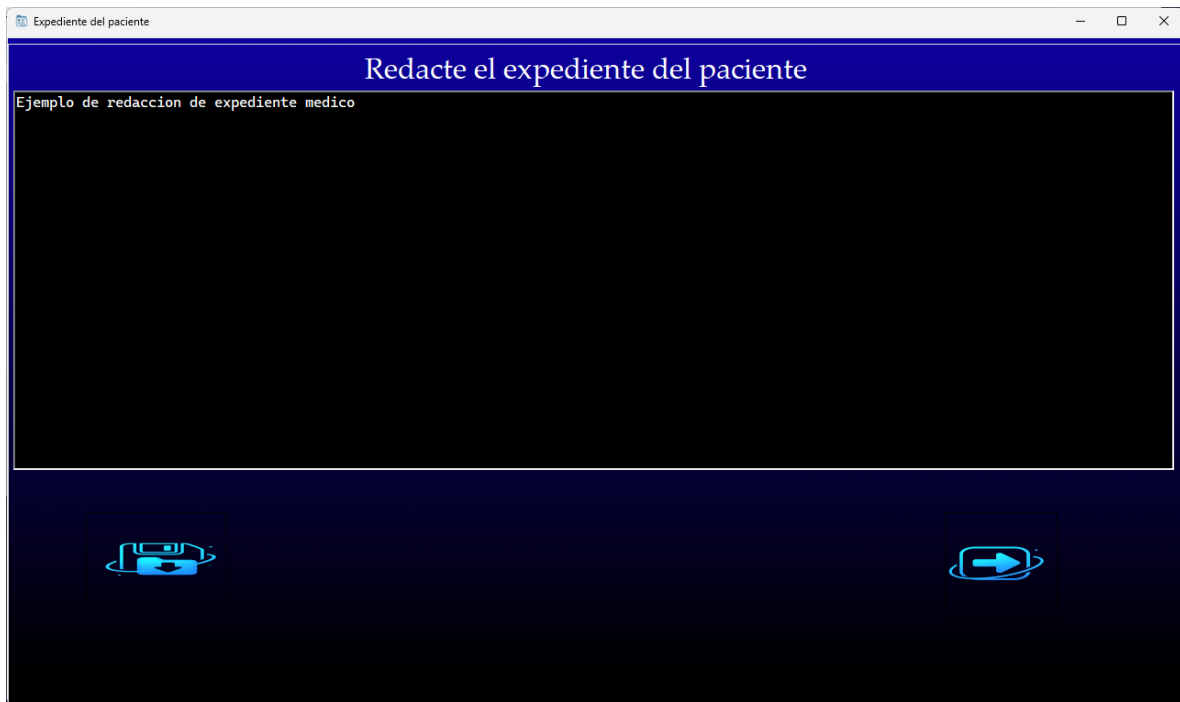
Modificar Cita



Eliminar Cita



Acceder al Expediente



Formulario de inicio de sesión

Código completo

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.Data.SqlClient;
using System.IO;
namespace Astra
{
    public partial class Form1 : Form
    {
        string rutaDb;
        string cadena_conexion;
        public Form1()
        {
            InitializeComponent();

            //Cadena de conexión para SQLite

            string ruta = Path.Combine(Application.StartupPath,
@"Data\AstraDB.mdf");
```

```

        cadena_conexion = "Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\AstraDB.mdf;
Integrated Security=True;Connect Timeout=30";

    }
    private void btnIniciarSesion_Click(object sender, EventArgs e)
    {
        //Creamos las variables para traer el texto de los textbox
        string usuario = txtUsuario.Text;
        string contraseña = txtContraseña.Text;

        using (SqlConnection con = new SqlConnection(cadena_conexion))
        {
            try // Es fundamental tener un try-catch que envuelva la
apertura de la conexión y las operaciones
            {
                con.Open();

                // Si tu tabla 'Usuarios' es la misma que 'usuarios' en
el ejemplo de Form4,
                // la capitalización podría importar dependiendo de la
configuración de la DB,
                // pero generalmente SQLite no distingue
mayúsculas/minúsculas para nombres de tablas/columnas.

                // --- CORRECCIÓN DE LA CONSULTA Y LOS PARÁMETROS ---
                // 1. Uso de parámetros nombrados (@usuario, @contraseña)
- MUY RECOMENDADO
                string consulta = "SELECT COUNT(*) FROM Usuarios WHERE
Usuario = @Usuario AND Contraseña = @Contraseña";

                // Corrección: SqlCommand -> SQLiteCommand
                using (SqlCommand comando = new SqlCommand(consulta,
con))
                {
                    // 2. Añadir parámetros por nombre, esto es más
robusto y claro.
                    comando.Parameters.AddWithValue("@Usuario", usuario);
                    // El nombre del parámetro debe coincidir con la consulta
                    comando.Parameters.AddWithValue("@Contraseña",
contraseña);

                    // --- CORRECCIÓN EN LA CONVERSIÓN DE EXECUTESCALAR -
--
                    // ExecuteScalar devuelve un 'object'. Para COUNT(*),
será un 'long' (0 o más).
                    // Es más seguro usar Convert.ToInt32() o un cast a
'long' primero.
                    object resultado = comando.ExecuteScalar();
                    int cuenta = 0;

                    if (resultado != null && resultado != DBNull.Value)
                    {
                        cuenta = Convert.ToInt32(resultado); // Convierte
el resultado a int de forma segura

```

```

        }
        // Si resultado es null o DBNull.Value (lo cual no
        debería ocurrir con COUNT(*)), cuenta seguirá siendo 0.

        if (cuenta > 0)
        {
            MessageBox.Show("Inicio de sesion exitoso");

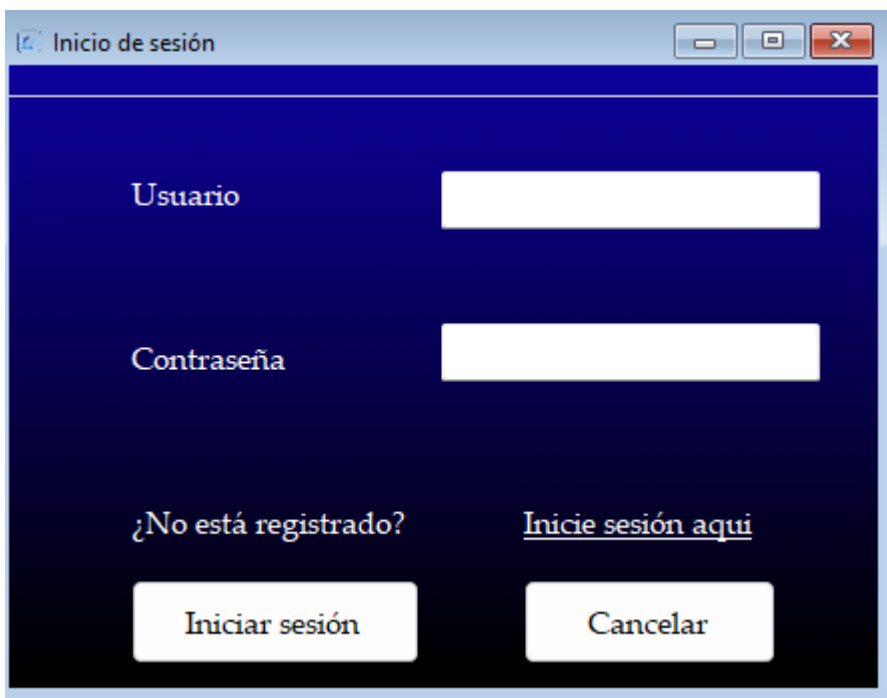
            Form3 = new Form3();
            form3.Show();
            this.Hide();
        }
        else
        {
            MessageBox.Show("Usuario y/o contraseña
incorrectos."); // Mensaje más preciso
        }
    }
    catch (SqlException ex) // Captura errores específicos de
SQLite
    {
        MessageBox.Show($"Error de base de datos: {ex.Message}",
"Error de Conexión", MessageBoxButtons.OK, MessageBoxIcon.Error);
        // Opcional: Debug.WriteLine(ex.ToString()); para ver más
detalles en la salida de depuración
    }
    catch (Exception ex) // Captura cualquier otro error
inesperado
    {
        MessageBox.Show($"Ocurrió un error inesperado:
{ex.Message}", "Error General", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    // El 'using' se encarga de cerrar la conexión
automáticamente al salir del bloque.
}

private void linkLabel1_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    Form2 = new Form2();
    form2.Show();
}

private void btnCancelar_Click(object sender, EventArgs e)
{
    txtUsuario.Clear();
    txtContraseña.Clear();
}
}
}

```

Diseño



The image shows a Windows-style window titled "Inicio de sesión" (Login). The window has a dark blue background. It contains two text input fields: one labeled "Usuario" (User) and one labeled "Contraseña" (Password). Below the password field, there is a link that says "¿No está registrado? Inicie sesión aquí". At the bottom of the window, there are two buttons: "Iniciar sesión" (Login) and "Cancelar" (Cancel).

Iniciar sesión

Configuramos el evento de click en el botón de iniciar previamente ya tenemos establecida una ruta y una cadena de conexión a nuestra base de datos sql, por lo que el evento de click manda a llamar a los datos de los campos de texto pertenecientes al usuario y a la contraseña para así realizar una búsqueda a través de la tabla, siguiendo la lógica si retorna un valor menor a 0, es decir negativo entonces quiere decir que no se encontró de ninguno modo registrado nuestro usuario y muestra un mensaje diciendo que el usuario y/o la contraseña son incorrectos, en el caso de retornar un valor mayor a 0 le permite al usuario entrar al menú principal.

```
private void btnIniciarSesion_Click(object sender, EventArgs e)
{
    //Creamos las variables para traer el texto de los textbox
    string usuario = txtUsuario.Text;
    string contraseña = txtContraseña.Text;

    using (SqlConnection con = new SqlConnection(cadena_conexion))
    {
        try // Es fundamental tener un try-catch que envuelva la apertura de
        la conexión y las operaciones
        {
            con.Open();

            // Si tu tabla 'Usuarios' es la misma que 'usuarios' en el
            ejemplo de Form4,
```

```

        // la capitalización podría importar dependiendo de la
        configuración de la DB,
        // pero generalmente SQLite no distingue mayúsculas/minúsculas
        para nombres de tablas/columnas.

        // --- CORRECCIÓN DE LA CONSULTA Y LOS PARÁMETROS ---
        // 1. Uso de parámetros nombrados (@usuario, @contraseña) - MUY
        RECOMENDADO
        string consulta = "SELECT COUNT(*) FROM Usuarios WHERE Usuario =
        @Usuario AND Contraseña = @Contraseña";

        using (SqlCommand comando = new SqlCommand(consulta, con))
        {
            // 2. Añadir parámetros por nombre, esto es más robusto y
            claro.
            comando.Parameters.AddWithValue("@Usuario", usuario); // El
            nombre del parámetro debe coincidir con la consulta
            comando.Parameters.AddWithValue("@Contraseña", contraseña);

            // --- CORRECCIÓN EN LA CONVERSIÓN DE EXECUTESCALAR ---
            // ExecuteScalar devuelve un 'object'. Para COUNT(*), será
            un 'long' (0 o más).
            // Es más seguro usar Convert.ToInt32() o un cast a 'long'
            primero.

            object resultado = comando.ExecuteScalar();
            int cuenta = 0;

            if (resultado != null && resultado != DBNull.Value)
            {
                cuenta = Convert.ToInt32(resultado); // Convierte el
                resultado a int de forma segura
            }
            // Si resultado es null o DBNull.Value (lo cual no debería
            ocurrir con COUNT(*)), cuenta seguirá siendo 0.

            if (cuenta > 0)
            {
                MessageBox.Show("Inicio de sesion exitoso");

                Form3 form3 = new Form3();
                form3.Show();
                this.Hide();
            }
            else
            {
                MessageBox.Show("Usuario y/o contraseña incorrectos.");
            }
        }
        // Mensaje más preciso
    }

    catch (SqlException ex) // Captura errores específicos de SQLite
    {
        MessageBox.Show($"Error de base de datos: {ex.Message}", "Error
        de Conexión", MessageBoxButtons.OK, MessageBoxIcon.Error);
        // Opcional: Debug.WriteLine(ex.ToString()); para ver más
        detalles en la salida de depuración
    }
}

```



```

        catch (Exception ex) // Captura cualquier otro error inesperado
        {
            MessageBox.Show($"Ocurrió un error inesperado: {ex.Message}",
"Error General", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        // El 'using' se encarga de cerrar la conexión automáticamente al
        salir del bloque.
    }
}

```

Cancelar

El evento de click para el botón cancelar es mas sencillo pues no realiza ningún ajuste directo en la base de datos ya que solo limpia usando la palabra reservada clear los campos de texto.

```

private void btnCancelar_Click(object sender, EventArgs e)
{
    txtUsuario.Clear();
    txtContraseña.Clear();
}

```

Iniciar sesión aquí

Este label de enlace permite visualizar un formulario para que el usuario registre un usuario y contraseña para acceder a la aplicación, haciendo uso de un insert a la base de datos y de buscar que no se relacionen los valores de los campos con uno ya existente permite registrar exitosamente al usuario.

```

private void linkLabel1_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    Form2 form2 = new Form2();
    form2.Show();
}

```

Formulario de registro de usuarios

Código completo

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Security.Cryptography;

```

```

using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.Data.SqlClient;

namespace Astra
{
    public partial class Form2 : Form
    {
        string ruta;

        string cadena_conexion;

        public Form2()
        {
            InitializeComponent();

            string ruta = Path.Combine(Application.StartupPath,
@"Data\AstraDB.mdf");
            cadena_conexion = "Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\AstraDB.mdf;
Integrated Security=True;Connect Timeout=30";

        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            string usuario = txtRegistroUsuario.Text;
            string contraseña = txtRegistroContraseña.Text;

            if (usuario == "" || contraseña == "")
            {
                MessageBox.Show("Por favor ingrese todos los datos");
                return;
            }

            using (SqlConnection con = new SqlConnection(cadena_conexion ))
            {
                try
                {
                    con.Open();
                    string consulta = "SELECT COUNT(*) FROM Usuarios WHERE
Usuario = @Usuario";
                    SqlCommand verificar = new SqlCommand(consulta, con);
                    verificar.Parameters.AddWithValue("@Usuario", usuario);
                    int existe = (int)verificar.ExecuteScalar();
                    if (existe > 0)
                    {
                        MessageBox.Show("El usuario ya esta registrado");
                        return;
                    }
                }
            }
        }
    }
}

```

```

    }
    //Insertar el nuevo usuario
    string insertar = "INSERT INTO Usuarios (Usuario,
Contraseña) VALUES (@Usuario,@Contraseña)";
    SqlCommand cmd = new SqlCommand(insertar,con);
    cmd.Parameters.AddWithValue("@Usuario", usuario);
    cmd.Parameters.AddWithValue("@Contraseña", contraseña);
    cmd.ExecuteNonQuery();

    MessageBox.Show("Usuario registrado con éxito");
    this.Close();
}
catch(Exception ex)
{
    MessageBox.Show("Error al registrar " + ex.Message);
}
}
}
}
}

```

Diseño

The image shows a Windows application window titled "Registro". The window has a standard Windows title bar with minimize, maximize, and close buttons. The main area of the window has a dark blue background. In the center, there are two white text input fields. The first field is labeled "Usuario" and the second field is labeled "Contraseña". Below these fields is a white button labeled "Registrar".

Registrar

Al igual que al iniciar sesión se toman los datos de las cajas de texto pertenecientes a usuario y contraseña de tal manera que se vuelve a realizar un insert en la en nuestra base de datos detectando que no exista un valor registrado que coincida con lo ingresado por el usuario. De manera que se hace un select que retorna la cantidad de usuarios registrados en nuestra base de datos y haciendo uso de nuestro sql command verificar va a ir retomando cada valor de la columna, es decir si buscamos un usuario “admin” va retomando cada valor registrado hasta que coincida con nuestro valor ingresado en este ejemplo es “admin” si coincide retorna un valor de 1 y si no existe retorna 0 entonces si existe y es mayor a 0 mostrara un mensaje que ya esta registrado, en el caso de que no se realiza un insert a la base de datos para agregar el nuevo valor.

```
private void btnRegistrar_Click(object sender, EventArgs e)
{
    string usuario = txtRegistroUsuario.Text;
    string contraseña = txtRegistroContraseña.Text;

    if (usuario == "" || contraseña == "")
    {
        MessageBox.Show("Por favor ingrese todos los datos");
        return;
    }

    using (SqlConnection con = new SqlConnection(cadena_conexion ))
    {
        try
        {
            con.Open();
            string consulta = "SELECT COUNT(*) FROM Usuarios WHERE Usuario = @Usuario";
            SqlCommand verificar = new SqlCommand(consulta, con);
            verificar.Parameters.AddWithValue("@Usuario", usuario);
            int existe = (int)verificar.ExecuteScalar();
            if (existe > 0)
            {
                MessageBox.Show("El usuario ya esta registrado");
                return;
            }
            //Insertar el nuevo usuario
            string insertar = "INSERT INTO Usuarios (Usuario, Contraseña) VALUES (@Usuario,@Contraseña)";
            SqlCommand cmd = new SqlCommand(insertar, con);
            cmd.Parameters.AddWithValue("@Usuario", usuario);
            cmd.Parameters.AddWithValue("@Contraseña", contraseña);
            cmd.ExecuteNonQuery();

            MessageBox.Show("Usuario registrado con exito");
            this.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error al registrar " + ex.Message);
        }
    }
}
```

```
}  
}  
}
```

Formulario general de los datos de los pacientes

Código completo

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using Microsoft.Data.SqlClient;  
using System.IO;  
  
namespace Astra  
{  
    public partial class Form3 : Form  
    {  
  
        string ruta;  
        string cadena_conexion;  
        public Form3()  
        {  
            InitializeComponent();  
            //Direcciones de la base de datos  
  
            string ruta = Path.Combine(Application.StartupPath,  
@"Data\AstraDB.mdf");  
            cadena_conexion = "Data  
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\AstraDB.mdf;  
Integrated Security=True;Connect Timeout=30";  
  
        }  
        //Metodo para cargar pacientes en el datagridview  
        private void CargarPacientes()  
        {  
            //La base de datos conexion apunta al archivo local de la base de  
datos de sqlclient  
  
            // Conexion y uso de la base de datos  
  
            using (SqlConnection con = new SqlConnection(cadena_conexion))  
            {  
                //Probamos la conexion y si se abre la base de datos  
                try  
                {  
                    con.Open();  
                    //Seleccionamos los pacientes y creamos un adaptador
```

```

        //El adaptador al momento de crear una tabla lo que hace
es rellenar esta tabla con los datos actualizados del datagrid
        string consulta = "SELECT * FROM Pacientes";

        // --- INICIO DE LA CORRECCIÓN SOLICITADA ---
        // Cambiado: SqlDataReader (no se instancia
directamente)
        // Cambiado: adaptador.Fill(tabla) (no existe, se usa
tabla.Load(reader))

        // Creamos un comando SQLite para ejecutar la consulta
using (SqlCommand comando = new SqlCommand(consulta,
con))
    {
        // Ejecutamos el comando y obtenemos un lector de
datos
        using (SqlDataReader reader =
comando.ExecuteReader())
        {
            DataTable tabla = new DataTable();
            // Llenamos el DataTable con los datos del lector
            tabla.Load(reader); // Esta es la función
equivalente a 'Fill' para DataTable con un DataReader

            dgvPacientes.DataSource = tabla;
        } // El DataReader se cerrará automáticamente aquí
    } // El comando se liberará automáticamente aquí
    // --- FIN DE LA CORRECCIÓN SOLICITADA ---

}
catch (SqlException ex) // Cambiado: Exception ->
SQLiteException para manejo más específico
{
    MessageBox.Show("Error al cargar pacientes: " +
ex.Message); // Mensaje mejorado
}
catch (Exception ex) // Para capturar cualquier otro tipo de
excepción no relacionada con SQLite
{
    MessageBox.Show("Error inesperado al cargar pacientes: "
+ ex.Message);
}

} // La conexión se cerrará automáticamente aquí gracias al
'using'
}
//Al momento de abrir el formulario se manda a llamar al metodo de
carga pacientes
private void Form3_Load(object sender, EventArgs e)
{
    CargarPacientes();
}
//El boton de agregar pacientes
private void btnAgregarPaciente_Click(object sender, EventArgs e)
{
    Form4 = new Form4();
    //Suscribirse al evento
    form4.PacienteAgregado += () =>

```

```

        {
            CargarPacientes();
        };
        form4.ShowDialog();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        if(dgvPacientes.SelectedRows.Count == 0)
        {
            MessageBox.Show("Por favor seleccione un paciente a
eliminar");
            return;
        }
        DialogResult confirmacion = MessageBox.Show("¿Esta seguro de que
desea eliminar al paciente? ", "Confirmar eliminacion ",
MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

        if (confirmacion == DialogResult.No)
            return;

        int idpaciente = Convert.ToInt32
            (dgvPacientes.SelectedRows[0].Cells["IdPaciente"].Value);

        using (SqlConnection conn = new SqlConnection(cadena_conexion))
        {
            try
            {
                conn.Open();
                string eliminar = "DELETE FROM Pacientes WHERE IdPaciente
= @IdPaciente";
                SqlCommand cmd = new SqlCommand(eliminar,conn);
                cmd.Parameters.AddWithValue("@IdPaciente", idpaciente);
                cmd.ExecuteNonQuery();
                MessageBox.Show("Paciente eliminado correctamente");
            }
            catch (Exception ex)
            {
                MessageBox.Show("No se pudo eliminar el paciente " +
ex.Message);
            }
        }
        CargarPacientes();
    }

    private void btnAgendar_Click(object sender, EventArgs e)
    {
        if (dgvPacientes.SelectedRows.Count == 0)
        {
            MessageBox.Show("Seleccione un paciente para agendar su cita
por favor");
        }
    }

```

```

        return;
    }
    int id =
int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());
    Form5 = new Form5(id);
    form5.CitaAgregada += () =>
    {
        CargarPacientes();
    };

    form5.ShowDialog();
}

private void btnModificar_Click(object sender, EventArgs e)
{
    if (dgvPacientes.SelectedRows.Count == 0)
    {
        MessageBox.Show("Seleccione un paciente para agendar su cita
por favor");
        return;
    }
    int id =
int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());
    Form5 = new Form5(id);
    form5.CitaAgregada += () =>
    {
        CargarPacientes();
    };

    form5.ShowDialog();
}

//Eliminar cita
private void btnEliminar_Click(object sender, EventArgs e)
{
    //Variables para obtener los valores de las celdas de acuerdo a su
tipo de dato: Fecha y numerico
    DateTime cita;

    object valor =
dgvPacientes.CurrentRow.Cells["Proxima_cita"].Value;

    if (string.IsNullOrEmpty(valor.ToString()))
    {
        MessageBox.Show("No hay cita agendada, error al eliminar");
    }
    else
    {
        cita =
DateTime.Parse(dgvPacientes.CurrentRow.Cells["Proxima_cita"].Value.ToString()
);
    }

    int idpaciente =
int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());

```



```

        //Variable de nuestra ruta de datos

        using (SqlConnection con = new SqlConnection(cadena_conexion))
        {
            try
            {
                con.Open();
                //Actualizacion de la base de datos
                string update = "UPDATE Pacientes SET Proxima_cita = NULL
WHERE IdPaciente = @IdPaciente";
                //Comando para agregar valores y actualizar
                SqlCommand cmd = new SqlCommand(update, con);

                cmd.Parameters.AddWithValue("@IdPaciente", idpaciente);

                cmd.ExecuteNonQuery();

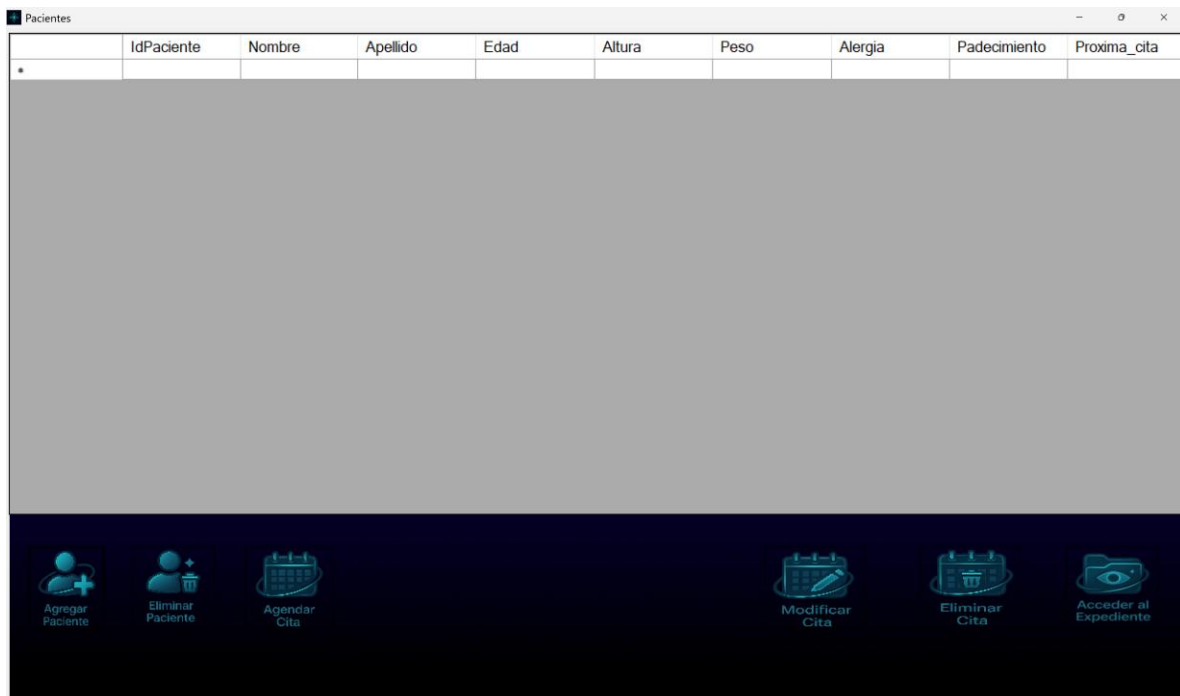
            }
            catch (Exception ex)
            {
                MessageBox.Show("Conexion fallida " + ex.Message);
            }
        }
        CargarPacientes();
    }

    private void btnExpediente_Click(object sender, EventArgs e)
    {
        int id =
int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());
        if(id == 0)
        {
            MessageBox.Show("Seleccione un paciente para ver su
expediente");
            return;
        }
        Form6 = new Form6(id);

        form6.ShowDialog();
    }
}
}

```

Diseño



Metodo de cargar pacientes

Al momento de abrir el formulario existe un evento de carga que manda a llamar a este método de carga de pacientes, lo que hace este método es escribir las columnas de nuestro datagridview del mismo modo en el que se encuentran en la tabla de la que vamos a proceder a realizar un select y vamos a usar un datareader, entonces primero se establece la conexión a la base de datos y obtenemos una consulta de los datos registrados a través de un select, después de ejecutar la consulta creamos y ejecutamos un datareader (lector de datos) y también vamos a crear una datatable, este funciona como una pequeña tabla y el usando el metodo loadreader se carga toda la información leída por datareader a la tabla y posteriormente esto mismo se vuelve a cargar esta vez en el datagridview.

```
private void CargarPacientes()
{
    //La base de datos conexion apunta al archivo local de la base de datos
    de sqlclient

    // Conexion y uso de la base de datos

    using (SqlConnection con = new SqlConnection(cadena_conexion))
    { //Probamos la conexion y si se abre la base de datos
        try
        {
            con.Open();
            //Seleccionamos los pacientes y creamos un adaptador
            //El adaptador al momento de crear una tabla lo que hace es
            rellenar esta tabla con los datos actualizados del datagrid
            string consulta = "SELECT * FROM Pacientes";
```

```

        // Cambiado: SQLiteDataReader (no se instancia directamente)
        // Cambiado: adaptador.Fill(tabla) (no existe, se usa
tabla.Load(reader))

        // Creamos un comando SQL para ejecutar la consulta
        using (SqlCommand comando = new SqlCommand(consulta, con))
        {
            // Ejecutamos el comando y obtenemos un lector de datos
            using (SqlDataReader reader = comando.ExecuteReader())
            {
                DataTable tabla = new DataTable();
                // Llenamos el DataTable con los datos del lector
                tabla.Load(reader); // Esta es la función equivalente a
'Fill' para DataTable con un DataReader

                dgvPacientes.DataSource = tabla;
            } // El DataReader se cerrará automáticamente aquí
        } // El comando se liberará automáticamente aquí

    }
    catch (SQLException ex) // Cambiado: Exception -> SQLiteException
para manejo más específico
    {
        MessageBox.Show("Error al cargar pacientes: " + ex.Message); //
Mensaje mejorado
    }
    catch (Exception ex) // Para capturar cualquier otro tipo de
excepción no relacionada con SQLite
    {
        MessageBox.Show("Error inesperado al cargar pacientes: " +
ex.Message);
    }

    } // La conexión se cerrará automáticamente aquí gracias al 'using'
}

```

Agregar paciente

Simplemente mandamos a llamar a nuestro formulario de agregar paciente pero hay que hacer unos ajustes importantes, dentro de nuestro formulario de agregar pacientes nos vamos a encontrar con un evento llamado “PacienteAgregado” por lo que es muy importante hacer uso de la suscripción del evento de paciente agregar desde este formulario. Y utilizamos showdialog para retornar hacia este formulario los datos de las variables que encontremos en el formulario de agregar paciente.

```

private void btnAgregarPaciente_Click(object sender, EventArgs e)
{
    Form4 = new Form4();
    //Suscribirse al evento
    form4.PacienteAgregado += () =>
    {
        CargarPacientes();
    };
}

```

```
form4.ShowDialog();  
}
```

Eliminar paciente

Para eliminar un paciente se hace directo del formulario principal entonces no es necesario llamar a otro formulario, simplemente se selecciona el paciente a eliminar y se extrae el valor de la celda de id del paciente, en este caso al extraer un valor se hace de tipo de string entonces se vuelve a convertir a tipo int, una vez hecho esto abrimos la conexión a la base de datos y ejecutamos un delete desde la tabla de usuarios Si el id de paciente es igual al id de paciente que extraimos y convertimos a tipo int y finalmente volvemos a llamar al método para recargar la información de los pacientes que aun quedan registrados .

```
private void button2_Click(object sender, EventArgs e)
{
    if(dgvPacientes.SelectedRows.Count == 0)
    {
        MessageBox.Show("Por favor seleccione un paciente a eliminar");
        return;
    }
    DialogResult confirmacion = MessageBox.Show("¿Esta seguro de que desea eliminar al paciente? ", "Confirmar eliminacion ", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

    if (confirmacion == DialogResult.No)
        return;

    int idpaciente = Convert.ToInt32
        (dgvPacientes.SelectedRows[0].Cells["IdPaciente"].Value);

    using (SqlConnection conn = new SqlConnection(cadena_conexion))
    {
        try
        {
            conn.Open();
            string eliminar = "DELETE FROM Pacientes WHERE IdPaciente = @IdPaciente";
            SqlCommand cmd = new SqlCommand(eliminar, conn);
            cmd.Parameters.AddWithValue("@IdPaciente", idpaciente);
            cmd.ExecuteNonQuery();
            MessageBox.Show("Paciente eliminado correctamente");
        }
        catch (Exception ex)
        {
            MessageBox.Show("No se pudo eliminar el paciente " + ex.Message);
        }
    }
    CargarPacientes();
}
```

Agendar y modificar cita

Estos 2 formularios siguen el mismo principio lógico del formulario que agrega paciente debemos seleccionar un paciente, conseguir el valor de la celda de id y posteriormente mandamos a llamar al formulario de agenda y modificación de cita incorporando el valor extraído de id y suscribimos al evento de agregarcita.

```
private void btnAgendar_Click(object sender, EventArgs e)
{
    if (dgvPacientes.SelectedRows.Count == 0)
    {
        MessageBox.Show("Seleccione un paciente para agendar su cita por favor");
        return;
    }
    int id =
int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());
    Form5 form5 = new Form5(id);
    form5.CitaAgregada += () =>
    {
        CargarPacientes();
    };

    form5.ShowDialog();
}

private void btnModificar_Click(object sender, EventArgs e)
{
    if (dgvPacientes.SelectedRows.Count == 0)
    {
        MessageBox.Show("Seleccione un paciente para agendar su cita por favor");
        return;
    }
    int id =
int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());
    Form5 form5 = new Form5(id);
    form5.CitaAgregada += () =>
    {
        CargarPacientes();
    };

    form5.ShowDialog();
}
```

Eliminar cita

Comenzamos obteniendo un valor de la celda de próxima cita, si no existe un valor entonces mandamos un mensaje de que no tiene una cita, y si existe mandamos a extraer ese valor para castearlo a tipo int y asignarlo a una variable, entramos nuevamente a la base

de datos y ejecutamos un update para actualizar el valor de la cita directamente en la columna de próxima cita como un valor null y finalmente mostramos un mensaje de confirmación y volvemos a llamar al evento de cargar paciente.

```
private void btnEliminar_Click(object sender, EventArgs e)
{
    //Variables para obtener los valores de las celdas de acuerdo a su tipo de
    dato: Fecha y numerico

    object valor = dgvPacientes.CurrentRow.Cells["Proxima_cita"].Value;

    if (string.IsNullOrEmpty(valor.ToString()))
    {
        MessageBox.Show("No hay cita agendada, error al eliminar");
    }

    int idpaciente =
    int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());
    //Variable de nuestra ruta de datos

    using (SqlConnection con = new SqlConnection(cadena_conexion))
    {
        try
        {
            con.Open();
            //Actualizacion de la base de datos
            string update = "UPDATE Pacientes SET Proxima_cita = NULL WHERE
            IdPaciente = @IdPaciente";
            //Comando para agregar valores y actualizar
            SqlCommand cmd = new SqlCommand(update, con);

            cmd.Parameters.AddWithValue("@IdPaciente", idpaciente);

            cmd.ExecuteNonQuery();
            MessageBox.Show("Cita eliminada correctamente " );
        }
        catch (Exception ex)
        {
            MessageBox.Show("Conexion fallida " + ex.Message);
        }
    }
    CargarPacientes();
}
```

Expediente

El uso de este botón es igual al de agregar paciente en el sentido de que solo manda a llamar un formulario nuevo para redactar nuestro expediente haciendo uso del valor del id del paciente del que queremos redactar o revisar un expediente.

```

private void btnExpediente_Click(object sender, EventArgs e)
{
    int id =
int.Parse(dgvPacientes.CurrentRow.Cells["IdPaciente"].Value.ToString());
    if(id == 0)
    {
        MessageBox.Show("Seleccione un paciente para ver su expediente");
        return;
    }
    Form6 = new Form6(id);
    form6.ShowDialog();
}

```

Formulario agregar paciente

Código completo

```

using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Security.Policy;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Astra
{
    public partial class Form4 : Form
    {
        string cadena_conexion;

        public event Action PacienteAgregado; // Evento personalizado

        public class Paciente
        {
            public string Nombre { get; set; }
            public string Apellidos { get; set; }
            public int Edad { get; set; }
            public double Altura { get; set; }
            public double Peso { get; set; }
            public string Alergias { get; set; }
            public string Padecimientos { get; set; }
        }

        public Form4()
        {
            InitializeComponent();

```

```

        string ruta = Path.Combine(Application.StartupPath,
@"Data\AstraDB.mdf");
        cadena_conexion = "Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\AstraDB.mdf;
Integrated Security=True;Connect Timeout=30";

    }
    private void btnAgregar_Click_1(object sender, EventArgs e)
    {
        //Obtener variables y datos en las variables
        Paciente = new Paciente();
        paciente.Nombre = txtNombre.Text;
        paciente.Apellidos = txtApellidos.Text;
        paciente.Edad = int.Parse(txtEdad.Text);
        paciente.Altura = double.Parse(txtAltura.Text);
        paciente.Peso = double.Parse(txtPeso.Text);
        paciente.Alergias = txtAlergias.Text;
        paciente.Padecimientos = txtPadecimientos.Text;

        //usando la base de datos
        using (SqlConnection con = new SqlConnection(cadena_conexion))
        {
            try
            {
                con.Open();
                //Orden insertar para usar la palabra reservada INSERT
                INTO para indicar "Insertar en" tabla Pacientes "valores"

                string insertar = @"INSERT INTO Pacientes (Nombre,
Apellido, Edad, Altura, Peso, Alergia, Padecimiento) VALUES (@Nombre,
@Apellido, @Edad,
@Altura,@Peso,@Alergia,@Padecimiento)";

                SqlCommand cmd = new SqlCommand(insertar, con);
                cmd.Parameters.AddWithValue("@Nombre", paciente.Nombre);
                cmd.Parameters.AddWithValue("@Apellido",
paciente.Apellidos);
                cmd.Parameters.AddWithValue("@Edad", paciente.Edad);
                cmd.Parameters.AddWithValue("@Altura", paciente.Altura);
                cmd.Parameters.AddWithValue("@Peso", paciente.Peso);
                cmd.Parameters.AddWithValue("@Alergia",
paciente.Alergias);
                cmd.Parameters.AddWithValue("@Padecimiento ",
paciente.Padecimientos);
                cmd.ExecuteNonQuery();
                MessageBox.Show("Paciente registrado correctamente" +
MessageBoxButtons.OK);
                // Disparar evento para actualizar el otro formulario
                PacienteAgregado?.Invoke();
                this.Close();
            }
            catch { }
        }
    }
}

```



```

    }
    catch (Exception ex)
    {
        MessageBox.Show("Error al registrar paciente" +
ex.Message);
    }
}
}
}
}

```

Diseño

The image shows a screenshot of a Windows application window titled "Nuevo paciente". The window has a standard Windows title bar with minimize, maximize, and close buttons. Below the title bar, there is a tab labeled "Paciente". The main area of the window contains a form with the following fields:

- Nombre**: A text input field.
- Apellidos**: A text input field.
- Edad**: A text input field.
- Altura**: A text input field.
- Peso**: A text input field.
- Alergias medicinales**: A text input field.
- Padecimientos**: A text input field.

At the bottom center of the form, there is a blue icon depicting a person with a checkmark, indicating a successful registration or confirmation.

Agregar paciente

Primero junto con la creación de la cadena de conexión a la base de datos debemos crear también un evento de acción public event action de paciente agregado (el mismo al que se suscribió el formulario anterior) y posteriormente vamos a trabajar con una clase paciente para poder hacer el registro, la clase paciente contiene todos los datos necesarios para poder asignarlos como atributos, (Nombre, Apellido, Edad, Altura, Peso, Alergias y Padecimientos. Después del constructor con las conexiones a la base de datos y en el evento de click al botón de agregar paciente vamos a instanciar la clase Paciente creando un objeto paciente y a cada atributo perteneciente le asignamos lo registrado por el usuario en las cajas de texto correspondientes, después de entrar a la base de datos vamos a crear un insert a la tabla de Pacientes y agregamos los valores de cada columna de datos y a través de un comando tomamos estos valores registrados en los atributos de la clase y los insertamos a la columna correspondiente de la tabla ahora usamos Invoke para disparar el evento agregado de agregar paciente y finalmente mandamos a cerrar esta ventana además de incorporar un catch para obtener cual es el error en el caso de contar con alguno.

```
private void btnAgregar_Click_1(object sender, EventArgs e)
{
    //Obtener variables y datos en los atributos de la clase Paciente
    Paciente paciente = new Paciente();
    paciente.Nombre = txtNombre.Text;
    paciente.Apellidos = txtApellidos.Text;
    paciente.Edad = int.Parse(txtEdad.Text);
    paciente.Altura = double.Parse(txtAltura.Text);
    paciente.Peso = double.Parse(txtPeso.Text);
    paciente.Alergias = txtAlergias.Text;
    paciente.Padecimientos = txtPadecimientos.Text;

    //usando la base de datos
    using (SqlConnection con = new SqlConnection(cadena_conexion))
    {
        try
        {
            con.Open();
            //Orden insertar para usar la palabra reservada INSERT INTO para
            indicar "Insertar en" tabla Pacientes "valores"

            string insertar = @"INSERT INTO Pacientes (Nombre, Apellido,
            Edad, Altura, Peso, Alergia, Padecimiento) VALUES (@Nombre, @Apellido, @Edad,
            @Altura, @Peso, @Alergia, @Padecimiento)";

            SqlCommand cmd = new SqlCommand(insertar, con);
            cmd.Parameters.AddWithValue("@Nombre", paciente.Nombre);
            cmd.Parameters.AddWithValue("@Apellido", paciente.Apellidos);
            cmd.Parameters.AddWithValue("@Edad", paciente.Edad);
            cmd.Parameters.AddWithValue("@Altura", paciente.Altura);
            cmd.Parameters.AddWithValue("@Peso", paciente.Peso);
            cmd.Parameters.AddWithValue("@Alergia", paciente.Alergias);
            cmd.Parameters.AddWithValue("@Padecimiento ",
            paciente.Padecimientos);
```

```

        cmd.ExecuteNonQuery();
        MessageBox.Show("Paciente registrado correctamente" +
        MessageBoxButtons.OK);
        // Disparar evento para actualizar el otro formulario
        PacienteAgregado?.Invoke();
        this.Close();

    }
    catch (Exception ex)
    {

        MessageBox.Show("Error al registrar paciente" + ex.Message);

    }
}
}

```

Formulario de agendar/modificar una cita

Código completo:

```

using Microsoft.Data.SqlClient;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Astra
{
    public partial class Form5 : Form
    {
        Form1 = new Form1();

        string ruta;
        string cadena_conexion;
        public event Action CitaAgregada;
        private int idpacienteseleccionado;

        public Form5(int IdPaciente)
        {
            InitializeComponent();
            //Direcciones de la base de datos

            string ruta = Path.Combine(Application.StartupPath,
@"Data\AstraDB.mdf");
            cadena_conexion = "Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\AstraDB.mdf;
Integrated Security=True;Connect Timeout=30";

```

```

        idpacienteseleccionado = IdPaciente;
    }

    public class Cita
    {
        public DateTime NuevaCita { get; set; }
    }

    private void btnAgendar_Click(object sender, EventArgs e)
    {

        Cita = new Cita();

        cita.NuevaCita = mcCita.SelectionStart;

        using (SqlConnection con = new SqlConnection(cadena_conexion))
        {
            try
            {
                con.Open();
                //Insercion a la base de datos

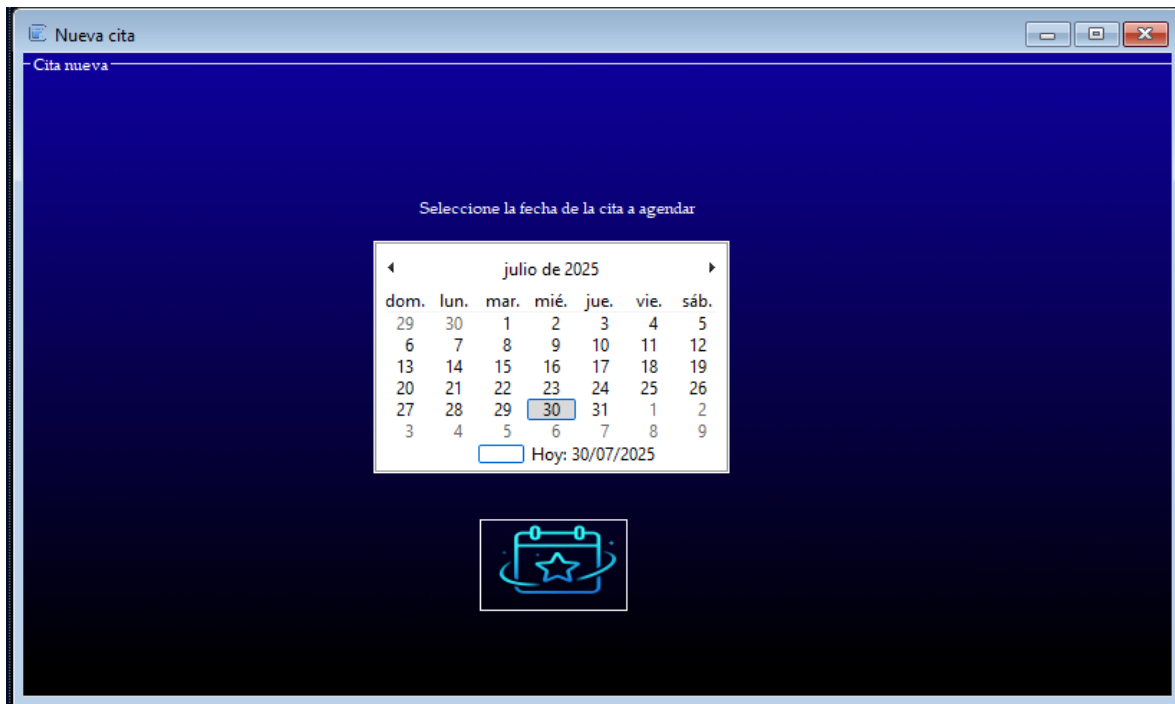
                string actualizar = @"UPDATE Pacientes SET Proxima_cita =
@Proxima_cita WHERE IdPaciente = @IdPaciente";

                SqlCommand cmd = new SqlCommand(actualizar, con);
                cmd.Parameters.AddWithValue("@Proxima_cita",
cita.NuevaCita);
                cmd.Parameters.AddWithValue("@IdPaciente",
idpacienteseleccionado);
                cmd.ExecuteNonQuery();
                MessageBox.Show("Cita seleccionada para la fecha: " +
cita.NuevaCita.ToShortDateString());

                CitaAgregada?.Invoke();
                this.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al agregar cita nueva " +
ex.Message);
            }
        }
    }
}

```

Diseño



Agendar/Modificar cita

El formulario sigue la misma lógica de crear un evento de acción publica y así mismo este evento debe de ser suscrito al formulario principal entonces seleccionamos un día del calendario y simplemente confirmamos, por lo que en el código dentro de la clase cita y su atributo de nueva cita del tipo DateTime se va a registrar el día seleccionado por el usuario. Por lo que instanciamos un objeto cita perteneciente a la clase Cita y mandamos a llamar el valor seleccionado a través de la propiedad de selectionstart del month calendar y como el valor por defecto de la celda es un valor nulo debemos hacer un update a la base de datos a la columna de próxima cita si la el valor de la columna de id paciente corresponde con el seleccionado anteriormente por lo que es importante tener una variable global que contenga el valor del id del paciente y se guarda a través del constructor del formulario mandando a llamar a la variable que debe estar definida en el formulario anterior, a esto yo le llamo variable de enlace. Finalmente solo hacemos una invocación al evento de acción de cita agregada y cerramos la ventana, también hacemos uso de un catch para poder mostrar el error en caso de tener alguno.

```
private void btnAgendar_Click(object sender, EventArgs e)
{
    Cita cita = new Cita();

    cita.NuevaCita = mcCita.SelectionStart;
```

```

using (SqlConnection con = new SqlConnection(cadena_conexion))
{
    try
    {
        con.Open();
        //Insercion a la base de datos

        string actualizar = @"UPDATE Pacientes SET Proxima_cita =
@Proxima_cita WHERE IdPaciente = @IdPaciente";

        SqlCommand cmd = new SqlCommand(actualizar, con);
        cmd.Parameters.AddWithValue("@Proxima_cita", cita.NuevaCita);
        cmd.Parameters.AddWithValue("@IdPaciente",
idpacienteseleccionado);
        cmd.ExecuteNonQuery();
        MessageBox.Show("Cita seleccionada para la fecha: " +
cita.NuevaCita.ToShortDateString());

        CitaAgregada?.Invoke();
        this.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error al agregar cita nueva " + ex.Message);
    }
}
}

```

Formulario de expediente

Código completo

```

using Microsoft.Data.SqlClient;
using System;
using System.IO;
using System.Windows.Forms;

namespace Astra
{
    public partial class Form6 : Form
    {
        string cadena_conexion;
        public Action pacienteAgregadoi;
        private int idpacienteseleccionado;
        private bool expedienteExiste = false;

        public Form6(int IdPaciente)
        {
            InitializeComponent();
            idpacienteseleccionado = IdPaciente;
        }
    }
}

```

```

        string ruta = Path.Combine(Application.StartupPath,
@"Data\AstraDB.mdf");
        cadena_conexion = "Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\AstraDB.mdf;
Integrated Security=True;Connect Timeout=30";

    }

    private void CargarExpediente()
    {
        using (SqlConnection con = new SqlConnection(cadena_conexion))
        {
            try
            {
                con.Open();
                string consulta = "SELECT Expediente FROM Expedientes
WHERE IdExpediente = @IdExpediente";
                SqlCommand cmd = new SqlCommand(consulta, con);
                cmd.Parameters.AddWithValue("@IdExpediente",
idpacienteseleccionado);
                SqlDataReader reader = cmd.ExecuteReader();
                if (reader.Read())
                {
                    Expediente.Text = reader["Expediente"].ToString();
                    expedienteExiste = true;
                }
                else
                {
                    expedienteExiste = false;
                    // 0 puedes dejar el TextBox vacío si es nuevo
                    Expediente.Text = "";
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al cargar el expediente: " +
ex.Message);
            }
        }
    }

    private void Form6_Load(object sender, EventArgs e)
    {
        CargarExpediente();
    }

    private void btnConfirmar_Click(object sender, EventArgs e)
    {
        string expediente = Expediente.Text.Trim();

        if (string.IsNullOrEmpty(expediente))
        {
            MessageBox.Show("El campo expediente no puede estar vacío.");
            return;
        }
    }

```

```

        using (SqlConnection con = new SqlConnection(cadena_conexion))
        {
            try
            {
                con.Open();

                string query;

                if (expedienteExiste)
                {
                    // Ya existe -> UPDATE
                    query = "UPDATE Expedientes SET Expediente =
@Expediente WHERE IdExpediente = @IdExpediente";
                }
                else
                {
                    // No existe -> INSERT
                    query = "INSERT INTO Expedientes (IdExpediente,
Expediente) VALUES (@IdExpediente, @Expediente)";
                }

                SqlCommand cmd = new SqlCommand(query, con);
                cmd.Parameters.AddWithValue("@Expediente", expediente);
                cmd.Parameters.AddWithValue("@IdExpediente",
idpacienteseleccionado);
                cmd.ExecuteNonQuery();

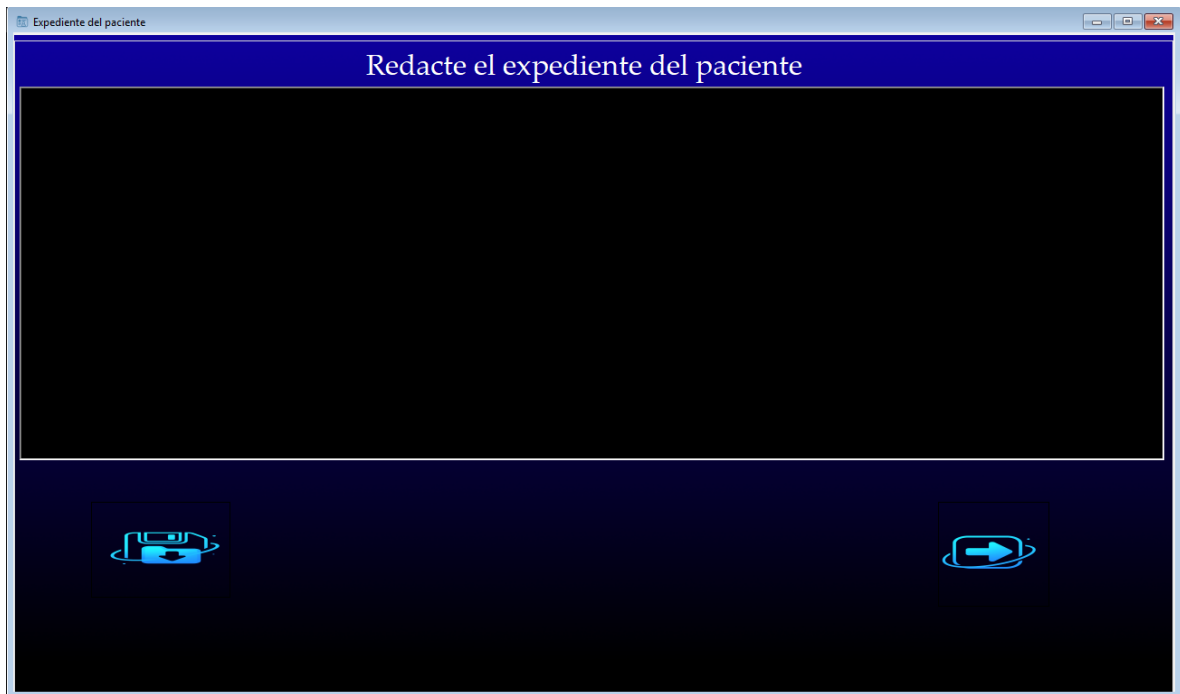
                MessageBox.Show("Expediente guardado correctamente");

                expedienteExiste = true; // Si era nuevo, ya existe
ahora.
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al guardar el expediente: " +
ex.Message);
            }
        }

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

```


Diseño



Expediente

La lógica del expediente permite que si no existe uno se pueda visualizar el richtextbox de manera vacía para poder redactar uno, requerimos al igual que para la cita un id de paciente seleccionado y adicionalmente requerimos una variable de tipo booleana inicializada en false lo que permitirá determinar si existe o no nuestro expediente y así mismo creamos un evento de acción público de expediente agregado. Por lo que primero traemos el id seleccionado a través del constructor y posteriormente vamos a crear un método para cargar el expediente, este método utiliza una propiedad de sql llamada data reader a través de un select para realizar una consulta a la base de datos instanciamos un reader y utilizando el id seleccionado y un comando de sql vamos a leer el dato guardado en la celda a través del reader y si este contiene un texto entonces actualizamos la variable booleana como true y en el caso contrario de no existir inicializamos la variable como false y dejamos el textbox vacío, este método se ejecuta en un evento de carga del formulario.

Confirmar expediente

Ahora volvemos a entrar a la base de datos y creamos un string llamado query con nuestra variable booleana si el expediente existe vamos a ejecutar un update a través del query para que al guardar lo nuevamente redactado esto no afecte a los valores anteriores y si no existe entonces se ejecuta un insert para agregar nuevos caracteres de texto, ejecutamos el cmd con los valores correspondientes del query y si el formulario hizo un insert, es decir si guardo un nuevo valor que antes no existía entonces actualizamos la variable booleana a true.

Base de datos

Localdb

Para que nuestra aplicación funcione de manera adecuada sin el uso de una conexión a internet utilizamos localdb a través de la descarga del cliente de sql proporcionado por Microsoft dentro del propio visual studio, entonces modelamos las tablas de la base de datos dentro de un archivo .mdf adjuntado directamente en los archivos del proyecto y establecemos en cada formulario una variable que contiene la ruta de la dirección de este archivo local creado e instanciado por la propia aplicación y conectamos a través de una cadena de conexión que apunta al archivo de la base de datos a través de un data source del mismo cliente de localdb y un attachdbfile que busca en el directorio llamado data donde se encuentra el archivo AstraDB.mdf.

Es importante importar la librería de Microsoft.Data.SqlClient.

```
string ruta = Path.Combine(Application.StartupPath, @"Data\AstraDB.mdf");  
cadena_conexion = "Data  
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\AstraDB.mdf;  
Integrated Security=True;Connect Timeout=30";
```

Tablas

	Nombre	Tipo de datos	Permitir valores NULL	Valor predeterminado
1	Id	int	<input type="checkbox"/>	
2	Usuario	nchar(100)	<input checked="" type="checkbox"/>	
3	Contraseña	nchar(10)	<input checked="" type="checkbox"/>	
4			<input type="checkbox"/>	

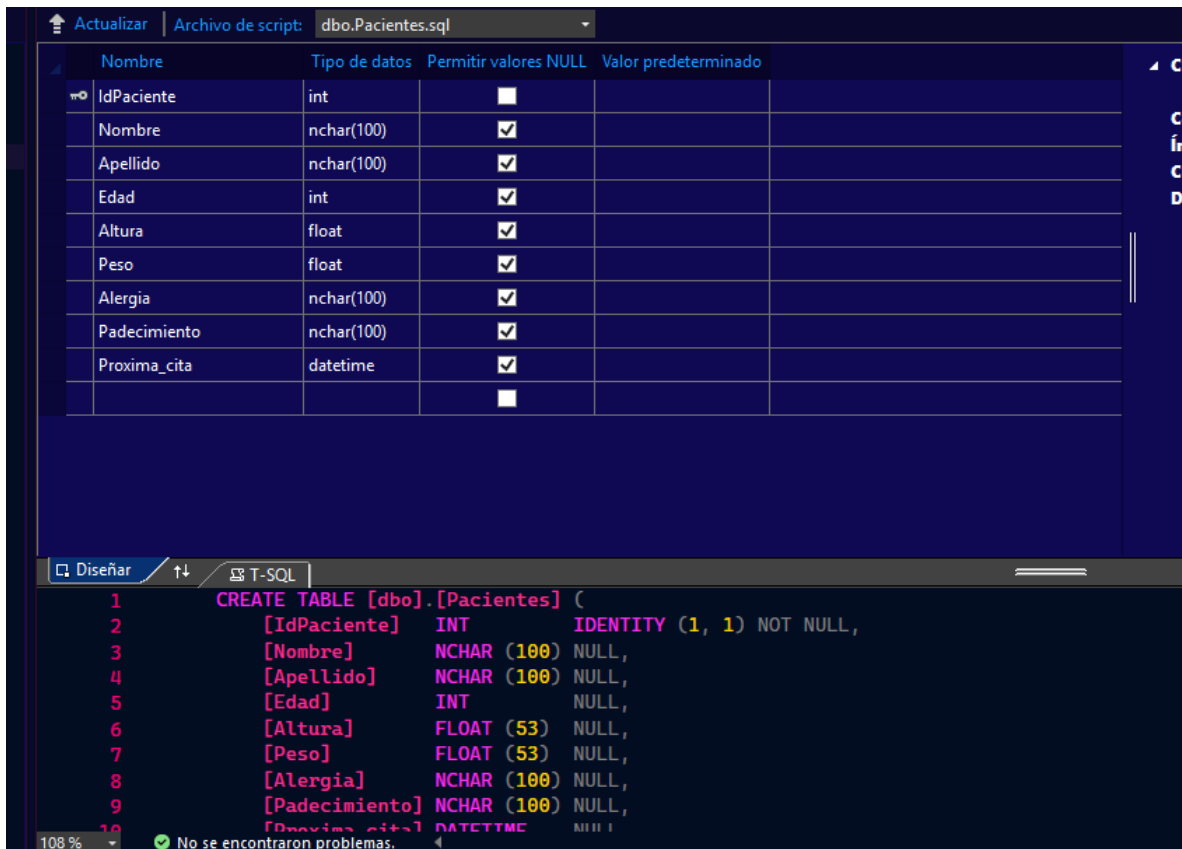
Diseñar

T-SQL

```
1 CREATE TABLE [dbo].[Usuarios] (  
2     [Id] INT IDENTITY (1, 1) NOT NULL,  
3     [Usuario] NCHAR (100) NULL,  
4     [Contraseña] NCHAR (10) NULL,  
5     PRIMARY KEY CLUSTERED ([Id] ASC)  
6 );  
7  
8
```

100 % ☒ No se encontraron problemas.

La primera tabla que modelamos para la base de datos es la de Usuarios, esta nos permite guardar en variables de tipo varchar(50) los datos de usuario y contraseña que ingrese el usuario y buscar si ya existen o no para determinar si el usuario tiene acceso o no a la base de datos.



Después modelamos la tabla del formulario principal: La tabla de Pacientes.

Siendo esta la que contiene la información principal de la que vamos a ir haciendo uso a los demás formularios y la que nos muestra los datos de cada paciente constando de diferentes columnas como una de Id, Nombre, Apellido, Edad, Estatura, Peso, Alergias y Padecimientos.

Ahora que ya esta creada esta tabla simplemente realizamos updates e insert correspondientes para cada datos necesario.

Nombre	Tipo de datos	Permitir valores NULL	Valor predeterminado
IdExpediente	int	<input type="checkbox"/>	
Expediente	nvarchar(MAX)	<input checked="" type="checkbox"/>	
		<input type="checkbox"/>	

Diseñar

T-SQL

```
1 CREATE TABLE [dbo].[Expedientes] (  
2     [IdExpediente] INT NOT NULL,  
3     [Expediente] NVARCHAR (MAX) NULL,  
4     PRIMARY KEY CLUSTERED ([IdExpediente] ASC)  
5 );  
6  
7
```

100 % No se encontraron problemas.

Y finalmente la tabla mas sencilla es la de expediente que utiliza un id y además de eso utiliza un enlace directo al id de paciente para registrarlo en el id del expediente, además de que nos guarda el expediente en una variable de tipo varchar usando el máximo de caracteres permitidos.