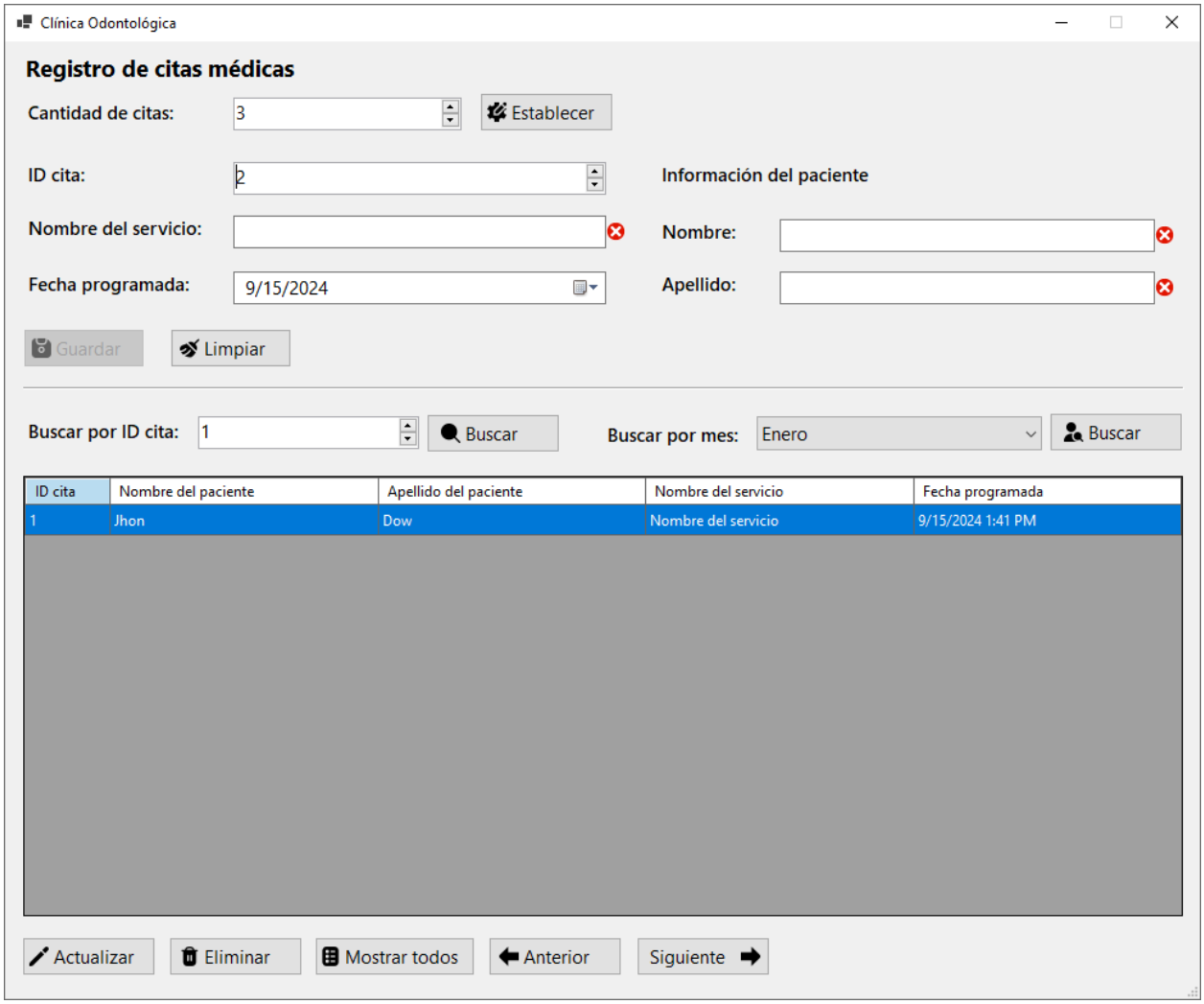
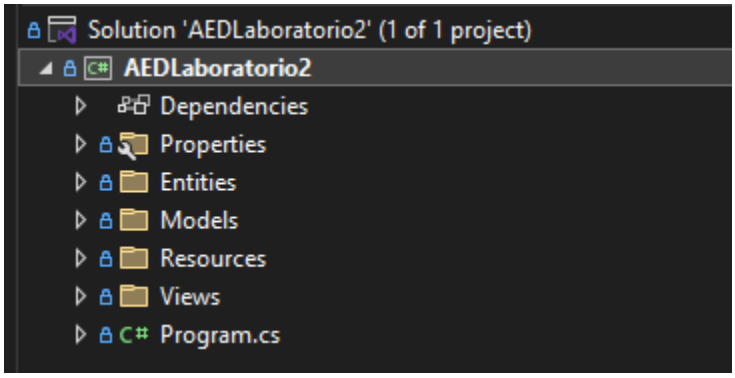
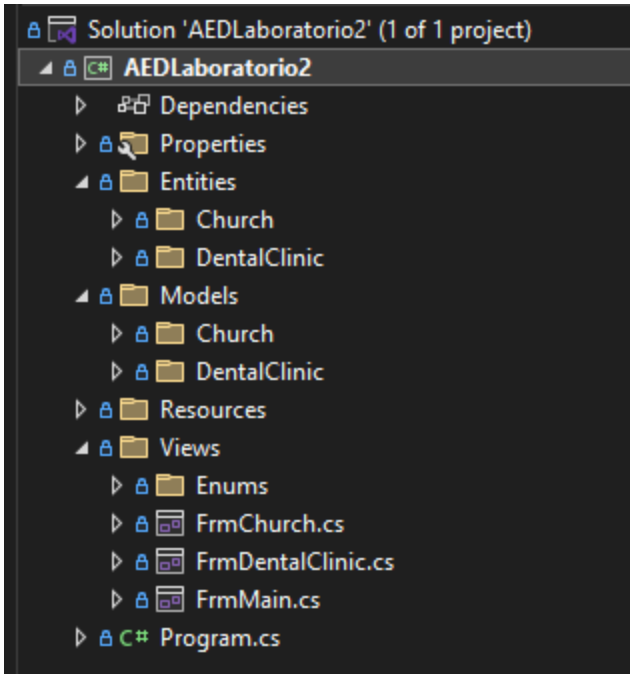
2do ejercicio - Sistema para control de citas de una clínica odontológica



Estructura del proyecto





Crear clase Appointment (Cita) dentro de la carpeta Entities > DentalClinic

```
1   namespace AEDLaboratorio2.Entities.DentalClinic
2   {
3       public class Appointment
4       {
5           public int Id { get; set; }
6           public string? PatientName { get; set; }
7           public string? PatientSurname { get; set; }
8           public string? Service { get; set; }
9           public DateTime ScheduledDate { get; set; }
10      }
11  }
```

Crear clase AppointmentModel dentro de la carpeta Models > DentalClinic

```
1   using AEDLaboratorio2.Entities.DentalClinic;
2
3   namespace AEDLaboratorio2.Models.DentalClinic
4   {
5       public class AppointmentModel
6       {
7           private int _size = 0, _quantity = 0;
8           private static Appointment[] _appointments = [];
```

Método para inicializar el tamaño del arreglo de citas

```
public (bool result, string message) InitializeArray(int size)
{
    if (size <= 0)
        return (false, "Tamaño del arreglo debe ser > 0");

    _size = size;
    _appointments = new Appointment[_size];

    return (true, "Arreglo creado exitosamente");
}
```

Método para obtener la posición de una cita en el arreglo por medio de su ID

```
private int FindIndex(int appointmentId)
{
    int index = -1;
    for (var i = 0; i < _quantity; i++)
        if (_appointments[i].Id == appointmentId)
        {
            index = i;
            return index;
        }

    return index;
}
```

Método para determinar la posición en el arreglo que ocupará la nueva cita

```
private int FindIndexToInsert(DateTime scheduledDate)
{
    int index = 0;

    while (index < _quantity && _appointments[index].ScheduledDate < scheduledDate)
        index++;

    if (index >= _quantity || _appointments[index].ScheduledDate > scheduledDate)
        return -index;

    return index;
}
```

Método para determinar si ya existe una cita con el ID ingresado

```
private bool IsExistingId(int id)
{
    for (var i = 0; i < _quantity; i++)
        if (_appointments[i].Id == id)
            return true;

    return false;
}
```

Método para agregar una nueva cita

```
public (bool result, string message) Add(Appointment appointment)
{
    if (_size == 0)
        return (false, "Primero debe asignar un tamaño al arreglo");

    if (_quantity >= _size)
        return (false, "No hay espacio en el arreglo para un nueva cita");

    if (IsExistingId(appointment.Id))
        return (false, "Ya existe una cita con el ID ingresado");

    var index = FindIndexToInsert(appointment.ScheduledDate);
    if (index > 0)
        return (false, "Ya existe una cita con el ID ingresado");

    index = -index;
    for (var i = _quantity; i > index; i--)
        _appointments[i] = _appointments[i - 1];

    _appointments[index] = appointment;
    ++_quantity;

    return (true, "Cita registrada exitosamente");
}
```

Método para obtener todas las citas

```
public Appointment[] GetAll() => _appointments.Take(_quantity).ToArray();
```

Método para eliminar una cita

```
public (bool result, string message) Delete(int id)
{
    if (_size == 0)
        return (false, "Primero debe asignar un tamaño al arreglo");

    if (_quantity == 0)
        return (false, "No hay citas para eliminar");

    var index = FindIndex(id);
    if (index < 0)
        return (false, "No existe una cita con el ID ingresado");

    for (int i = index; i < _quantity - 1; i++)
        _appointments[i] = _appointments[i + 1];

    --_quantity;

    return (true, "Cita eliminado exitosamente");
}
```

Método para actualizar una cita

```
public (bool result, string message) Update(Appointment appointment)
{
    if (_size == 0)
        return (false, "Primero debe asignar un tamaño al arreglo");

    if (_quantity == 0)
        return (false, "No hay cita para actualizar");

    var (result, message) = Delete(appointment.Id);
    if (!result)
        return (false, message);

    var (result1, message1) = Add(appointment);
    if (!result1)
        return (false, message1);

    return (true, "Cita actualizada exitosamente");
}
```

Método para obtener una cita por medio de su ID

```
public (bool result, string message, Appointment? appointment) GetById(int id)
{
    if (_size == 0)
        return (false, "Primero debe asignar un tamaño al arreglo", null);

    if (_quantity == 0)
        return (false, "No hay citas para buscar", null);

    var index = FindIndex(id);

    return id < 0 ? (false, "No existe una cita con el ID ingresado", null) :
        (true, $"Cita con Id: {id} encontrado", _appointments[index]);
}
```

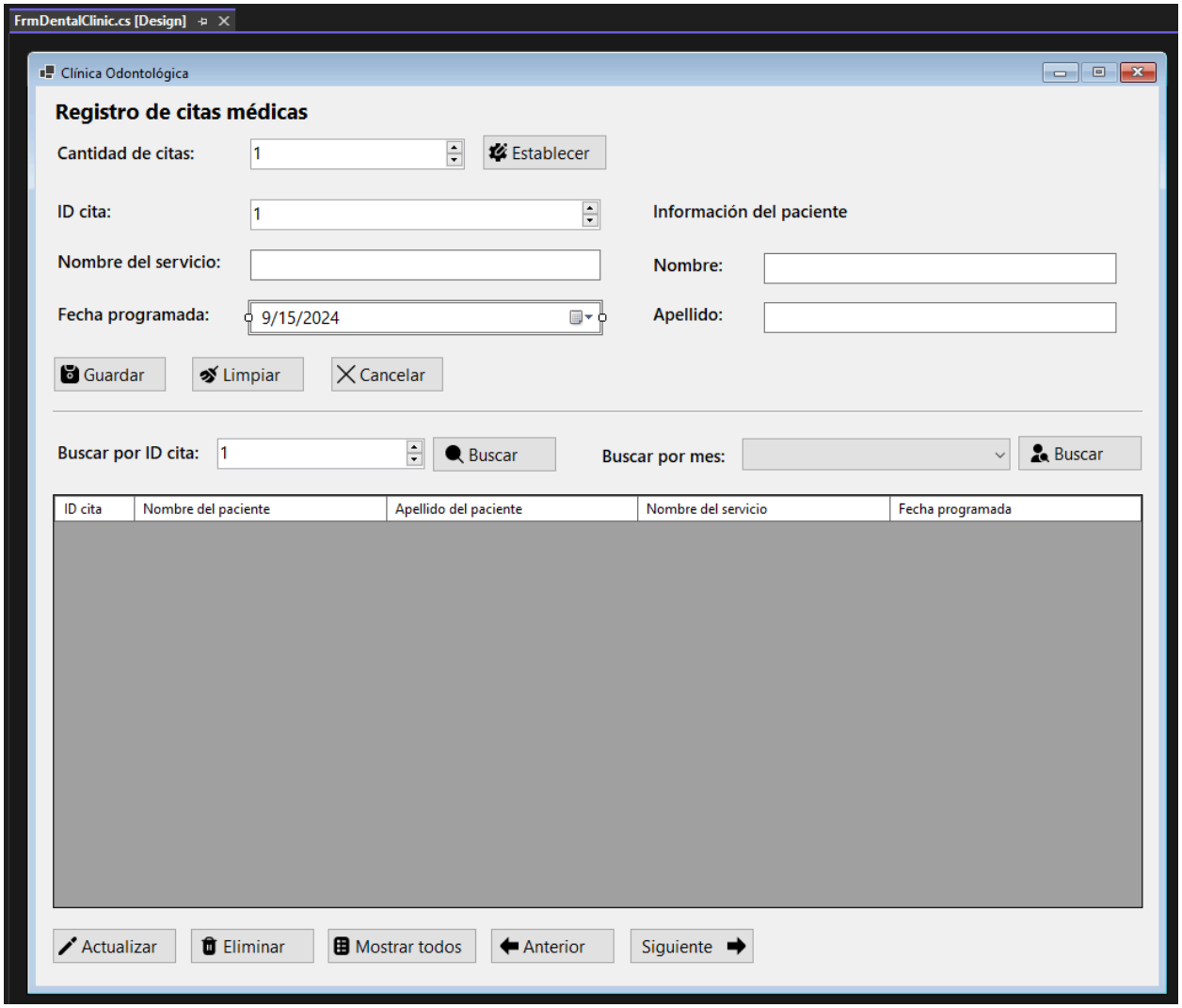
Método para obtener citas por mes

```
public (bool result, string message, Appointment[] appointments) GetByMonth(int month)
{
    if (_size == 0)
        return (false, "Primero debe asignar un tamaño al arreglo", []);

    if (_quantity == 0)
        return (false, "No hay citas para buscar", []);

    var appointments = _appointments
        .Take(_quantity).Where(a => a.ScheduledDate.Month == month).ToArray();

    return (true, "Citas encontradas", appointments);
}
```

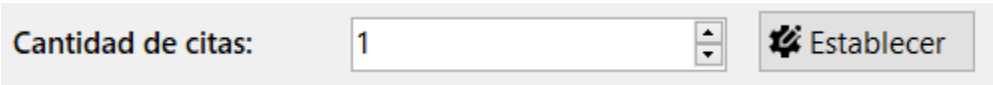
Resumen de toda la clase AppointmentModel

```
public class AppointmentModel
{
    private int _size = 0, _quantity = 0;
    private static Appointment[] _appointments = [];

    public (bool result, string message) InitializeArray(int size)...

    private int FindIndex(int appointmentId)...

    private int FindIndexToInsert(DateTime scheduledDate)...

    private bool IsExistingId(int id)...

    public (bool result, string message) Add(Appointment appointment)...

    public (bool result, string message) Update(Appointment appointment)...

    public (bool result, string message) Delete(int id)...

    public (bool result, string message, Appointment? appointment) GetById(int id)...

    public (bool result, string message, Appointment[] appointments) GetByMonth(int month)...

    public Appointment[] GetAll() => _appointments.Take(_quantity).ToArray();
}
```

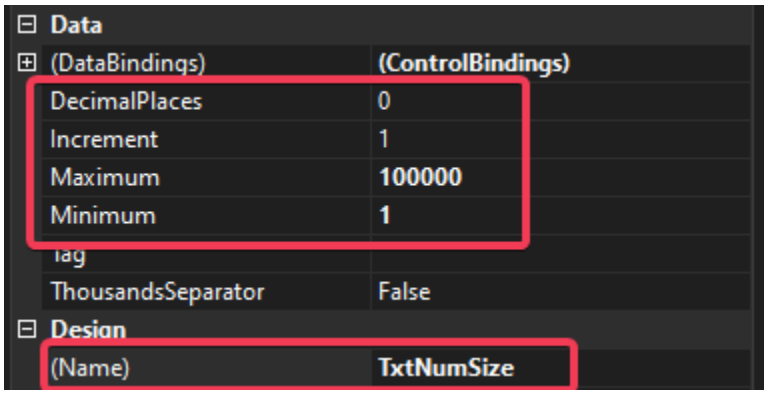
Diseño de la interfaz de usuario



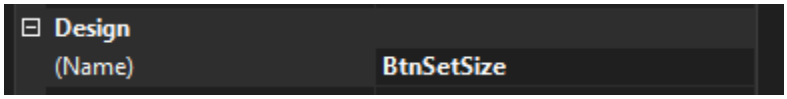
Propiedades de los componentes



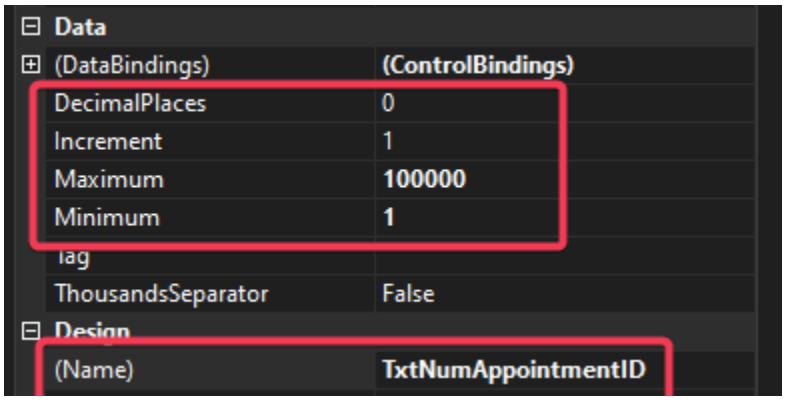
NumericUpDown (Campo numérico para cantidad de citas a registrar)



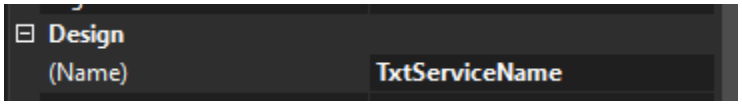
Button (Establecer)

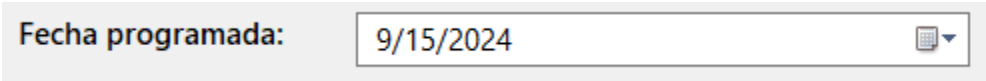


NumericUpDown (Campo numérico para ID de la cita)

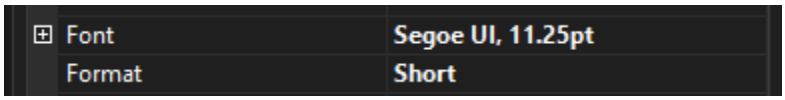


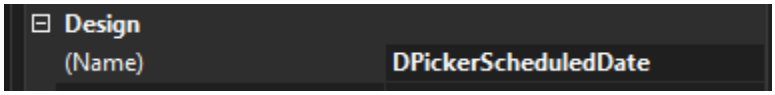
TextBox (Campo para el nombre del servicio)

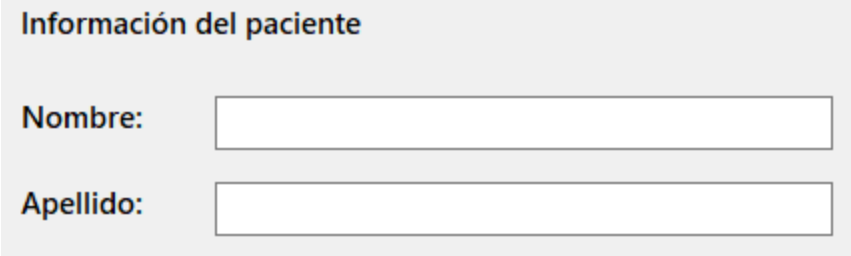




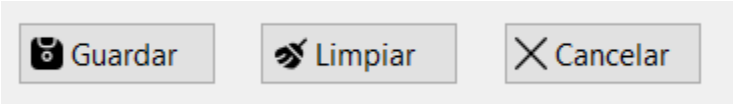
DateTimePicker (Fecha programada para la cita)



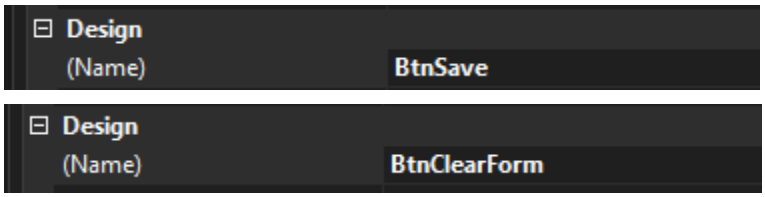




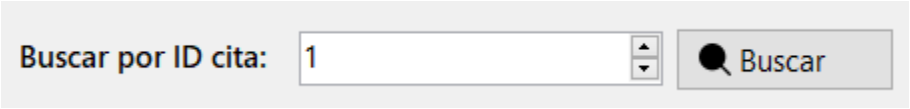
TextBox (Nombre y apellido del paciente)



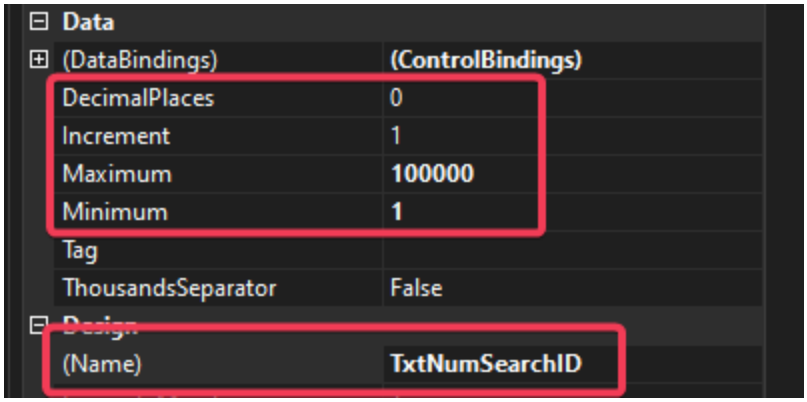
Button (Botón guardar, limpiar y cancelar)



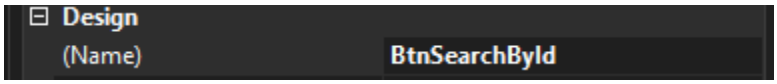
Botón **Cancelar** (Esta invisible por defecto, será visible solo al momento de editar un registro)

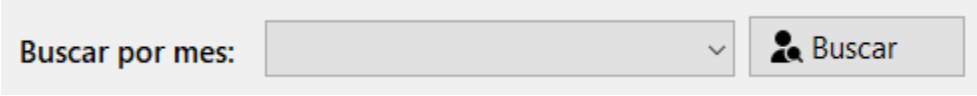


NumericUpDown (Campo numérico para buscar cita por medio de su ID)

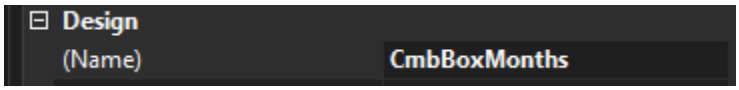


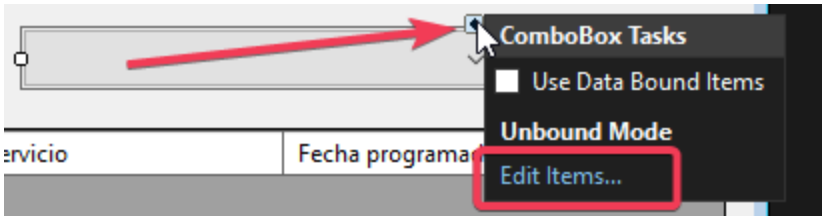
Button (Buscar)

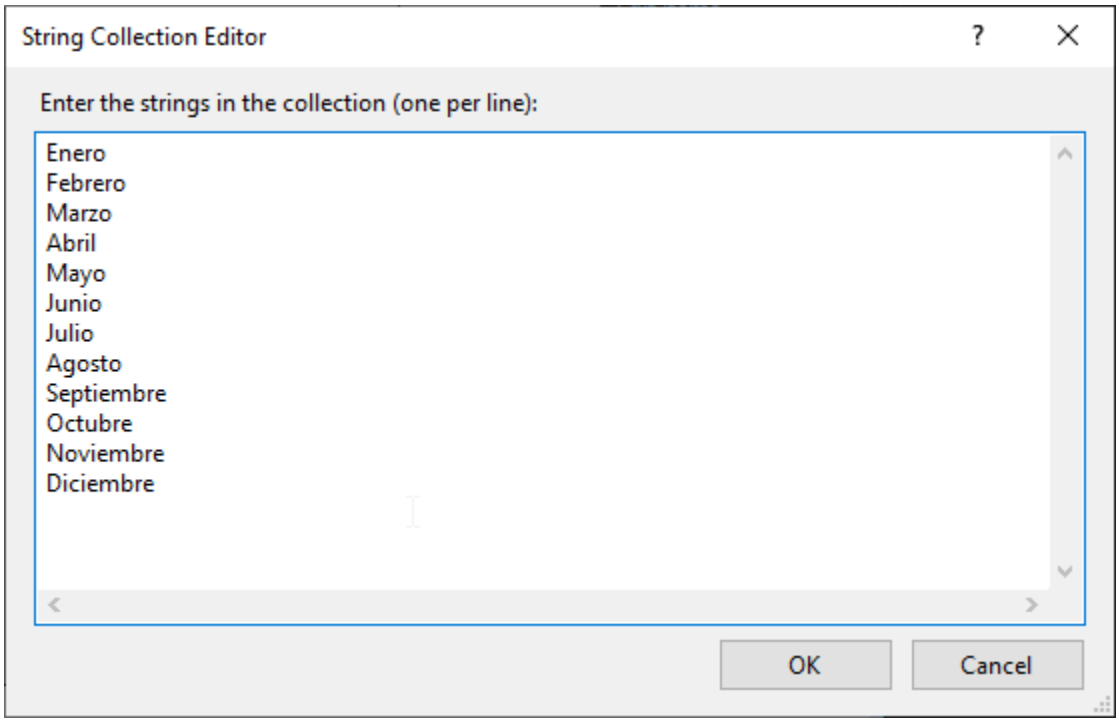




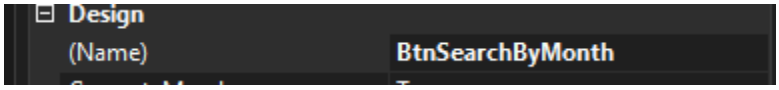
ComboBox (Para los meses del año)

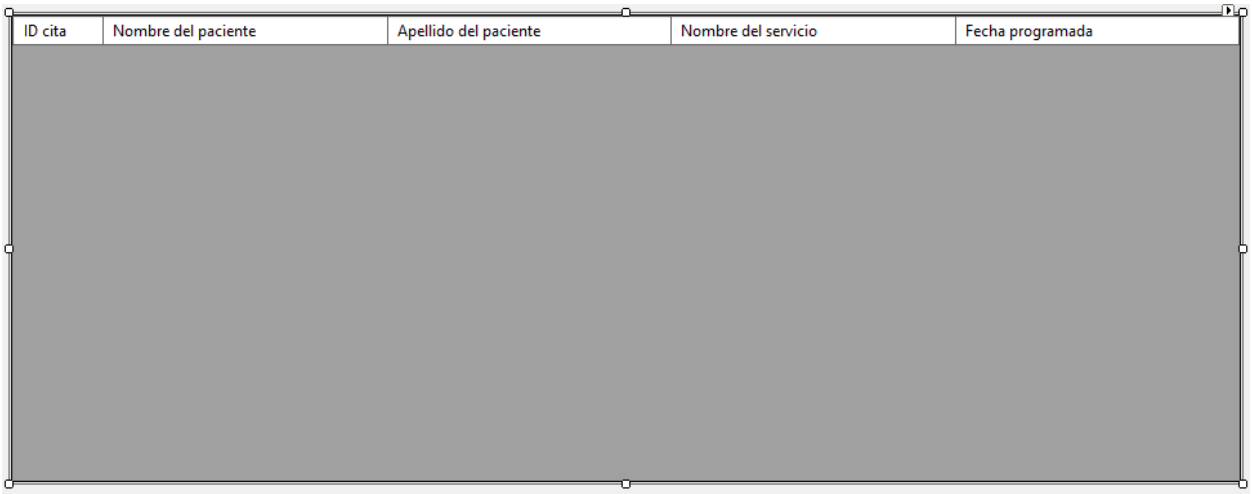




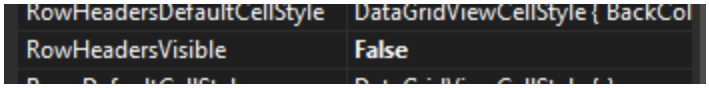


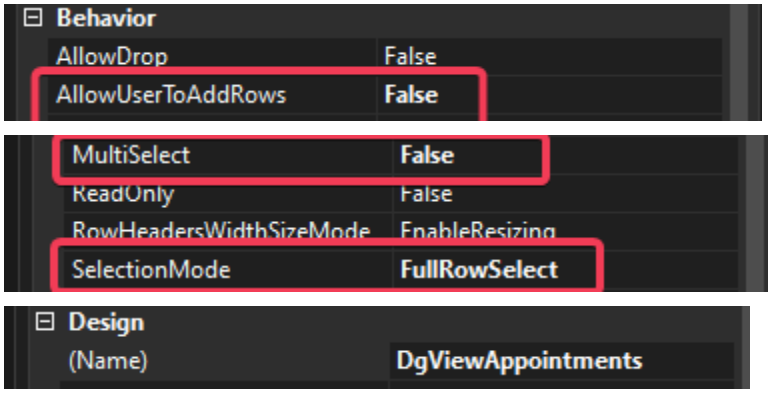
Button (Buscar)

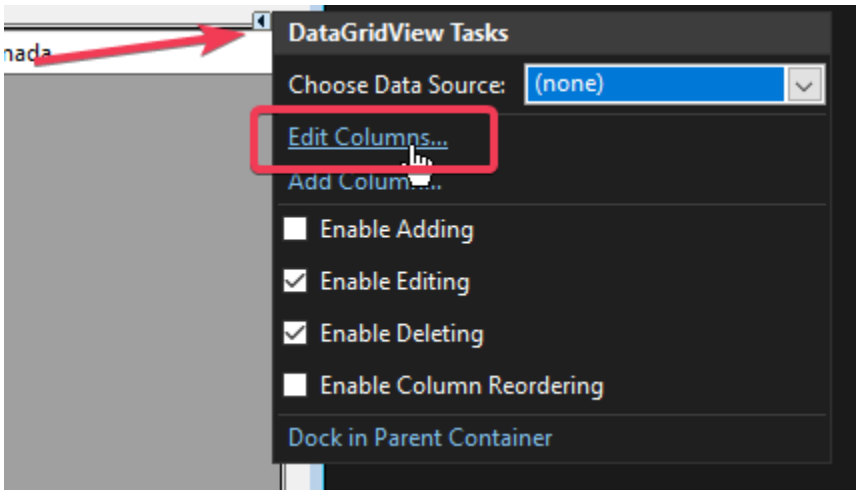




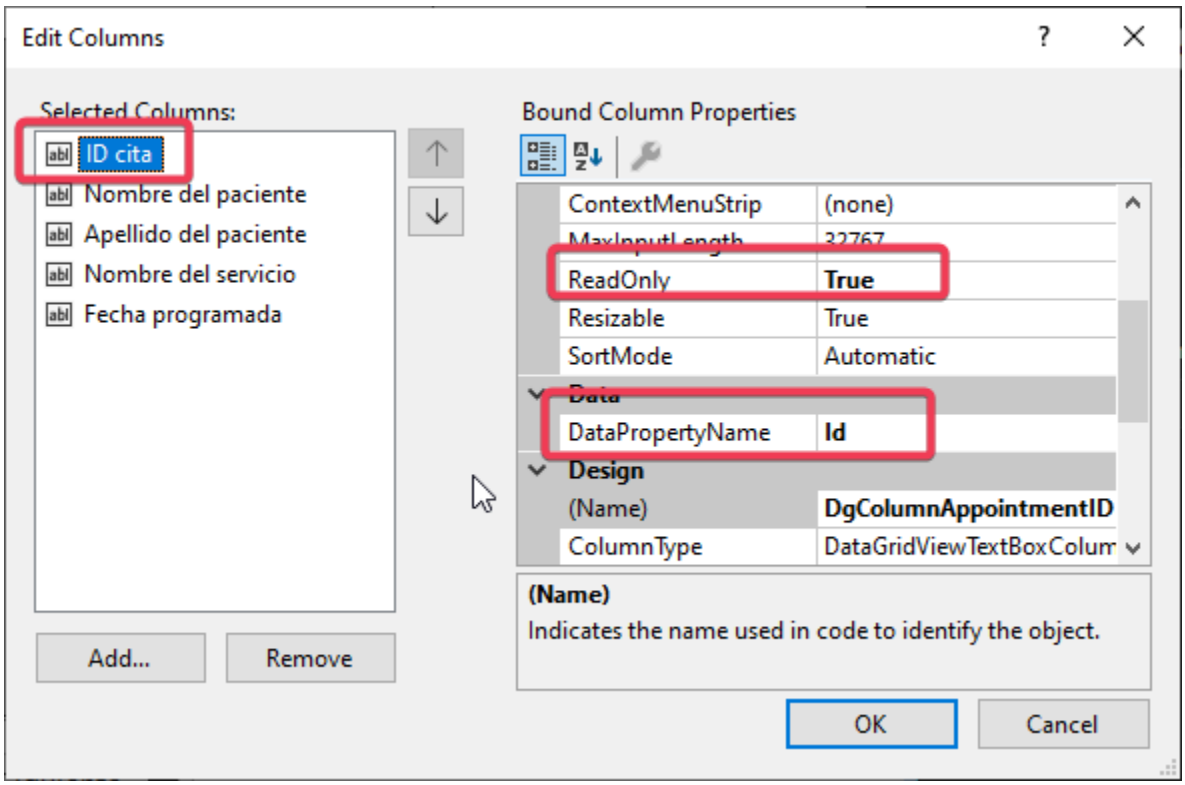
DataGridView (Tabla para mostrar las citas)

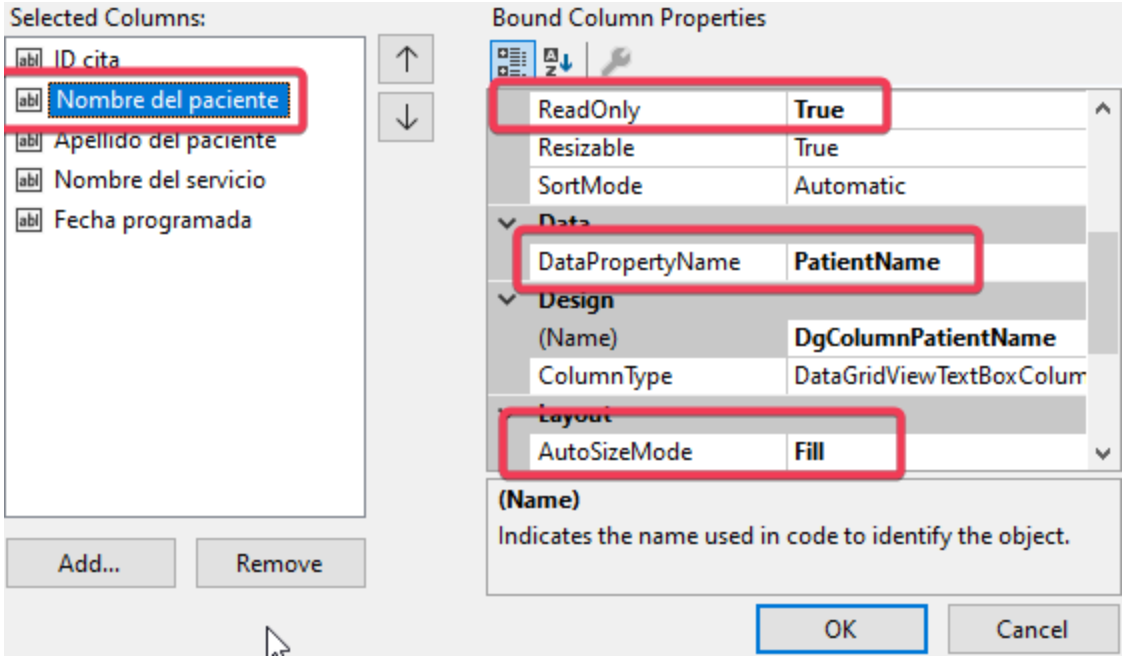


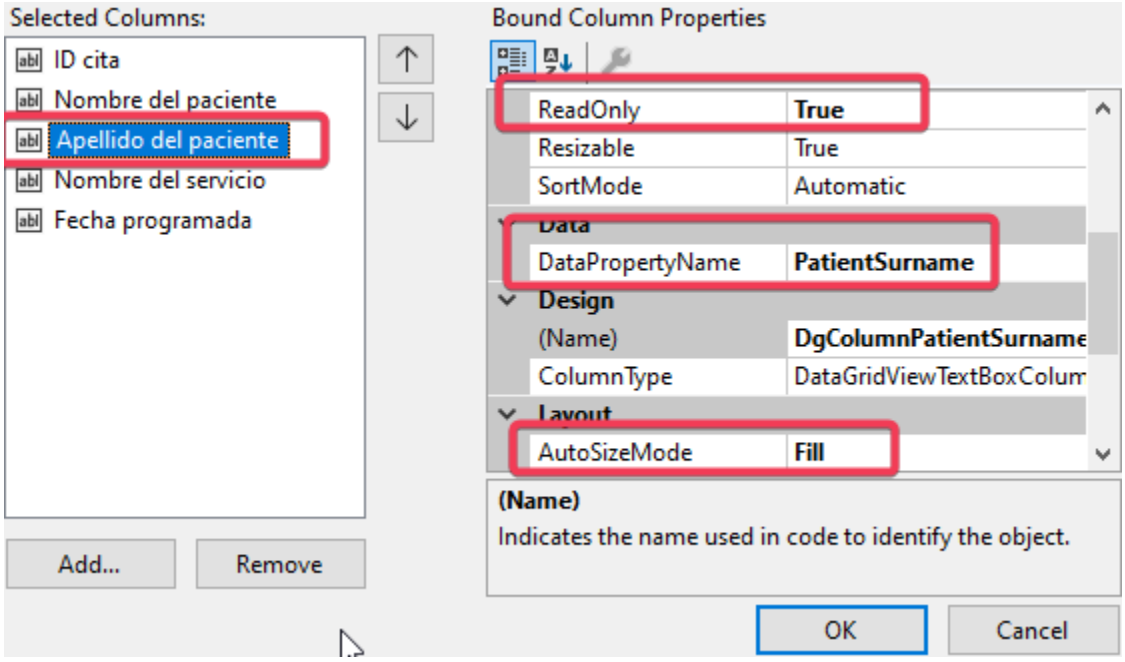


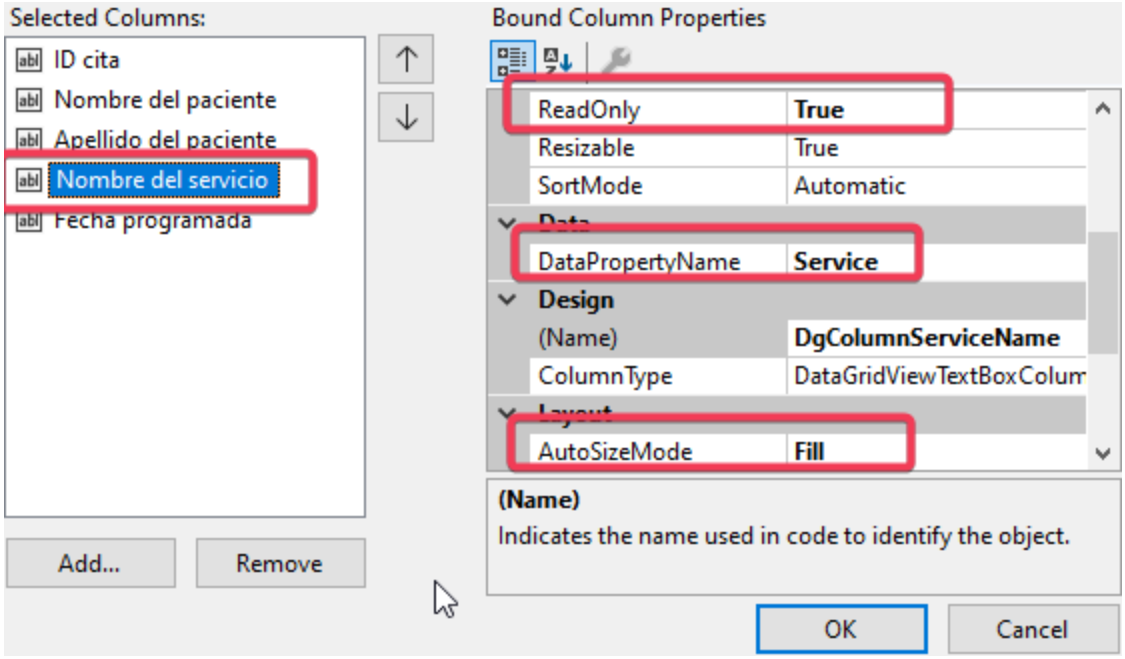


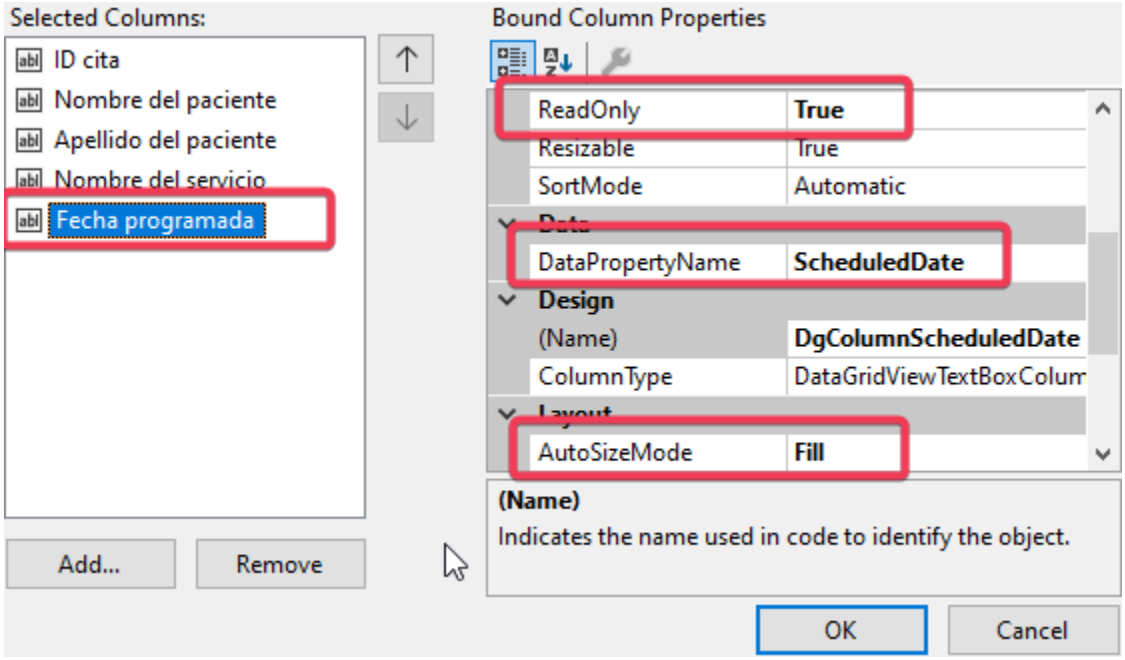
Agregar columnas



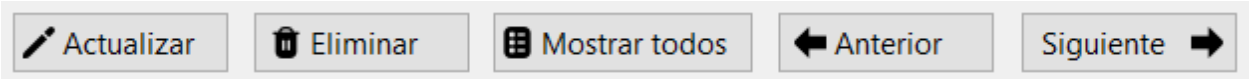

Selected Columns:
ID cita
Nombre del paciente
Apellido del paciente
Nombre del servicio
Fecha programada
Add...
Remove
Bound Column Properties
ReadOnly
True
Resizable
True
SortMode
Automatic
DataPropertyName
PatientName
Design
(Name)
DgColumnPatientName
ColumnType
DataGridViewTextBoxColum
AutoSizeMode
Fill
(Name)
Indicates the name used in code to identify the object.
OK
Cancel

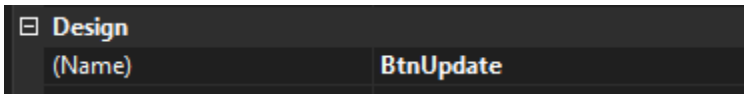

Selected Columns:
ID cita
Nombre del paciente
Apellido del paciente
Nombre del servicio
Fecha programada
Add...
Remove
Bound Column Properties
ReadOnly
True
Resizable
True
SortMode
Automatic
DataPropertyName
PatientSurname
Design
(Name)
DgColumnPatientSurname
ColumnType
DataGridViewTextBoxColum
AutoSizeMode
Fill
(Name)
Indicates the name used in code to identify the object.
OK
Cancel

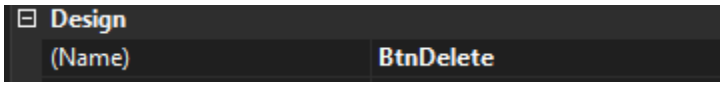


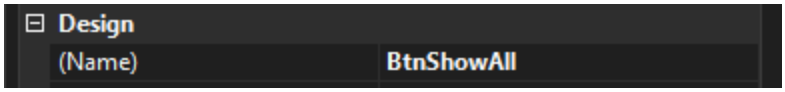


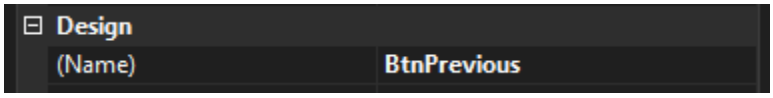
Ultimos botones

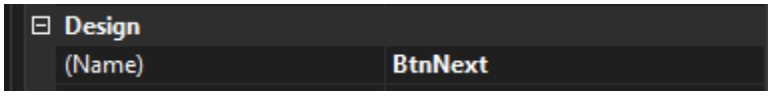




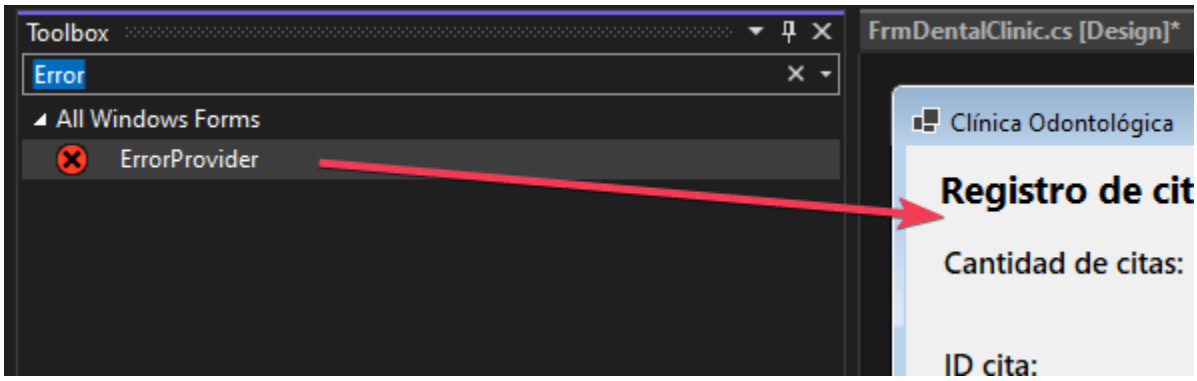


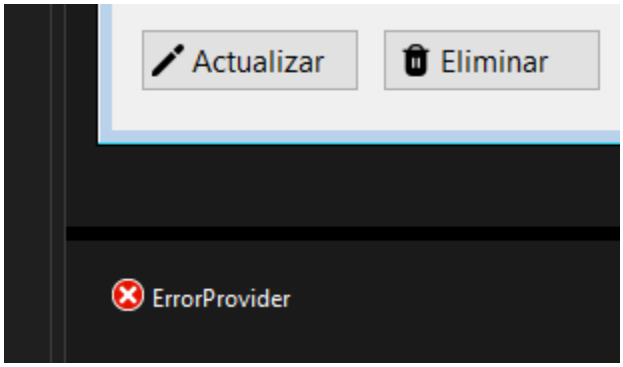


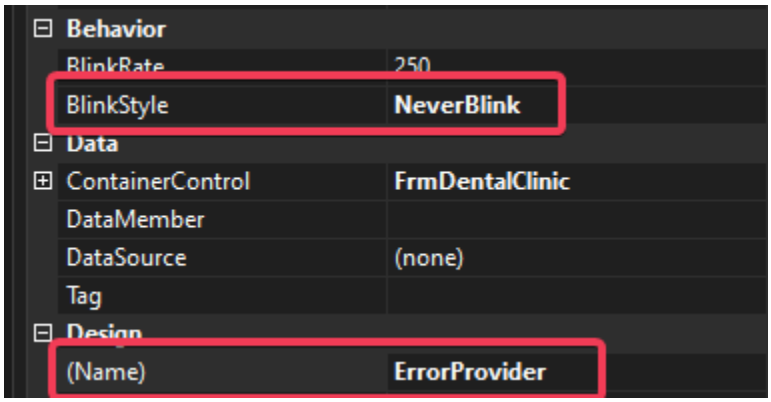




Finalmente arrastramos un ErrorProvider sobre el formulario







Crear enum “FormOperation” en Views > Enums

```
namespace AEDLaboratorio2.Views.Enums
{
    public enum FormOperation
    {
        Create, Update
    }
}
```

Variables globales del formulario

```
public partial class FrmDentalClinic : Form
{
    private readonly AppointmentModel _appointmentModel = new();
    private FormOperation _formOperation = FormOperation.Create;
    private bool _isValidForm;
```

Evento Click del botón “Establecer”

```
private void BtnSetSize_Click(object sender, EventArgs e)
{
    var (result, msg) = _appointmentModel.InitializeArray(Convert.ToInt32(TxtNumSize.Value));
    if (!result)
        MessageBox.Show(msg, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Método “ClearForm”

```
private void ClearForm()
{
    TxtNumAppointmentID.Value = TxtNumAppointmentID.Minimum;
    TxtServiceName.Text = string.Empty;
    DPickerScheduledDate.Value = DateTime.Now;
    TxtPatientName.Text = string.Empty;
    TxtPatientSurname.Text = string.Empty;
}
```

Evento Click del botón “Limpiar”

```
private void BtnClearForm_Click(object sender, EventArgs e) => ClearForm();
```

Evento Click del botón “Anterior” para retroceder una fila por el DataGridView

```
private void BtnPrevious_Click(object sender, EventArgs e)
{
    if (DgViewAppointments.SelectedRows.Count <= 0)
        return;

    var currentRowIndex = DgViewAppointments.SelectedRows[0].Index;
    if (currentRowIndex > 0)
        DgViewAppointments.Rows[currentRowIndex - 1].Selected = true;
}
```

Evento Click del botón “Siguiente” para avanzar una fila por el DataGridView

```
private void BtnNext_Click(object sender, EventArgs e)
{
    if (DgViewAppointments.SelectedRows.Count <= 0)
        return;

    var currentRowIndex = DgViewAppointments.SelectedRows[0].Index;
    if (currentRowIndex < DgViewAppointments.Rows.Count - 1)
        DgViewAppointments.Rows[currentRowIndex + 1].Selected = true;
}
```

Evento Click del botón “Buscar cita por ID”

```
private void BtnSearchById_Click(object sender, EventArgs e)
{
    (bool result, string message, Appointment? appointment) =
        _appointmentModel.GetById(Convert.ToInt32(TxtNumSearchID.Value));

    if (!result && appointment is null)
    {
        MessageBox.Show(message, "Atención", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return;
    }

    DgViewAppointments.DataSource = new Appointment?[] { appointment };
}
```

Evento Click del botón “Buscar cita por mes”

```
private void BtnSearchByMonth_Click(object sender, EventArgs e)
{
    (bool result, string message, Appointment[] appointments) =
        _appointmentModel.GetByMonth(CmbBoxMonths.SelectedIndex + 1);

    if (!result && appointments is null)
    {
        MessageBox.Show(message, "Atención", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return;
    }

    DgViewAppointments.DataSource = appointments;
}
```

Método “ValidateTextBox” para realizar validaciones usando ErrorProvider

```
private bool ValidateTextBox(TextBox textBox, string errorMessage)
{
    if (string.IsNullOrWhiteSpace(textBox.Text))
    {
        ErrorProvider.SetError(textBox, errorMessage);
        return false;
    }

    ErrorProvider.SetError(textBox, string.Empty);

    return true;
}
```

Método “ValidateTextBoxes” que se apoya de “ValidateTextBox” para mostrar mensaje de error por medio de ErrorProvider y retorna una tupla indicando si el valor de cada TextBox es correcto o no

```
private (bool isServiceValid, bool isPatientNameValid, bool isPatientSurnameValid) ValidateTextBoxes()
{
    bool isServiceValid = ValidateTextBox(TxtServiceName, "Nombre del servicio es obligatorio");
    bool isPatientNameValid = ValidateTextBox(TxtPatientName, "Nombre del paciente es obligatorio");
    bool isPatientSurnameValid = ValidateTextBox(TxtPatientSurname, "Apellido del paciente es obligatorio");

    return (isServiceValid, isPatientNameValid, isPatientSurnameValid);
}
```

Método “TxtTextChanged” que ejecuta el método “ValidateTextBoxes” cada vez que cambia el texto de cada campo del formulario, si todos contienen la información en el formato esperado el botón de “Guardar” se habilita automáticamente, caso contrario se deshabilita.

```
private void TxtTextChanged()
{
    (bool isServiceValid, bool isPatientNameValid, bool isPatientSurnameValid) = ValidateTextBoxes();
    _isValidForm = isServiceValid && isPatientNameValid && isPatientSurnameValid;
    BtnSave.Enabled = _isValidForm;
}
```

Método “TxtKeyPress” para asegurarnos que solo se ingrese letras, espacios en blancos y backspace en los campos de texto

```
private void TxtKeyPress(object? sender, KeyPressEventArgs e)
{
    if (!char.IsLetter(e.KeyChar) && !char.IsWhiteSpace(e.KeyChar) && e.KeyChar != (char)Keys.Back)
    {
        e.Handled = true;
        return;
    }
}
```

Vincular métodos “TxtTextChanged” y “TxtKeyPress” a los campos de texto en el constructor del formulario

```
public FrmDentalClinic()
{
    InitializeComponent();
    TxtServiceName.KeyPress += TxtKeyPress;
    TxtPatientName.KeyPress += TxtKeyPress;
    TxtPatientSurname.KeyPress += TxtKeyPress;
    TxtServiceName.TextChanged += (s, e) => TxtTextChanged();
    TxtPatientName.TextChanged += (s, e) => TxtTextChanged();
    TxtPatientSurname.TextChanged += (s, e) => TxtTextChanged();
    TxtTextChanged();
}
```

Método “UpdateControlState” que se encarga de habilitar o deshabilitar los botones del formulario en base a una condición

```
private void UpdateControlState(Func<bool> condition)
{
    BtnSearchById.Enabled = BtnSearchByMonth.Enabled = BtnUpdate.Enabled =
    BtnDelete.Enabled = BtnShowAll.Enabled = BtnPrevious.Enabled = BtnNext.Enabled = condition();
}
```

Método “Load” del formulario, se marca como seleccionado en el combobox el primer mes y se deshabilitan los controles de Eliminar, Actualizar, navegar por la tabla si no hay registros al arrancar la app

```
private void FrmDentalClinic_Load(object sender, EventArgs e)
{
    CmbBoxMonths.SelectedIndex = 0;
    UpdateControlState(() => DgViewAppointments.Rows.Count > 0);
}
```

Método “RefreshDataGridView” para actualizar el contenido de la tabla cada vez que se realice una operación en los registros

```
private void RefreshDataGridView()
{
    DgViewAppointments.DataSource = _appointmentModel.GetAll();
    UpdateControlState(() => DgViewAppointments.Rows.Count > 0);
}
```

Evento Click del botón “Guardar”

```
private void BtnSave_Click(object sender, EventArgs e)
{
    var appointment = new Appointment()
    {
        Id = Convert.ToInt32(TxtNumAppointmentID.Value),
        PatientName = TxtPatientName.Text,
        PatientSurname = TxtPatientSurname.Text,
        Service = TxtServiceName.Text,
        ScheduledDate = DPickerScheduledDate.Value
    };

    (bool result, string message) tuple;
    if (_formOperation == FormOperation.Create)
        tuple = _appointmentModel.Add(appointment);
    else
        tuple = _appointmentModel.Update(appointment);

    if (!tuple.result)
    {
        MessageBox.Show(tuple.message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    if (_formOperation == FormOperation.Update)
    {
        _formOperation = FormOperation.Create;
        BtnCancel.Visible = false;
    }

    RefreshDataGridView();
    ClearForm();
}
```

Evento Click del botón “Eliminar”

```
private void BtnDelete_Click(object sender, EventArgs e)
{
    var selectedRowIndex = DgViewAppointments.SelectedRows[0].Index;
    if (DgViewAppointments.Rows[selectedRowIndex].DataBoundItem is not Appointment selectedAppointment)
    {
        MessageBox.Show("No se ha seleccionado ninguna cita para eliminar", "Atención",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return;
    }

    var dlgResult = MessageBox.Show($"Seguro que desea eliminar la cita para " +
        $"{selectedAppointment.PatientName} {selectedAppointment.PatientSurname}?",
        "Atención", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (dlgResult == DialogResult.No)
        return;

    var (result, message) = _appointmentModel.Delete(selectedAppointment.Id);
    if (!result)
    {
        MessageBox.Show(message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    RefreshDataGridView();
}
```

Evento Click del botón “Actualizar”

```
private void BtnUpdate_Click(object sender, EventArgs e)
{
    var selectedRowIndex = DgViewAppointments.SelectedRows[0].Index;
    if (DgViewAppointments.Rows[selectedRowIndex].DataBoundItem is not Appointment selectedAppointment)
    {
        MessageBox.Show("No se ha seleccionado ninguna cita para actualizar", "Atención",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return;
    }

    TxtNumAppointmentID.Value = selectedAppointment.Id;
    TxtNumAppointmentID.ReadOnly = true;
    TxtServiceName.Text = selectedAppointment.Service;
    DPickerScheduledDate.Value = selectedAppointment.ScheduledDate;
    TxtPatientName.Text = selectedAppointment.PatientName;
    TxtPatientSurname.Text = selectedAppointment.PatientSurname;

    _formOperation = FormOperation.Update;
    UpdateControlState(() => false);
    BtnCancel.Visible = true;
}
```

Evento Click del botón “Cancelar”

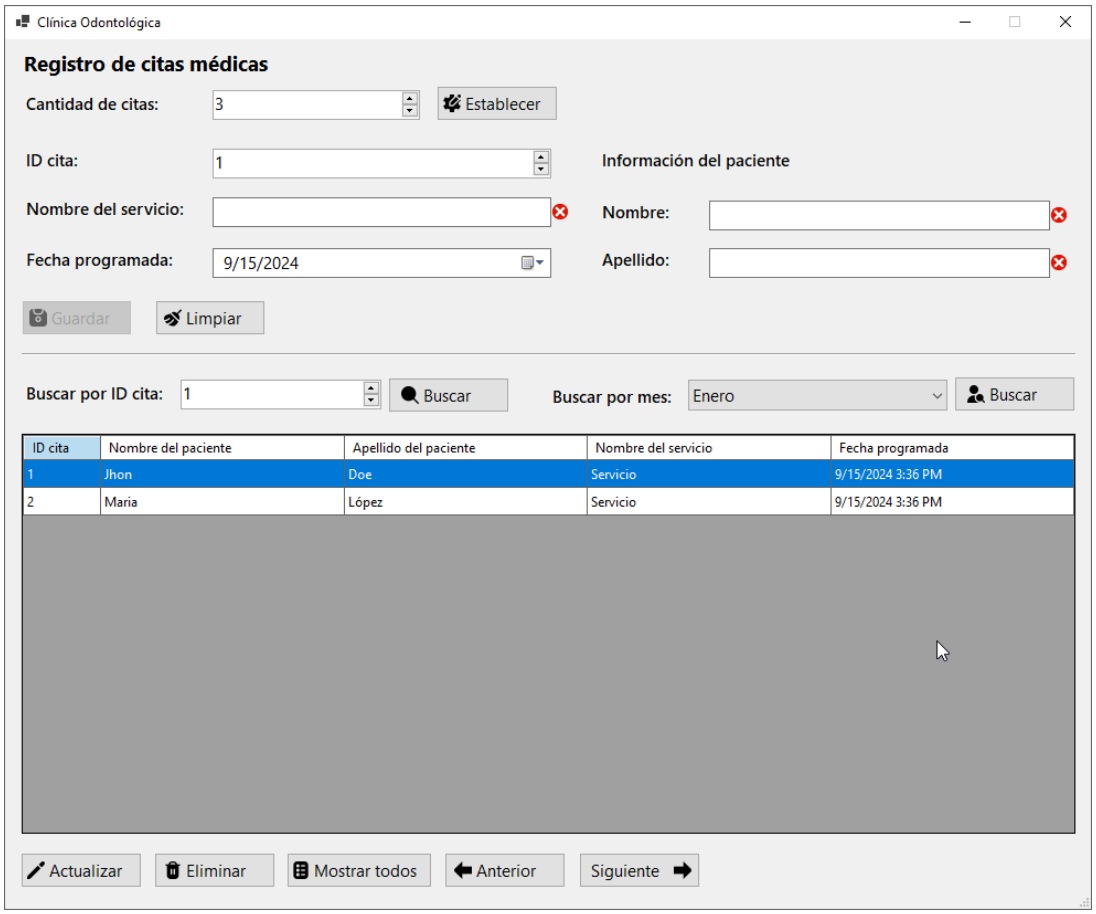
```
private void BtnCancel_Click(object sender, EventArgs e)
{
    if (_formOperation == FormOperation.Create)
        return;

    var dlgResult = MessageBox.Show("Seguro que desea cancelar la actualización de la cita?",
        "Atención", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (dlgResult == DialogResult.No)
        return;

    _formOperation = FormOperation.Create;
    BtnCancel.Visible = false;
    RefreshDataGridView();
    ClearForm();
}
```

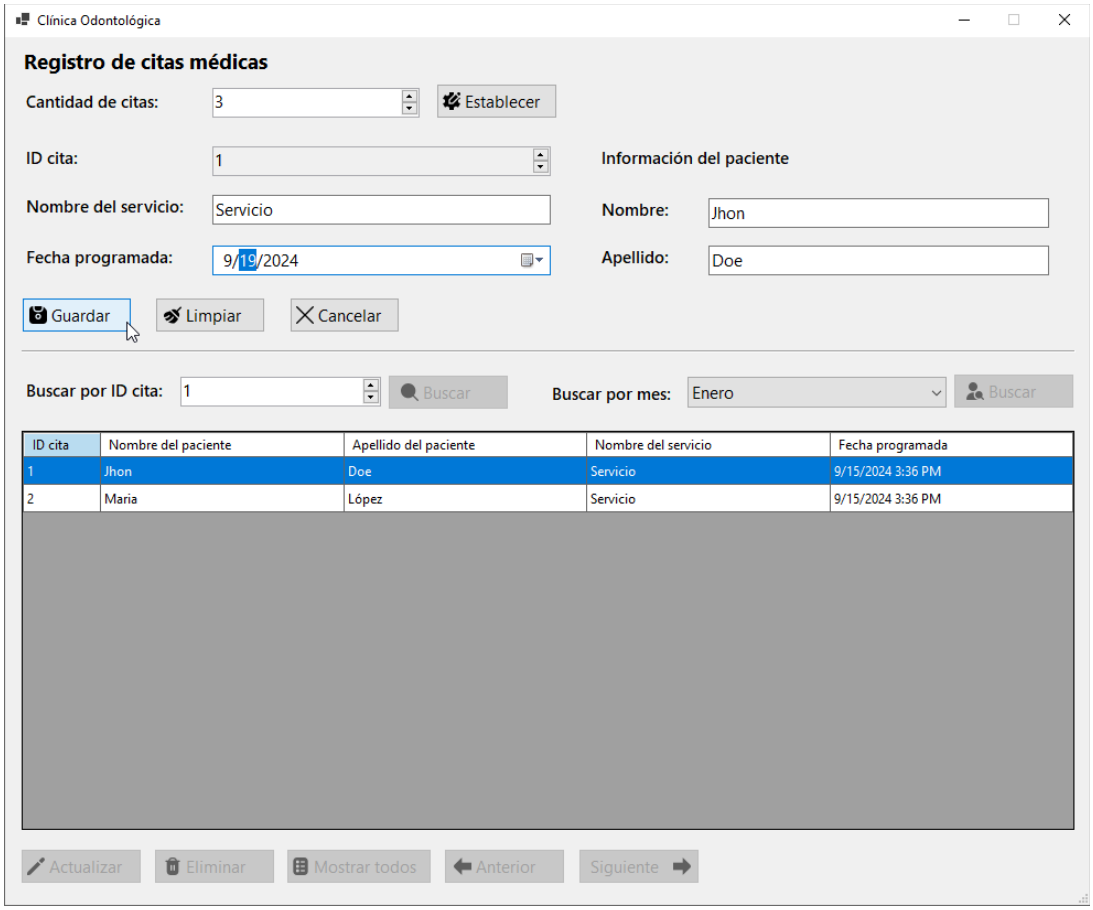
Resumen de todos los métodos del formulario

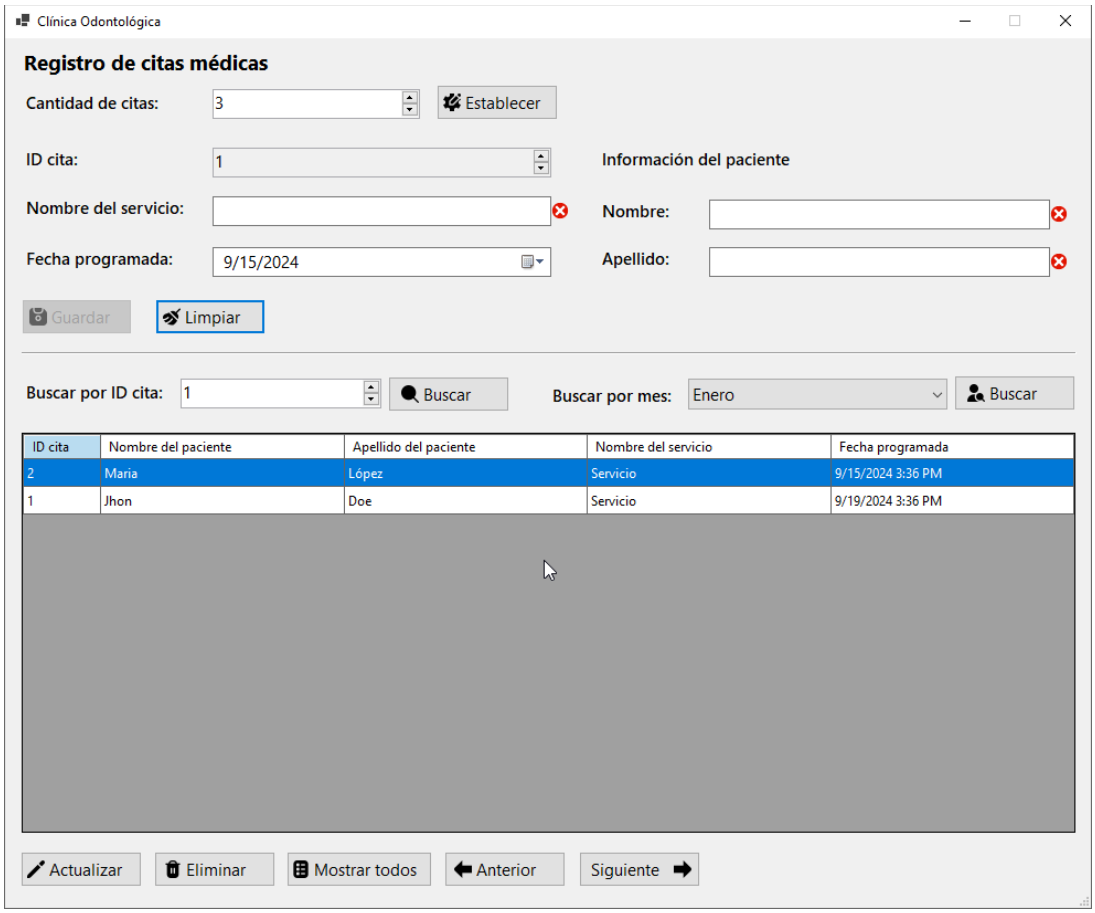
```
namespace AEDLaboratorio2.Views
{
    public partial class FrmDentalClinic : Form
    {
        private readonly AppointmentModel _appointmentModel = new();
        private FormOperation _formOperation = FormOperation.Create;
        private bool _isValidForm;

        public FrmDentalClinic()...

        private void BtnSetSize_Click(object sender, EventArgs e)...

        private void BtnClearForm_Click(object sender, EventArgs e) => ClearForm();

        private void BtnSave_Click(object sender, EventArgs e)...

        private void TxtKeyPress(object? sender, KeyPressEventArgs e)...

        private void TxtTextChanged()...

        private bool ValidateTextBox(TextBox textBox, string errorMessage)...

        private (bool isServiceValid, bool isPatientNameValid, bool isPatientSurnameValid) ValidateTextBoxes()...
```

```
        private void ClearForm()...
        private void BtnPrevious_Click(object sender, EventArgs e)...
        private void BtnNext_Click(object sender, EventArgs e)...
        private void BtnShowAll_Click(object sender, EventArgs e) => RefreshDataGridView();
        private void BtnDelete_Click(object sender, EventArgs e)...
        private void RefreshDataGridView()...
        private void BtnUpdate_Click(object sender, EventArgs e)...
        private void UpdateControlState(Func<bool> condition)...
        private void FrmDentalClinic_Load(object sender, EventArgs e)...
        private void BtnCancel_Click(object sender, EventArgs e)...
        private void BtnSearchById_Click(object sender, EventArgs e)...
        private void BtnSearchByMonth_Click(object sender, EventArgs e)...
    }
}
```

Ejecución del programa




Clínica Odontológica
Registro de citas médicas
Cantidad de citas:
3
Establecer
ID cita:
1
Información del paciente
Nombre del servicio:
Servicio
Nombre:
Jhon
Fecha programada:
9/19/2024
Apellido:
Doe
Guardar
Limpiar
Cancelar
Buscar por ID cita:
1
Buscar
Buscar por mes:
Enero
Buscar
ID cita
Nombre del paciente
Apellido del paciente
Nombre del servicio
Fecha programada
1
Jhon
Doe
Servicio
9/15/2024 3:36 PM
2
Maria
López
Servicio
9/15/2024 3:36 PM
Actualizar
Eliminar
Mostrar todos
Anterior
Siguiente


Clínica Odontológica
Registro de citas médicas
Cantidad de citas:
3
Establecer
ID cita:
1
Información del paciente
Nombre del servicio:
Nombre:
Fecha programada:
9/15/2024
Apellido:
Guardar
Limpiar
Buscar por ID cita:
1
Buscar
Buscar por mes:
Enero
Buscar
ID cita
Nombre del paciente
Apellido del paciente
Nombre del servicio
Fecha programada
2
Maria
López
Servicio
9/15/2024 3:36 PM
1
Jhon
Doe
Servicio
9/19/2024 3:36 PM
Actualizar
Eliminar
Mostrar todos
Anterior
Siguiente