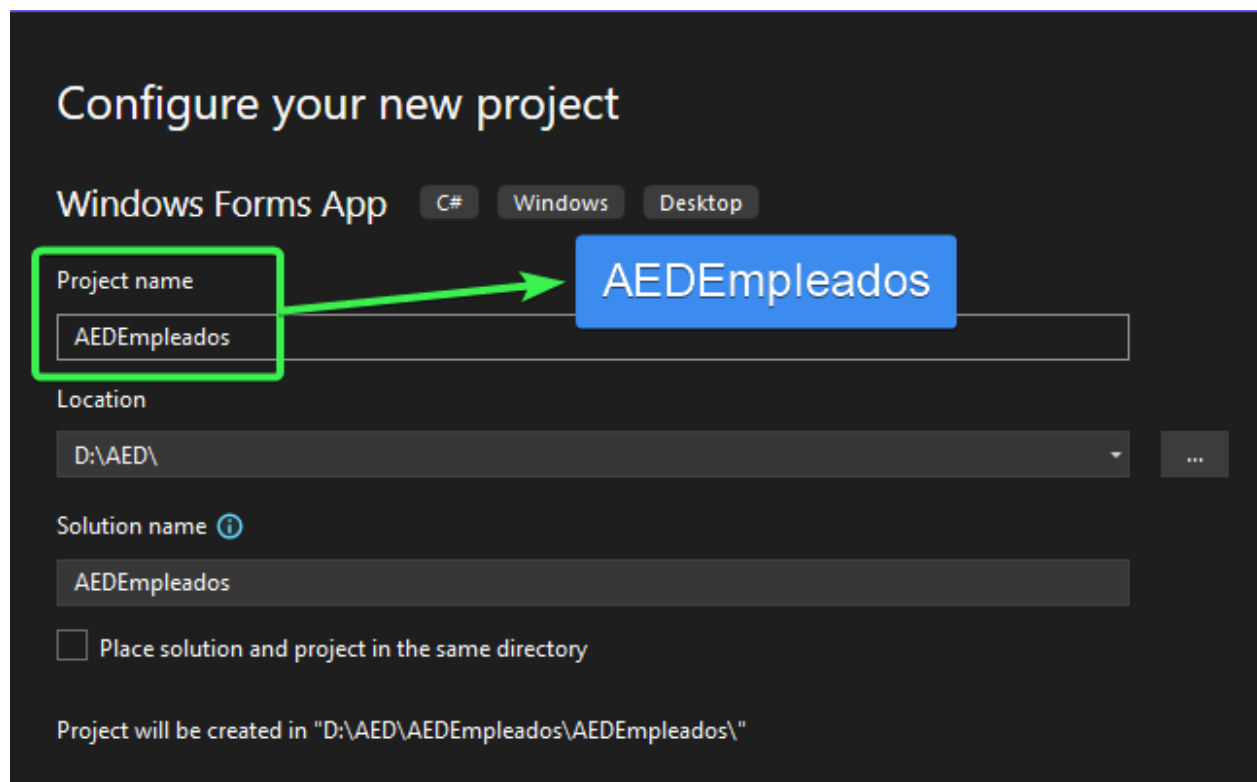
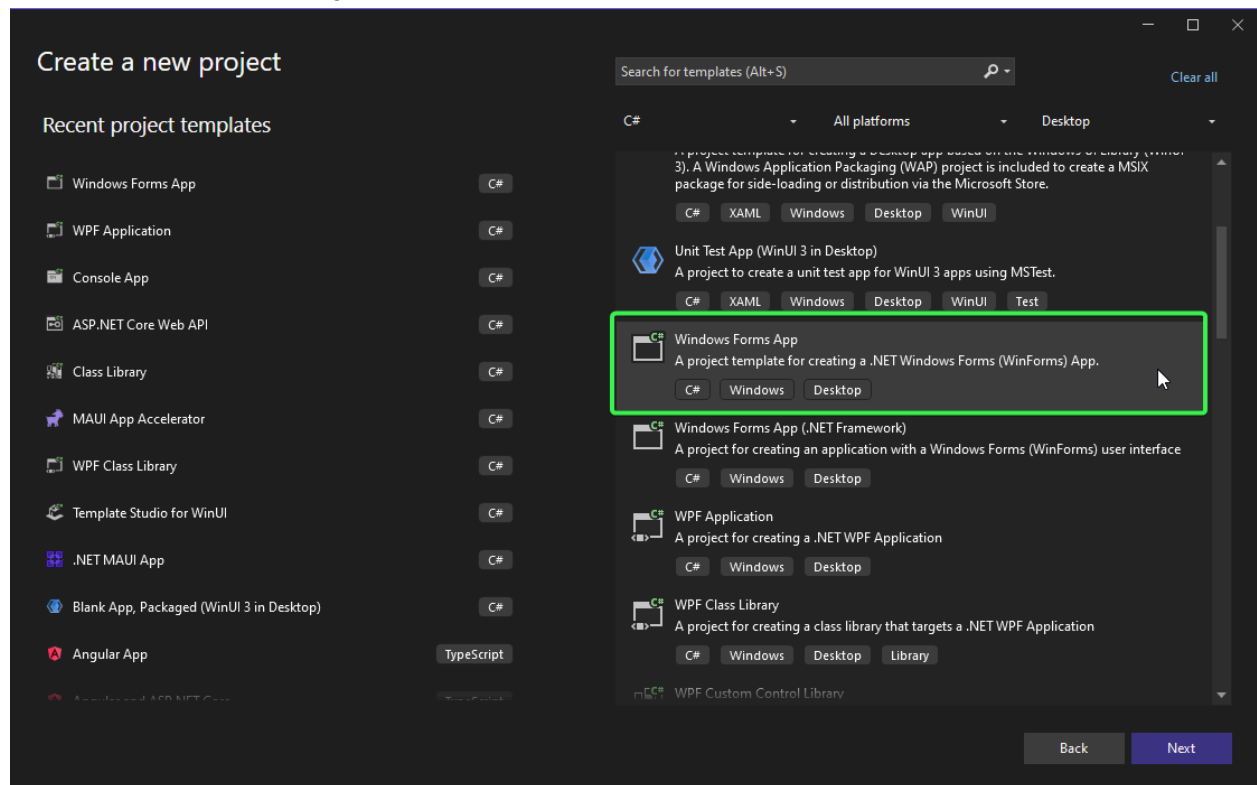
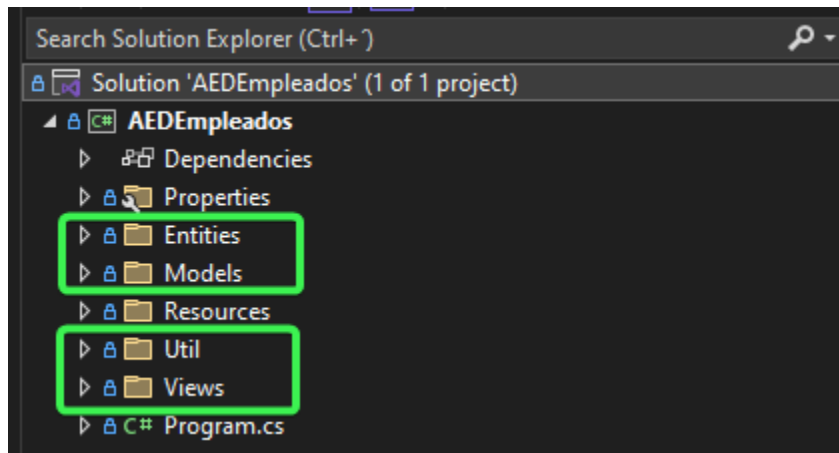


# Sistema para registro y aumento de salario de los empleados

## 1. Creación del proyecto en Windows Forms



## 2. Estructura del proyecto a utilizar:



Crear cada una de las carpetas que se muestran en la imagen para una mejor organización del código a la hora de trabajar, separando la lógica de negocio de las entidades e interfaz de usuario, a continuación una breve explicación:

**Entities (Entidades):** Representa las entidades de la aplicación, es decir, los objetos que forman parte del dominio del problema. Estos son los datos con los que trabajaremos y que a su vez dan forma al cómo se realizan cada una de las operaciones [CRUD](#) dentro de la aplicación.

**Models (Modelos):** La carpeta Models es donde se maneja la lógica de negocio y las operaciones relacionadas con los datos. A diferencia de las entidades, que son solo representaciones de datos, los modelos contienen la lógica que define cómo interactuar con esas entidades.

**Util (Utilidades):** La carpeta Util contiene funciones y clases de utilidad que no pertenecen directamente a una entidad o modelo, pero que son necesarias para el funcionamiento de la aplicación. Estas clases proporcionan métodos que pueden ser reutilizados en diferentes partes del proyecto.

**Views (Vistas):** La carpeta Views contiene la interfaz de usuario de tu aplicación, en este caso, los formularios de Windows Forms. Aquí es donde los usuarios interactúan directamente con la aplicación. Las vistas son responsables de mostrar los datos y de recibir la entrada del usuario (input).

*Para más información podemos visitar:*

- Patrón de diseño MVC

[Model-view-controller - Wikipedia](#)

- Diferencia entre Entity y Objeto

[Is there a difference between an Entity and a object? - Stack Overflow](#)

### 3. Creación de las entidades de la aplicación

Dentro de la carpeta Entities vamos a crear una clase llamada **Employee** y a su vez un enum llamado **Sex**. (Para crear enums en Visual Studio, creamos una clase llamada **Sex** y una vez generado el archivo Sex.cs reemplazamos class por enum)

#### Clase **Employee**

```
1  namespace AEDEmpleados.Entities
2  {
3      public class Employee
4      {
5          public string? NationalID { get; set; }
6          public string? Name { get; set; }
7          public string? Surname { get; set; }
8          public decimal Salary { get; set; }
9          public int Childrens { get; set; }
10         public Sex Sex { get; set; }
11         public byte[]? Photo { get; set; }
12     }
13 }
14
```

#### Enum **Sex**

```
1  namespace AEDEmpleados.Entities
2  {
3      public enum Sex
4      {
5          Male, Female
6      }
7  }
```

### 4. Creación del modelo de datos para los empleados

Dentro de la carpeta Models vamos a crear una clase llamada **EmployeeModel** la cual va a contener toda la lógica de acceso a los datos de nuestra aplicación y esta a su vez será compartida para su uso desde cualquier otra parte (formularios por ejemplo). Es una capa que se encarga de abstraer toda la manipulación de los datos (crear, actualizar, eliminar, buscar, realizar cálculos, etc) y solo expone las métodos para interactuar con el arreglo (**Employee[]**) más no permite que se pueda acceder a él desde fuera, todo para mantener la integridad de los datos.

```

1  using AEDEmpleados.Entities;
2
3  namespace AEDEmpleados.Models
4  {
5      public class EmployeeModel
6      {
7          // Arreglo privado que almacena los empleados
8          private Employee[] _employees = [];
9
10         // Atributos privados para controlar el tamaño total
11         // y la cantidad de empleados
12         private int _size = 0, _quantity = 0;
13     }
14 }

```

Ahora creamos un método que nos permita agregar nuevos empleados al arreglo

```

public void Add(Employee employee)
{
    /*
     * if: Si no queda espacio disponible en el arreglo, se redimensiona aumentando
     * un espacio más de su capacidad actual y agregando el nuevo empleado al final
     *
     * else: Si el arreglo no está lleno, se utilizan esos espacios vacíos hasta que
     * eventualmente se llene y toque redimensionarlo nuevamente
     *
     * Finalmente aumentar en uno la cantidad de empleados registrados
     */
    if (_quantity >= _size)
    {
        Array.Resize(ref _employees, ++_size);
        _employees[_size - 1] = employee;
    }
    else
        _employees[_quantity] = employee;

    ++_quantity;
}

```

De esta forma solo redimensionamos el arreglo cuando es necesario, ya que en la siguiente situación donde se podrían registrar 2 empleados seguidos, se elimina uno de ellos (cualquiera) y luego se registra un tercer empleado, si no se implementara esta lógica al registrar el tercer empleado estaríamos reservando más memoria de la que necesitamos de forma innecesaria, ya que el arreglo tenía espacio disponible para almacenar ese tercer empleado sin necesidad de reservar un espacio más en memoria, ese espacio disponible sería el espacio ocupado por el empleado anterior que luego se eliminó.

Ahora implementamos el método para poder actualizar un empleado por medio de su cédula de identidad la cual tendremos que garantizar de que sea única (esa validación se realiza en otro lado)

```
public bool Update(Employee employee)
{
    /*
     * if: Verificar si se proporcionó una cédula de identidad
     *
     * Array.FindIndex: Busca el índice del empleado en el array que coincide con la
     * cédula proporcionada, si no se encuentra el método FindIndex devuelve -1,
     * por lo tanto retornamos false ya que no hay un empleado a actualizar
     *
     * Si se encuentra se actualiza el empleado de la posición index y se retorna true
     * que indica que el empleado se ha actualizado exitosamente
     */
    if (string.IsNullOrEmpty(employee.NationalID)) return false;

    var index = Array.FindIndex(_employees, e => employee.NationalID.Equals(e.NationalID));
    if (index < 0) return false;

    _employees[index] = employee;

    return true;
}
```

Lógica para eliminar un empleado por medio de su cédula de identidad si existe

```
public bool Delete(string nationalID)
{
    /*
     * Array.FindIndex: Busca el índice del empleado en el array que coincide con la
     * cédula proporcionada, si no se encuentra el método FindIndex devuelve -1,
     * por lo tanto retornamos false ya que no existe el empleado que se desea eliminar
     *
     * Si se encuentra entonces se desplazan los elementos del arreglo hacia la izquierda
     * empezando en la posición donde se encuentra el empleado a eliminar
     *
     * Se reduce el contador de empleados registrados _quantity y se retorna true indicando
     * que el empleado se ha eliminado exitosamente
     */
    var index = Array.FindIndex(_employees, e => nationalID.Equals(e.NationalID));
    if (index < 0) return false;

    for (int i = index; i < _employees.Length - 1; i++)
        _employees[i] = _employees[i + 1];

    --_quantity;

    return true;
}
```

Lógica del método GetAll para obtener todos los empleados activos, para ello devolvemos un arreglo que va desde el primer empleado hasta el que está en la posición \_quantity que sería el último empleado registrado, ignorando los datos basura que pueden haber al final del arreglo y si está vacío simplemente se ignoran esos espacios y se devuelven igualmente los registrados

```
public Employee[] GetAll() => _employees.Take(_quantity).ToArray();
```

Ahora agregamos un método que nos permita obtener el salario promedio de todos los empleados registrados de la siguiente forma:

```
public decimal AverageSalary() => _employees.Average(e => e.Salary);
```

Luego agregaremos un método que nos permita realizar un aumento de salario a los empleados, dado un porcentaje y una condición que nosotros deseemos:

```
public int SalaryIncrease(decimal percentage, Predicate<Employee> condition)
{
    int count = 0;
    foreach (var emp in _employees)
    {
        if (condition(emp))
        {
            emp.Salary += emp.Salary * percentage;
            ++count;
        }
    }

    return count;
}
```

El método SalaryIncrease recibe por parámetro primeramente el porcentaje que se aplicará al aumento y un [Predicate](#), delegado o condición booleana a aplicar sobre la lista, la cual se evaluará elemento a elemento y todos los que cumplan la condición se les dará el aumento con el porcentaje proporcionado. Finalmente el método retorna un entero que indica la cantidad de empleados que se vieron beneficiados por el aumento de salario.

Para más información podemos visitar:

[Predicate<T> Delegate \(System\) | Microsoft Learn](#)

[What is a predicate in c#? - Stack Overflow](#)

Adicionalmente agregamos un método que nos permite buscar empleados por medio de su cédula de identidad que nos servirá para realizar la validación de cédulas ya existentes cuando registremos un nuevo empleado desde el formulario. Tomamos sólo los empleados activos (hasta **\_quantity**) y comparamos buscando uno que coincida con la cédula proporcionada).

```
public Employee? Find(string nationalID) =>
    _employees.Take(_quantity).FirstOrDefault(e => nationalID.Equals(e.NationalID));
```

Con esto concluimos los métodos que tendrá la clase **EmployeeModel**, pero aún queda algo pendiente y es lo siguiente...

```
EmployeeModel employeeModel = new();
EmployeeModel employeeModel2 = new();
EmployeeModel employeeModel3 = new();
// Más instancias aquí...
// ....
```

```
private Employee[] _employees = [];  
private int _size = 0, _quantity = 0;
```

Cada vez que creamos una instancia de la clase **EmployeeModel** estamos creando un nuevo arreglo vacío de tipo empleado y lo que queremos es que nuestro modelo de datos sea el mismo para cualquier otra clase o formulario que vaya a utilizarla sin importar cuantas instancias se puedan crear o desde donde se acceda, para ello tenemos algunas soluciones:

### 1. Patrón de diseño Singleton

Este patrón de diseño restringe la creación de instancias de la clase **EmployeeModel** a una única instancia que además será compartida por todos los que deseen usar la clase, a su vez proporciona un punto de acceso global para todos y esta sería una forma de mantener un solo arreglo de tipo **Employee[]** en toda la ejecución del programa.

Link: [Singleton pattern - Wikipedia](#)

### 2. Declarar el arreglo Employee[] como estático

Al hacer esto el arreglo de empleados ahora pertenece a la clase y no a sus instancias, por lo tanto se podrán crear muchas instancias de **EmployeeModel** pero todas tendrán la misma referencia del arreglo **Employee[]** ya que es un campo de la clase.

Link: [Static Classes and Static Class Members - C# | Microsoft Learn](#)

### 3. Declara la clase EmployeeModel como estática

Una clase estática no nos permite crear instancias de ella por lo tanto todos sus miembros o atributos le pertenecen a la clase y de esta forma también podemos garantizar de que habrá una sola instancia del arreglo **Employee[]** en toda la aplicación.

Link: [Static Classes and Static Class Members - C# | Microsoft Learn](#)

### 4. Usar una clase estática separada para almacenar los empleados

Una última alternativa sería crear una clase aparte que sea estática para que tenga los beneficios explicados en el punto anterior y simplemente desde **EmployeeModel** con una referencia a esta clase manipular el arreglo de **Employee[]**.

En este caso, implementaremos el patrón de diseño **Singleton** en la clase **EmployeeModel** para continuar con la guía, para ello hacemos lo siguiente:

```
1 namespace AEDEmpleados.Models
2 {
3     public class EmployeeModel
4     {
5         // Campo estático y readonly que almacena una instancia única
6         // de la clase EmployeeModel.
7         // 'readonly' significa que este campo solo puede ser
8         // asignado una vez, ya sea en su declaración o en un
9         // constructor, en este caso un constructor estático.
10        private static readonly EmployeeModel _instance = new();
11
12        // Propiedad pública para acceder a la instancia única.
13        // Esta propiedad expone la instancia estática _instance
14        // de la clase EmployeeModel, de modo que cualquier clase
15        // que quiera usar EmployeeModel debe acceder a esta instancia.
16        public static EmployeeModel Instance { get => _instance; }
17
18        // Constructor privado para evitar que otras clases puedan crear
19        // nuevas instancias de EmployeeModel.
20        // Esto es clave para el patrón Singleton, que asegura que solo
21        // haya una instancia de esta clase.
22        private EmployeeModel()
23        { }
24
25        // Resto del código...
26    }
27 }
```

Con esto ya hemos finalizado nuestro modelo de datos.



## 5. Diseño del formulario principal o FrmMain

AED Registro de Empleados - ZM1-CO

Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
------	---------------------	---------	-----------	---------	-------	------

Nuevo Editar Eliminar Aumento de salario Anterior Siguiente

AED Registro de Empleados - ZM1-CO

Registro de Empleados → Label

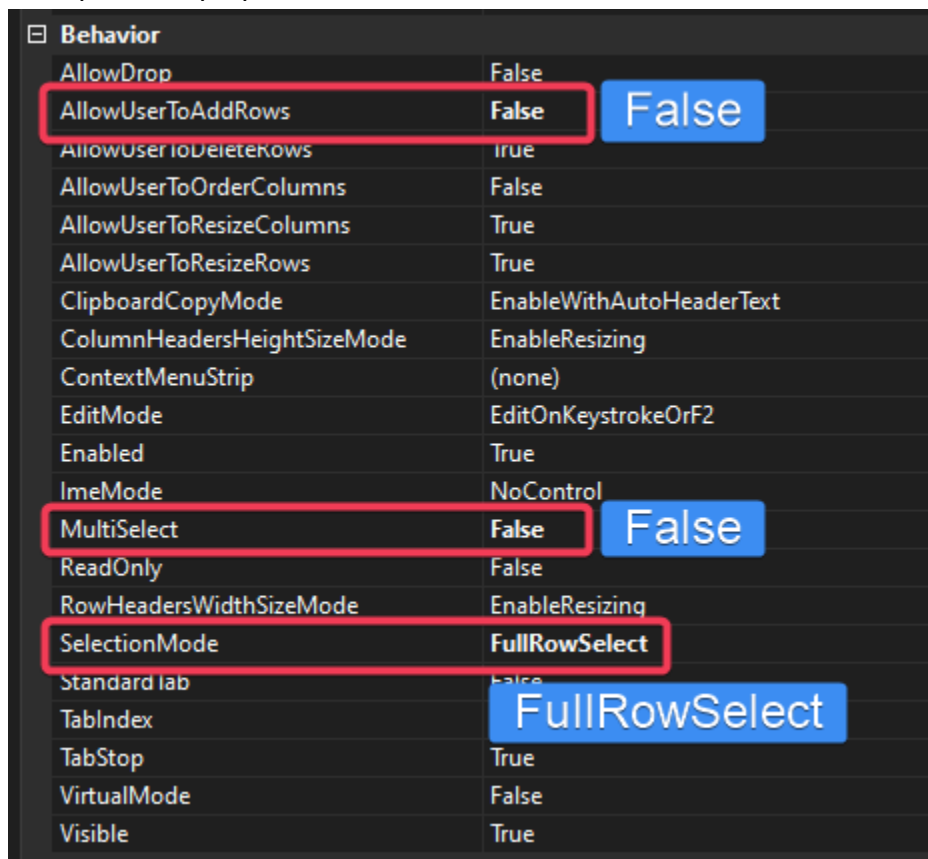
Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
------	---------------------	---------	-----------	---------	-------	------

→ DataGridView

Nuevo Editar Eliminar Aumento de salario Anterior Siguiente → Botones

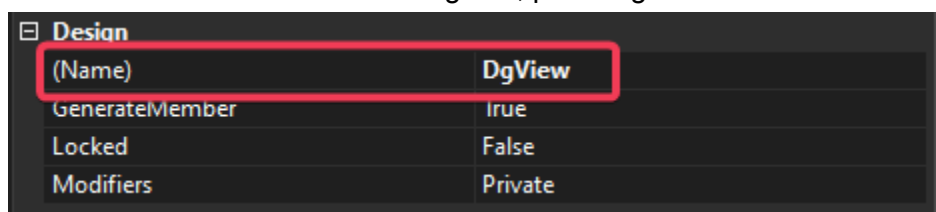
El diseño de cada componente **Label**, **Buttons** y **DataGridView** se dejan a gusto del estudiante, nos centramos solamente en las propiedades funcionales del **DataGridView** que nos permitirán mostrar datos en ella serían las siguientes:

En el panel de propiedades del **DataGridView**, en la sección **Behavior**

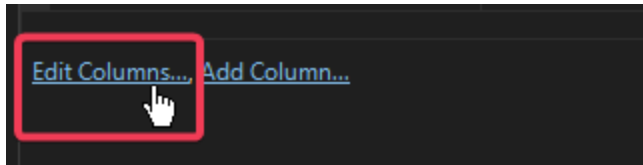


- **AllowUserToAddRows:** Eliminará la fila vacía que aparece al final de los registros
- **MultiSelect:** Hará que solamente se pueda seleccionar una fila a la vez
- **SelectionMode:** Hará que al hacer click sobre cualquier fila de la tabla se seleccione toda la fila y no solo la celda donde se hizo click

El nombre del DataGridView es a gusto, para la guía le llamaremos de la siguiente forma

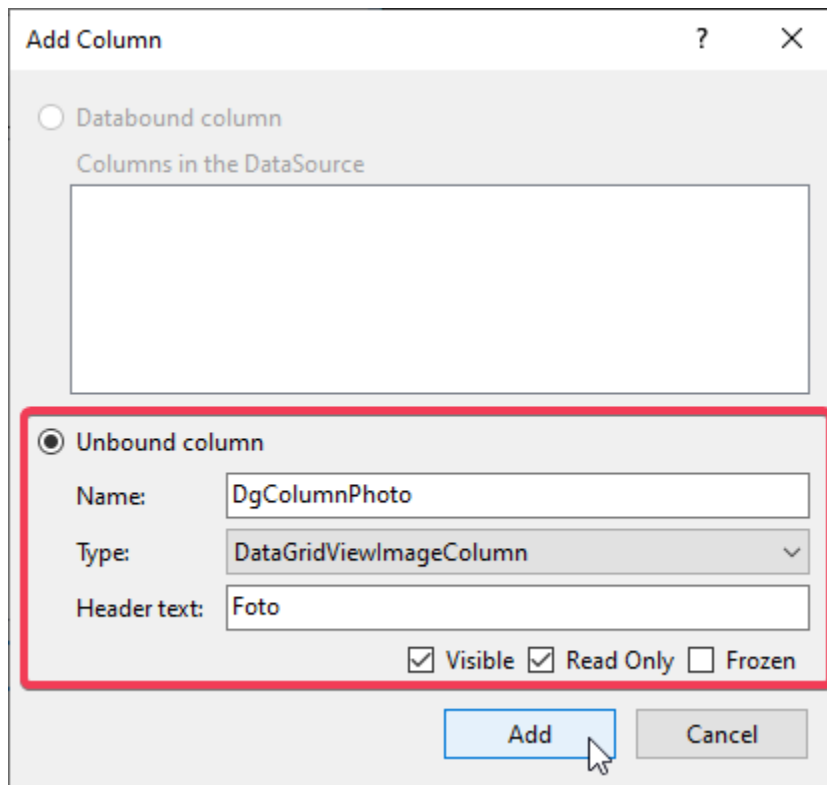


Ahora vamos a definir el número de columnas, el nombre de cada una de ellas, el tipo de dato de cada columna y finalmente la propiedad de nuestra entidad que la utilizara, al final del panel de propiedades tenemos la opción **Edit Columns...**

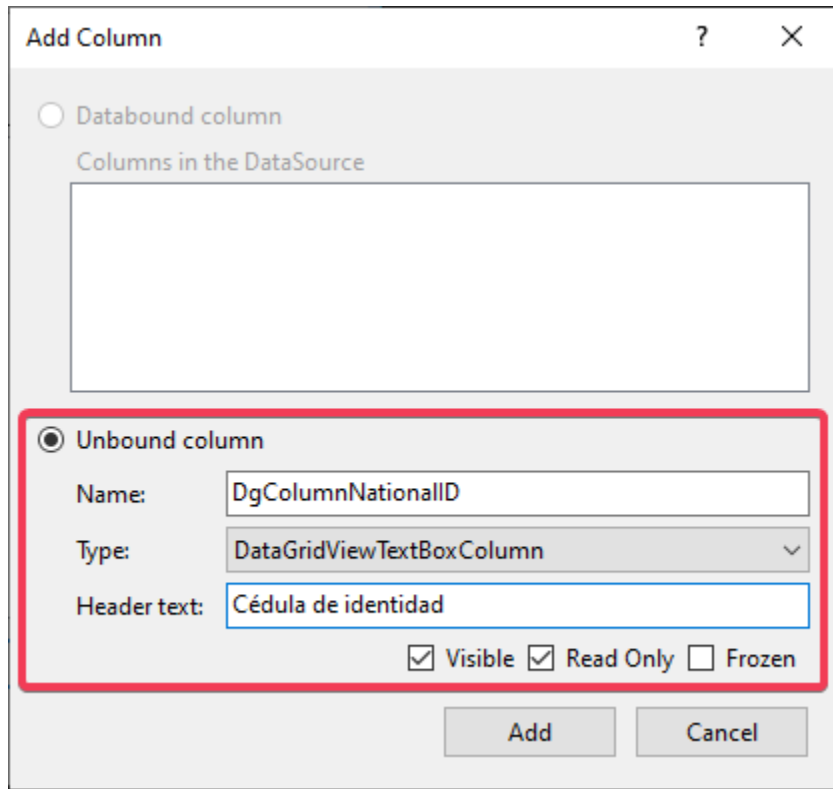


En la siguiente ventana de diálogo agregaremos cada una de las siguientes columnas:

### Columna Foto (Type DataGridViewImageColumn)



## Columna Cédula de identidad, Nombres, Apellidos, Salario, Hijos, Sexo (Type DataGridViewTextBoxColumn)



Add Column

☐ Databound column

Columns in the DataSource

☒ Unbound column

Name: DgColumnNationalID

Type: DataGridViewTextBoxColumn

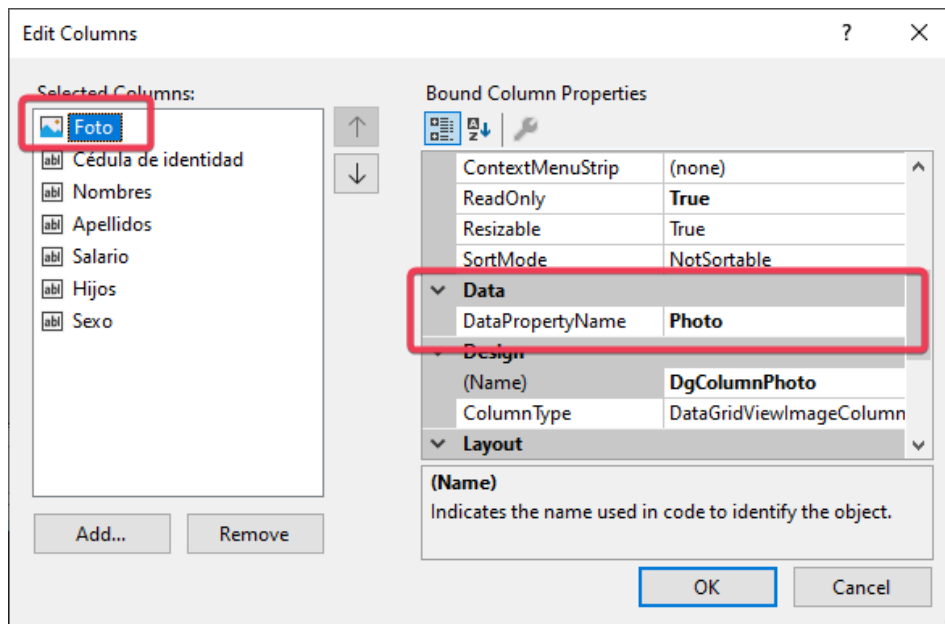
Header text: Cédula de identidad

☒ Visible ☒ Read Only ☐ Frozen

Add Cancel

En este caso el **Name** de la columna no es importante para vincularla a nuestra entidad **Employee**, es solo el nombre de la columna en el archivo **.cs** a continuación veremos cómo vincular cada columna con cada propiedad de la entidad desde el diseñador.

Una vez con las columnas ya agregadas, procedemos a buscar en las propiedades de cada columna la que dice **DataPropertyName** y ahí pondremos el nombre de la propiedad de nuestra entidad que queremos que utilice esa columna (a continuación en la imagen)



Edit Columns

Selected Columns:

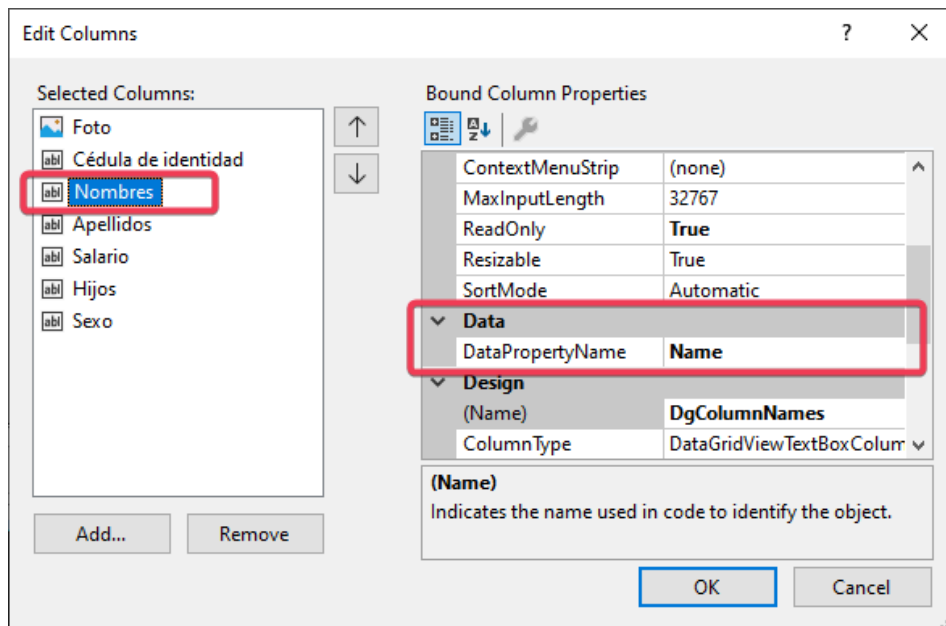
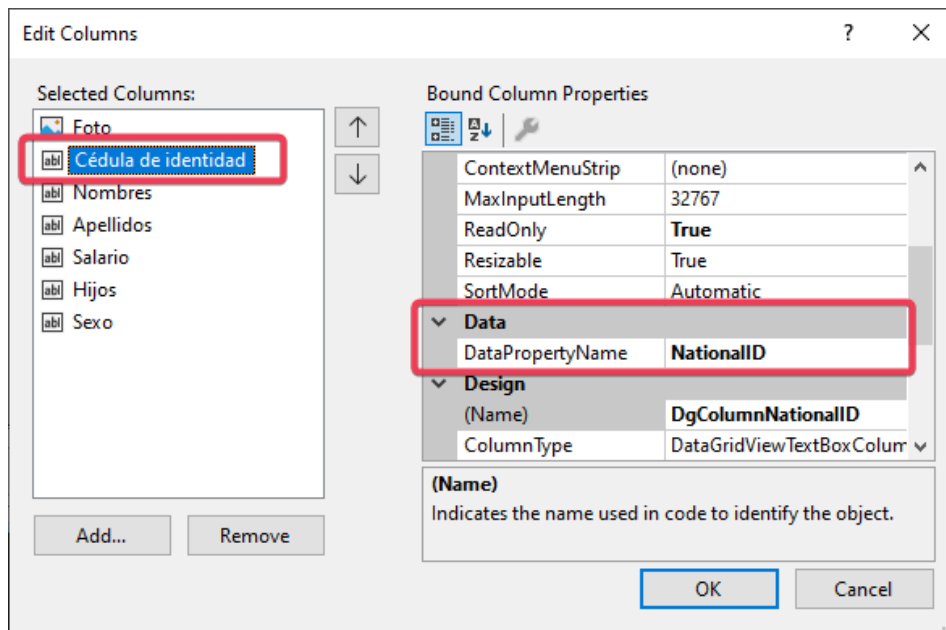
- Foto
- Cédula de identidad
- Nombres
- Apellidos
- Salario
- Hijos
- Sexo

Bound Column Properties

ContextMenuStrip	(none)
ReadOnly	True
Resizable	True
SortMode	NotSortable
<b>Data</b>	
DataPropertyName	Photo
<b>Design</b>	
(Name)	DgColumnPhoto
ColumnType	DataGridViewImageColumn
<b>Layout</b>	
(Name)	
Indicates the name used in code to identify the object.	

Add... Remove

OK Cancel



Debemos hacer lo mismo con cada una de las siguientes columnas restantes **Apellidos**, **Salario**, **Hijos** y **Sexo**.

```

public class Employee
{
    public string? NationalID { get; set; }
    public string? Name { get; set; }
    public string? Surname { get; set; }
    public decimal Salary { get; set; }
    public int Childrens { get; set; }
    public Sex Sex { get; set; }
    public byte[]? Photo { get; set; }
}

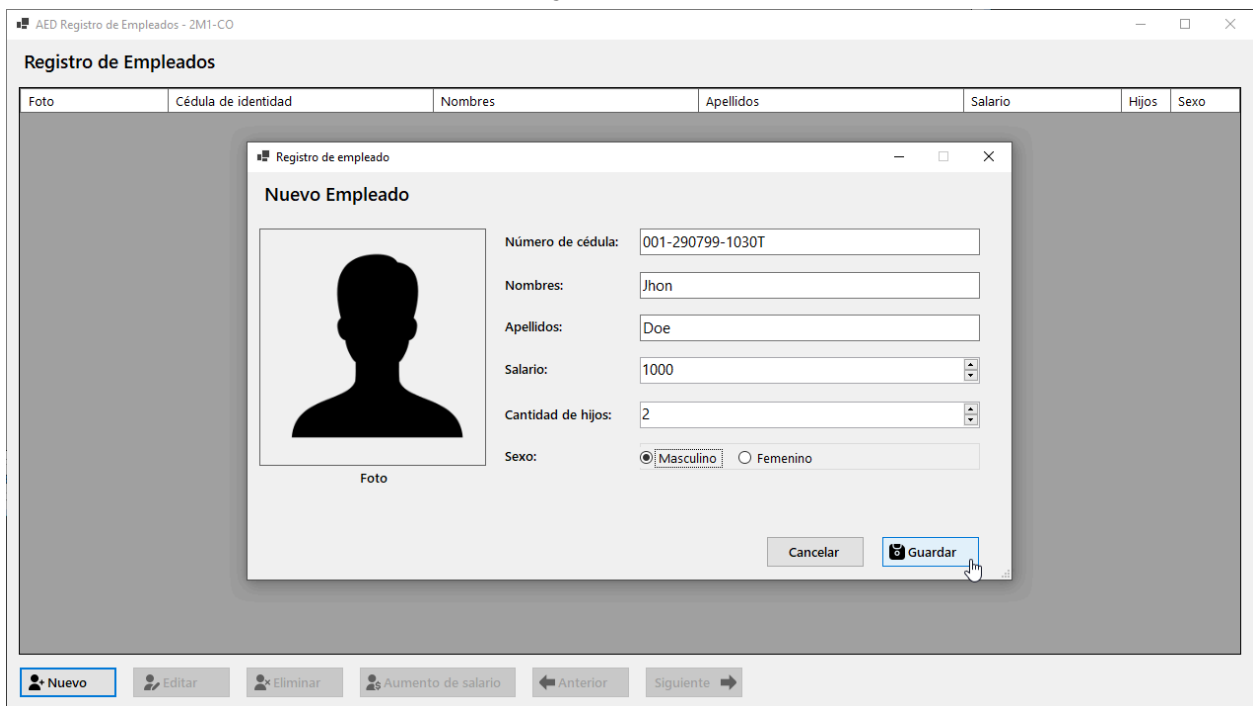
```

DataPropertyName

Bien, ahora a los botones les asignaremos un nombre y posteriormente su evento click desde el panel de propiedades.

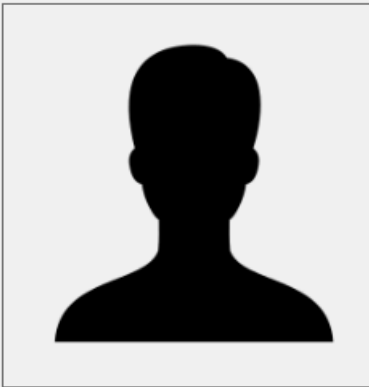


## 6. Diseño del formulario FrmEmployee



Registro de empleado

## Nuevo Empleado



Foto

Número de cédula:

Nombres:

Apellidos:

Salario:

Cantidad de hijos:

Sexo: ☐ Masculino ☐ Femenino

Cancelar Guardar

Registro de empleado

Nuevo Empleado

Foto

Número de cédula:

Nombres:

Apellidos:

Salario:

Cantidad de hijos:

Sexo: ☐ Masculino ☐ Femenino

Cancelar Guardar

Labels

MaskedTextBox

TextBoxs

NumericUpDown

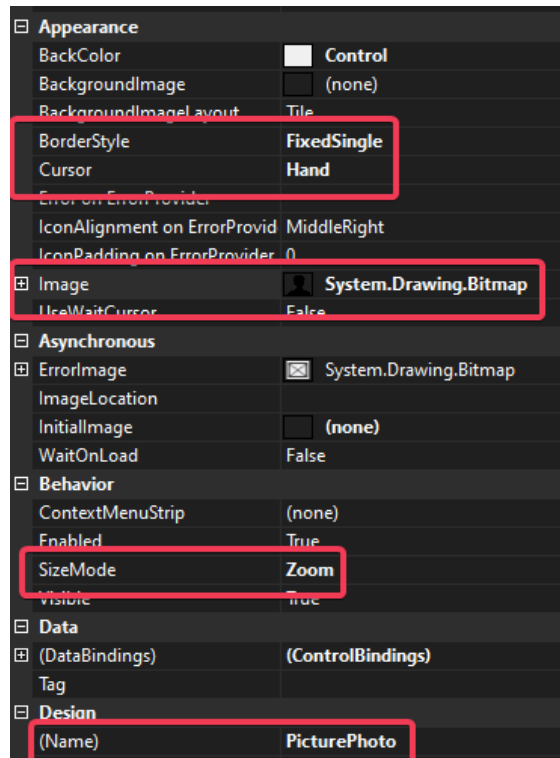
GroupBox

RadioButtons

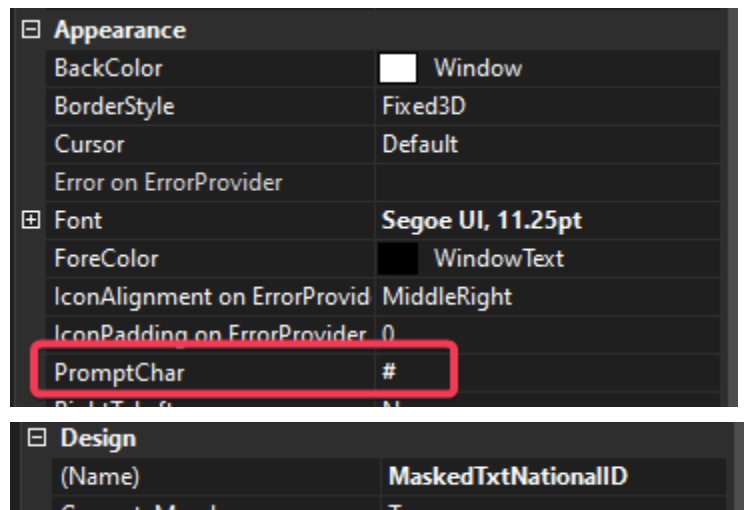
Buttons

PictureBox

Propiedades del PictureBox (Foto)



### Propiedades de MaskedTextBox (Cédula de identidad)





Behavior	
AllowDrop	False
AllowPromptAsInput	True
AsciiOnly	False
BeepOnError	False
ContextMenuStrip	(none)
Culture	en-US
CutCopyMaskFormat	IncludeLiterals
Enabled	True
HidePromptOnLeave	False
HideSelection	True
ImeMode	NoControl
InsertKeyMode	Default
Mask	000-000000-0000>L

La máscara se utiliza para restringir y validar el formato de la entrada de datos mientras el usuario escribe. La siguiente máscara se encarga de validar el formato de la cédula

**000-000000-0000>L**

**0:** Dígito 0-9

**>:** Todo lo que este despues lo pondrá en mayúscula

**L:** Una letra a-z A-Z

Para ver una lista completa de caracteres disponibles y su significado al crear una máscara:

[MaskedTextBox Class \(System.Windows.Forms\) | Microsoft Learn](#)

[MaskedTextBox.Mask Property \(System.Windows.Forms\) | Microsoft Learn](#)

### Propiedades de los TextBox (Nombres y Apellidos)

Design	
(Name)	TxtNames
Design	
(Name)	TxtSurnames

### Propiedades del NumericUpDown (Salario)

(El máximo y mínimo son valores arbitrarios, en este caso va de 0 a 1,000,000)

Data	
(DataBindings)	(ControlBindings)
DecimalPlaces	2
Increment	1
Maximum	1000000
Minimum	0
Tag	
ThousandsSeparator	False
Design	
(Name)	TxtNumericSalary

## Propiedades del NumericUpDown (Hijos)

(El máximo y mínimo son valores arbitrarios)

Data	
(DataBindings)	(ControlBindings)
DecimalPlaces	0
Increment	1
Maximum	100
Minimum	0
Tag	
ThousandsSeparator	False
Design	
(Name)	TxtNumericChildrens

## Propiedades del GroupBox que contiene los RadioButtons

Design	
(Name)	GroupRadioBtn

## Propiedades de los RadioButtons (Masculino y Femenino)

Text	Masculino
------	-----------

Design	
(Name)	RadioBtnMale

Text	Femenino
------	----------

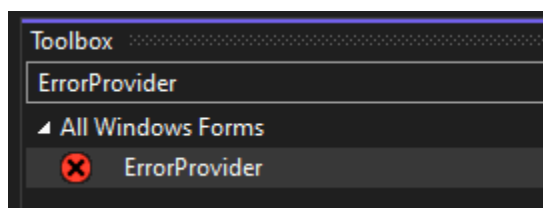
Design	
(Name)	RadioBtnFemale

## Propiedades de los Button (Cancelar y Guardar)

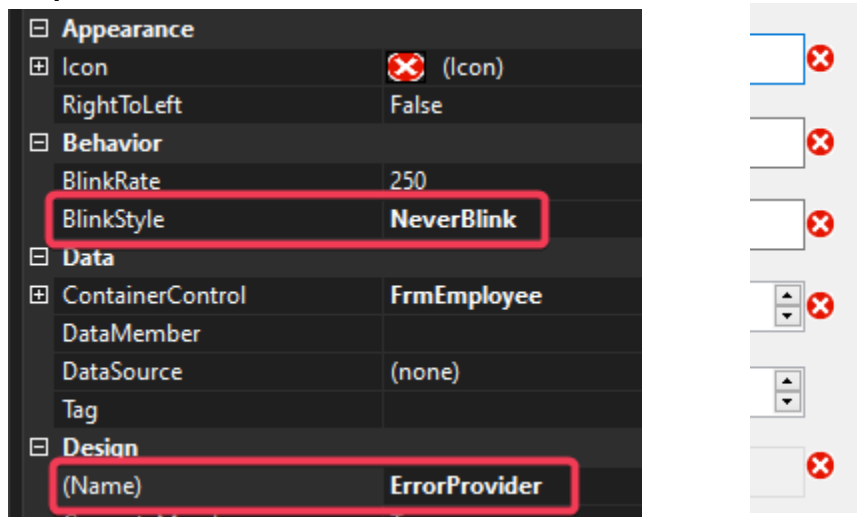
Design	
(Name)	BtnCancel

Design	
(Name)	BtnSave

Ahora para poder agregar el indicador de error a la par de cada componente, debemos arrastrar un **ErrorProvider** sobre el formulario en cualquier parte, este no se le da una ubicación desde el diseñador si no desde código programaremos la lógica para que se muestre en cada componente.

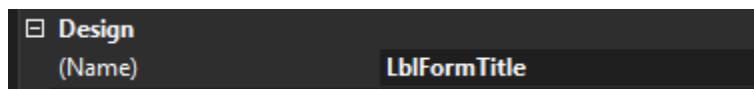
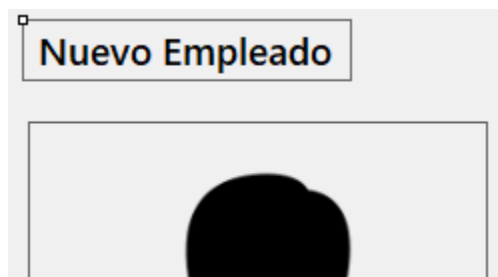


### Propiedades del ErrorProvider



\* Este no aparece aún después de modificar sus propiedades la imagen de la derecha es para ilustrar cual es el ErrorProvider y como debe de quedar al final.

### Propiedades del Label principal del formulario




El texto cambiará dependiendo el modo en que se utilice el formulario, ya sea para registrar nuevos empleados o para editar empleados ya existentes.

## 7. Diseño del formulario FrmEmployeeSalary

AED Registro de Empleados - 2M1-CO

Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
	002-290289-1020G	Jhon	Doe	1000.00	2	Male

**Aumento de salario**

Porcentaje: 0.10

Seleccionar empleados

☒ Con salario menor que el promedio  
☐ Con salario mayor que el promedio

Salario promedio: 1000.00

Cancelar Aplicar

Nuevo Editar Eliminar Aumento de salario Anterior Siguiente

**Aumento de salario**

Porcentaje: 0.10

Seleccionar empleados

☒ Con salario menor que el promedio  
☐ Con salario mayor que el promedio

Salario promedio: 1000.00

Cancelar Aplicar

**Aumento de salario**

Porcentaje: 0.10

Seleccionar empleados

☒ Con salario menor que el promedio  
☐ Con salario mayor que el promedio

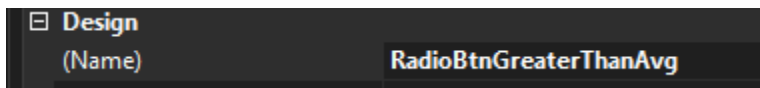
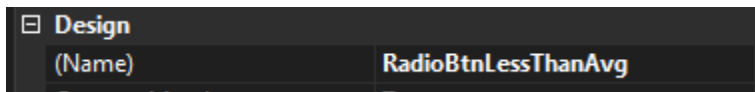
Salario promedio: 1000.00

Cancelar Aplicar

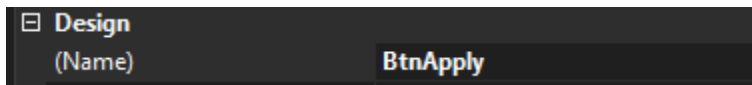
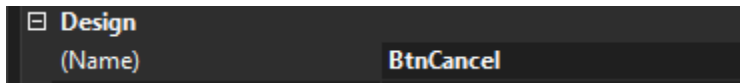
### Propiedades del NumericUpDown(Porcentaje)

Data	
(DataBindings)	(ControlBindings)
DecimalPlaces	2
Increment	0.01
Maximum	1
Minimum	0
Tag	
ThousandsSeparator	False
Design	
(Name)	TxtPercentage

### Propiedades de los RadioButtons



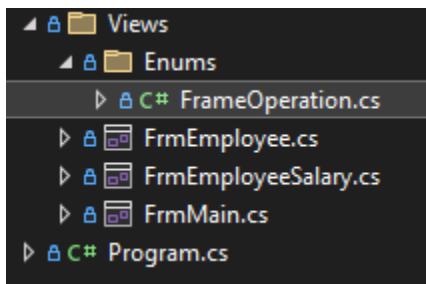
### Propiedades de los Botones (Cancelar y Aplicar)



### Propiedad del Label (Salario promedio)



## 8. Creación de enums de tipo FrameOperation en carpeta Views > Enums



```
1 namespace AEDEmpleados.Views.Enums
2 {
3     public enum FrameOperation
4     {
5         Create, Update
6     }
7 }
8
```

Por medio de este enum podremos indicar al **FrmEmployee** si queremos agregar un nuevo empleado o si queremos editar la información de uno existente cuando lo vayamos a mostrar.

## 9. Lógica del formulario FrmEmployee

Registro de empleado

Nuevo Empleado

Foto

Número de cédula: ###-#####-####

Nombres:

Apellidos:

Salario: 0.00

Cantidad de hijos: 0

Sexo: ☐ Masculino ☐ Femenino

Cancelar Guardar

### Evento KeyPress de los TextBox Nombres y Apellidos

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

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

Se valida cada vez que se pulsa una tecla para que sea una letra, espacio en blanco o Backspace (tecla eliminar).

### Evento click del botón Cancel

```
private void BtnCancel_Click(object sender, EventArgs e) => Close();
```

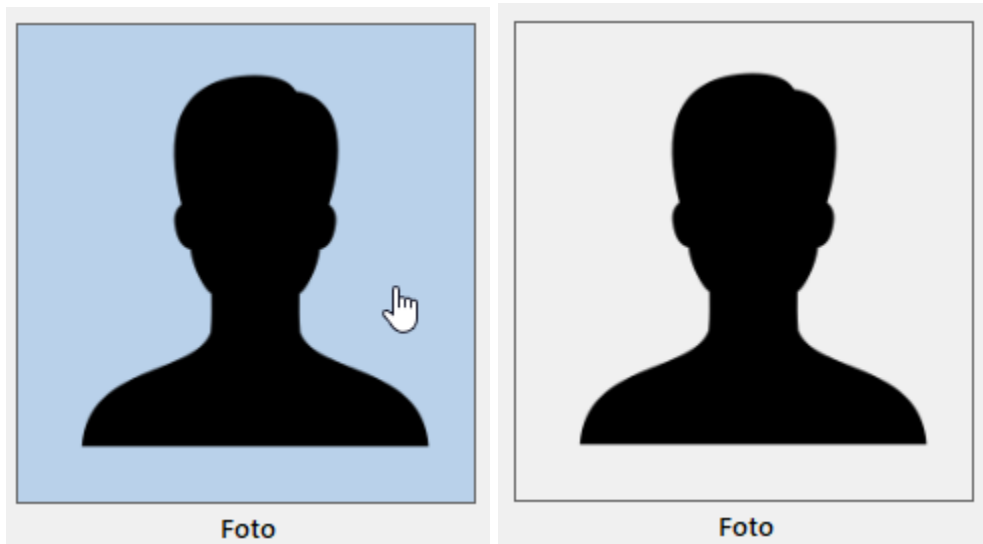
Simplemente cerramos el formulario **FrmEmployee**

### Evento MouseHover y MouseLeave del PictureBox

```
private void PicturePhoto_MouseHover(object sender, EventArgs e) =>
    PicturePhoto.BackColor = SystemColors.GradientActiveCaption;

private void PicturePhoto_MouseLeave(object sender, EventArgs e) =>
    PicturePhoto.BackColor = SystemColors.Control;
```

Se cambia color de fondo del picturebox cada vez que se pasa el mouse sobre el componente



### Evento click del PictureBox

```
private void PicturePhoto_Click(object sender, EventArgs e)
{
    var openFileDialog = new OpenFileDialog
    {
        Filter = "Image Files|*.bmp;*.jpg;*.jpeg;*.png;*.gif;*.tiff|All Files|*.*",
        Title = "Select an Image",
        Multiselect = false
    };
    if (openFileDialog.ShowDialog() == DialogResult.OK)
        PicturePhoto.Image = Image.FromFile(openFileDialog.FileName);
}
```

Abre el explorador de archivo y filtra imágenes por uno de los formatos soportados por el picture box de Windows Forms, si se selecciona una imagen se muestra en el componente.

### Variables globales del formulario FrmEmployee

```
public partial class FrmEmployee : Form
{
    private readonly FrameOperation _operation;
    private readonly Employee? _employee;
    private bool _isValidForm;
```

Creamos un método para validar la entrada de cada componente apoyandonos del **ErrorProvider** para dar feedback al usuario

Para utilizar el **ErrorProvider** sobre un componente, debemos de indicarle cual es el componente y un mensaje de error que acompañe al icono de error

#### Validación de cédula de identidad

```
private bool ValidateNationalID()
{
    if (!MaskedTextBox1.MaskCompleted)
    {
        ErrorProvider.SetError(MaskedTextBox1,
            "El número de cédula no es correcto, formato ###-#####-####X");
        return false;
    }
    ErrorProvider.SetError(MaskedTextBox1, string.Empty);
    return true;
}
```

Para validar la cédula de identidad lo hacemos por medio de la propiedad **MaskCompleted** del componente **MaskedTextBox1**

#### Validación de los nombres

```
private bool ValidateNames()
{
    if (string.IsNullOrWhiteSpace(TxtNames.Text))
    {
        ErrorProvider.SetError(TxtNames, "Los nombres son obligatorios");
        return false;
    }
    ErrorProvider.SetError(TxtNames, string.Empty);
    return true;
}
```

#### Validación de apellidos

```
private bool ValidateSurnames()
{
    if (string.IsNullOrWhiteSpace(TxtSurnames.Text))
    {
        ErrorProvider.SetError(TxtSurnames, "Los apellidos son obligatorios");
        return false;
    }
    ErrorProvider.SetError(TxtSurnames, string.Empty);
    return true;
}
```



### Validación del NumericUpDown salario

```
private bool ValidateSalary()
{
    if (TxtNumericSalary.Value == 0)
    {
        ErrorProvider.SetError(TxtNumericSalary, "El salario debe ser mayor a 0");
        return false;
    }
    ErrorProvider.SetError(TxtNumericSalary, string.Empty);
    return true;
}
```

### Validación del GroupBox que contiene a los RadioButtons de Sex

```
private bool ValidateSex()
{
    if (!RadioBtnMale.Checked && !RadioBtnFemale.Checked)
    {
        ErrorProvider.SetError(GroupRadioBtn, "Debe seleccionar un sexo");
        return false;
    }
    ErrorProvider.SetError(GroupRadioBtn, string.Empty);
    return true;
}
```

### Validación del NumericUpDown hijos

\* Para este campo de texto no se agregó una validación, dejando de forma opcional el agregar una cantidad de hijos, hijos  $\geq 0$ .

### Método SetFormTitle()

```
private void SetFormTitle() =>
    LblFormTitle.Text = _operation == FrameOperation.Create ? "Nuevo Empleado" : "Editar Empleado";
```

Este método nos ayudará a establecer el nombre del formulario dependiendo el modo en el que se abra, ya sea para registrar un nuevo empleado o para editar uno ya existente.

The image shows two screenshots of a user interface. The top screenshot is titled "Nuevo Empleado" and shows a text field labeled "Número de cédula:" with the value "001-290702-1050T". The bottom screenshot is titled "Editar Empleado" and shows the same text field with the same value. Both screenshots show a partial view of a profile picture on the left.

### Método ValidateFields()

```
private void ValidateFields()
{
    bool[] validationResults =
    [
        ValidateNationalID(),
        ValidateNames(),
        ValidateSurnames(),
        ValidateSalary(),
        ValidateSex()
    ];

    _isValidForm = true;
    foreach (bool result in validationResults)
    {
        if (!result)
        {
            _isValidForm = false;
            break;
        }
    }

    BtnSave.Enabled = _isValidForm;
}
```

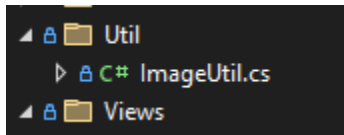
Verificamos si alguna de las validaciones es **false**, si algún componente contiene información incorrecta o está vacío, a parte del **ErrorProvider** que le hará saber al usuario el botón de **Guardar** se deshabilita y se habilitará nuevamente hasta que todos los componentes pasen las validaciones y haya en ellos la información en el formato solicitado.

### Método AttachEvents()

```
private void AttachEvents()
{
    MaskedTxtNationalID.TextChanged += (s, e) => ValidateFields();
    TxtNames.TextChanged += (s, e) => ValidateFields();
    TxtSurnames.TextChanged += (s, e) => ValidateFields();
    TxtNumericSalary.ValueChanged += (s, e) => ValidateFields();
    RadioBtnMale.CheckedChanged += (s, e) => ValidateFields();
    RadioBtnFemale.CheckedChanged += (s, e) => ValidateFields();
}
```

Desde código agregamos el evento **TextChanged** para todos los campos de texto y el evento **CheckedChanged** de los radio buttons, vamos a asignarles a todos el método **ValidateFields()** que creamos anteriormente para que se ejecute cada vez que cambie el texto de los campos de texto y al momento que se marque o se desmarque un radio button por otro.

## Crear clase ImageUtil en la carpeta Util



```
1 namespace AEDEmpleados.Util
2 {
3     public static class ImageUtil
4     {
5         public static byte[] ToByteArray(this Image image)
6         {
7             using var ms = new MemoryStream();
8             image.Save(ms, image.RawFormat);
9             return ms.ToArray();
10        }
11
12        public static Image ToImage(this byte[] array)
13        {
14            using var ms = new MemoryStream(array);
15            return Image.FromStream(ms);
16        }
17    }
18 }
```

La clase **ImageUtil** contiene dos **Extension methods** para convertir de **Image** a **byte[]** y viceversa. Los extension methods son métodos que se utilizan para agregar nuevos métodos a tipos existentes sin modificar directamente su código fuente ni crear una clase derivada. Esto es útil cuando se quiere agregar funcionalidad adicional a una clase que no se puede o no se desea modificar.

[Extension Methods - C# | Microsoft Learn](#)

## Método FillForm()

```
private void FillForm()
{
    if (_employee is null)
    {
        MessageBox.Show("No se encontró información del empleado para actualizar",
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    MaskedTextBox.NationalID.Text = _employee.NationalID;
    MaskedTextBox.NationalID.ReadOnly = true;

    TxtNames.Text = _employee.Name;
    TxtSurnames.Text = _employee.Surname;
    TxtNumericSalary.Value = _employee.Salary;
    TxtNumericChildrens.Value = _employee.Childrens;

    RadioButtonMale.Checked = _employee.Sex == Sex.Male;
    RadioButtonFemale.Checked = _employee.Sex == Sex.Female;

    PictureBox.Photo.Image = _employee.Photo?.ToImage();
}
```

Este método se encargará de rellenar el formulario apoyándose de la instancia global `_employee` que es donde se almacenará el empleado que recibiremos desde el formulario principal para editar y a su vez del método `ToImage()` de la clase `ImageUtil` para agregar la imagen al picturebox. También es importante notar que el campo **Cedula de identidad** se establece primero su valor y luego se vuelve solo de lectura, **ReadOnly = true** esto se hace para evitar que se pueda modificar el número de cédula del empleado ya que sería su clave primaria para realizar operaciones sobre el.

### Constructores del formulario FrmEmployee

```
public partial class FrmEmployee : Form
{
    private readonly FrameOperation _operation;
    private readonly Employee? _employee;
    private bool _isValidForm;

    public FrmEmployee(FrameOperation operation)
    {
        InitializeComponent();
        _operation = operation;
        InitControls();
    }

    public FrmEmployee(FrameOperation operation, Employee employee)
    {
        InitializeComponent();
        _operation = operation;
        _employee = employee;
        InitControls();
        FillForm();
    }

    private void InitControls()
    {
        AttachEvents();
        ValidateFields();
        SetFormTitle();
    }
}
```

Se crean dos constructores, el primero de ellos se utilizara al momento de crear un nuevo registro principalmente y el segundo de ellos se utilizara al momento de editar la información de un empleado existente, también creamos el método `InitControls()` para invocar varios de los métodos que creamos anteriormente una vez se crea el formulario.

## Evento click del botón Guardar

```
private void BtnSave_Click(object sender, EventArgs e)
{
    var employee = new Employee
    {
        NationalID = MaskedTextBox1.Text,
        Name = TxtNames.Text,
        Surname = TxtSurnames.Text,
        Childrens = Convert.ToInt32(TxtNumericChildrens.Value),
        Salary = TxtNumericSalary.Value,
        Sex = RadioBtnMale.Checked ? Sex.Male : Sex.Female,
        Photo = PicturePhoto.Image?.ToByteArray()
    };

    if (_operation == FrameOperation.Create)
    {
        var emp = EmployeeModel.Instance.Find(employee.NationalID);
        if (emp is not null)
        {
            MessageBox.Show("Ya existe un empleado con la misma cédula de identidad",
                "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        EmployeeModel.Instance.Add(employee);
    }
    else
        EmployeeModel.Instance.Update(employee);

    (Owner as FrmMain)?.RefreshDataGridView();

    Close();
}
```

Para guardar o editar un empleado, primero tomamos la información de cada uno de los componentes y no hace falta agregar validaciones en esta parte ya que recordemos que se podrá hacer click solo cuando el formulario contenga los datos en el formato solicitado y además el botón ya este habilitado, así que solo se toman los datos del empleado y se comprueba:

1. **Modo Create:** Se comprueba si ya existe un empleado con el mismo número de cédula, si existe se le indica al usuario, en caso contrario se procede a guardar el empleado haciendo uso de nuestro modelo de datos **EmployeeModel**
2. **Modo Update:** Se actualiza la información del empleado por medio del modelo de datos **EmployeeModel**

Finalmente se invoca el método **RefreshDataGridView()** del **FrmMain** que crearemos a continuación para que se actualice el modelo de datos de la tabla con la nueva información, posteriormente se cierra el formulario **FrmEmployee**.

## Resumen de todos los métodos del formulario FrmEmployee

```
public partial class FrmEmployee : Form
{
    private readonly FrameOperation _operation;
    private readonly Employee? _employee;
    private bool _isValidForm;

    public FrmEmployee(FrameOperation operation) ...

    public FrmEmployee(FrameOperation operation, Employee employee) ...

    private void InitControls() ...

    private void TxtNames_KeyPress(object sender, KeyPressEventArgs e) ...

    private void TxtSurnames_KeyPress(object sender, KeyPressEventArgs e) ...

    private void BtnCancel_Click(object sender, EventArgs e) => Close();

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

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

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

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

    private void AttachEvents() ...

    private void ValidateFields() ...

    private void FillForm() ...

    private void SetFormTitle() ...

    private bool ValidateNationalID() ...

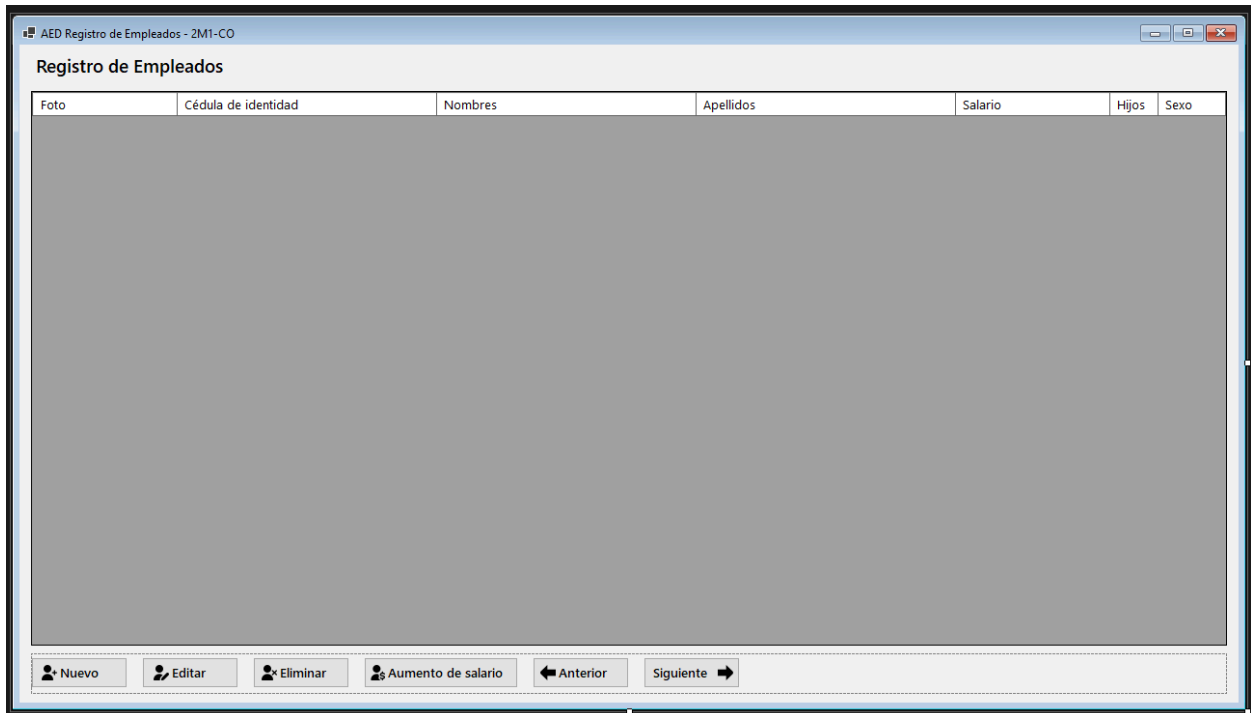
    private bool ValidateNames() ...

    private bool ValidateSurnames() ...

    private bool ValidateSalary() ...

    private bool ValidateSex() ...
}
```

## 10. Lógica del formulario FrmMain



Agregamos el evento Load() del formulario y el método UpdateControlState()

```
private void MainForm_Load(object sender, EventArgs e)
{
    UpdateControlState();
}

private void UpdateControlState() =>
    BtnEdit.Enabled = BtnDelete.Enabled = BtnPrevious.Enabled =
    BtnNext.Enabled = BtnSalaryIncrease.Enabled = DgView.Rows.Count > 0;
```

El método **UpdateControlState()** se encarga de habilitar o deshabilitar los botones de **Editar**, **Eliminar**, **Aumento de salario**, **Anterior** y **Siguiente** si no hay filas en la tabla, ya que no tendrían uso alguno mientras no haya registros, una vez se agrega un registro los botones mencionados anteriormente se habilitan.

Se invoca por primera vez al cargar el formulario para deshabilitar los botones y que se habiliten una vez se agregue un nuevo registro.

**Método RefreshDataGridView()**

```
public void RefreshDataGridView()
{
    DgView.DataSource = EmployeeModel.Instance.GetAll();
    UpdateControlState();
}
```

Actualiza el contenido de la tabla trayendo los últimos cambios apoyándose de **EmployeeModel** y finalmente invocando **UpdateControlState()** para ver si tiene que habilitar o deshabilitar los componentes.

#### Evento click del botón Nuevo

```
private void BtnNew_Click(object sender, EventArgs e)
{
    var frmEmployee = new FrmEmployee(FrameOperation.Create)
    {
        Owner = this
    };
    frmEmployee.ShowDialog();
}
```

Al pulsar el botón **Nuevo** abrimos el formulario **FrmEmployee** en modo **Create** para agregar un nuevo empleado, también usamos **ShowDialog()** para que la ventana sea modal, es decir que no se pueda hacer click fuera de ella sin antes cerrarla o registrar un nuevo empleado, de esa forma nos aseguramos que no hayan más de un formulario de registro a la vez. De igual forma le indicamos que el **FrmMain** es el **Owner** del formulario **FrmEmployee**, así podremos invocar métodos del formulario **FrmMain** desde **FrmEmployee**.

#### Evento click del botón Editar

```
private void BtnEdit_Click(object sender, EventArgs e)
{
    var selectedIndex = DgView.SelectedRows[0].Index;
    if (DgView.Rows[selectedIndex].DataBoundItem is not Employee selectedEmployee)
    {
        MessageBox.Show("No se ha seleccionado ningún empleado para editar", "Atención",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return;
    }

    var frmEmployee = new FrmEmployee(FrameOperation.Update, selectedEmployee)
    {
        Owner = this
    };
    frmEmployee.ShowDialog();
}
```

Al pulsar sobre el botón **Editar** se obtiene el empleado de la fila seleccionada y se pasa al formulario **FrmEmployee** en modo **Update**, de igual forma se abre la ventana con **ShowDialog()** para que sea modal.



### Evento click del botón Eliminar

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

    if (string.IsNullOrEmpty(selectedEmployee.NationalID))
    {
        MessageBox.Show("No se ha encontrado cédula del empleado a eliminar", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    var dlgResult = MessageBox.Show($"Seguro que desea eliminar a " +
        $"{selectedEmployee.Name} {selectedEmployee.Surname}?",
        "Atención", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (dlgResult == DialogResult.No)
        return;

    EmployeeModel.Instance.Delete(selectedEmployee.NationalID);
    RefreshDataGridView();
}
```

Al pulsar sobre el botón **Eliminar** se obtiene el empleado de la fila seleccionada, preguntamos antes al usuario si está seguro que desea eliminarlo y si la respuesta es distinta de NO procedemos a eliminar el Empleado apoyándonos en **EmployeeModel** y finalmente actualizamos el contenido de la tabla.

### Evento click del botón Anterior y Siguiente

```
private void BtnPrevious_Click(object sender, EventArgs e)
{
    var currentRowIndex = DgView.SelectedRows[0].Index;
    if (currentRowIndex > 0)
        DgView.Rows[currentRowIndex - 1].Selected = true;
}

private void BtnNext_Click(object sender, EventArgs e)
{
    var currentRowIndex = DgView.SelectedRows[0].Index;
    if (currentRowIndex < DgView.Rows.Count - 1)
        DgView.Rows[currentRowIndex + 1].Selected = true;
}
```

Estos botones se utilizan para navegar por cada fila de la tabla, se toma la fila actual y dependiendo cual se pulsa se aumenta en una posición o se disminuye una posición y luego esa fila se marca como seleccionada.

#### Evento click del botón Aumento de salario

```
private void BtnSalaryIncrease_Click(object sender, EventArgs e)
{
    var frmEmployeeSalary = new FrmEmployeeSalary()
    {
        Owner = this
    };
    frmEmployeeSalary.ShowDialog();
}
```

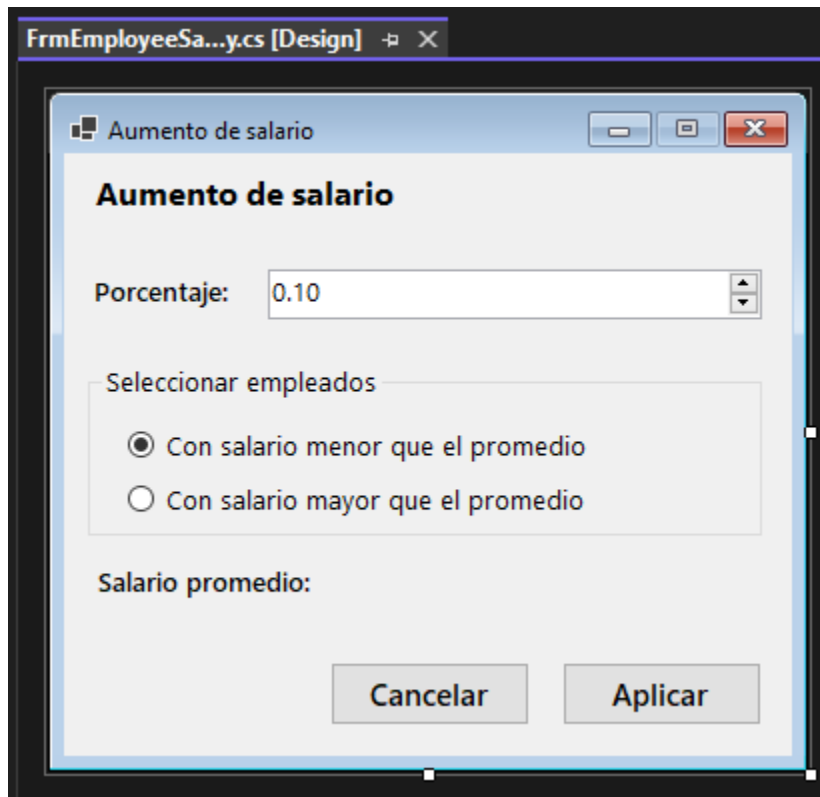
Se abre el formulario **FrmEmployeeSalary()** de forma modal con **ShowDialog()**.

#### Resumen de todos los métodos del formulario FrmMain

```
public partial class FrmMain : Form
{
    public FrmMain()
    {
        InitializeComponent();
    }

    private void MainForm_Load(object sender, EventArgs e)...
    private void UpdateControlState()...
    public void RefreshDataGridView()...
    private void BtnNew_Click(object sender, EventArgs e)...)
    private void BtnEdit_Click(object sender, EventArgs e)...)
    private void BtnDelete_Click(object sender, EventArgs e)...)
    private void BtnSalaryIncrease_Click(object sender, EventArgs e)...)
    private void BtnPrevious_Click(object sender, EventArgs e)...)
    private void BtnNext_Click(object sender, EventArgs e)...)
}
```

## 11. Lógica del formulario FrmEmployeeSalary



### Evento Load() del formulario

```
private void FrmEmployeeSalary_Load(object sender, EventArgs e) =>
    LblAvgSalary.Text = $"Salario promedio: {EmployeeModel.Instance.AverageSalary()}";
```

Se obtiene el salario promedio de los empleados apoyándonos en el modelo de datos **EmployeeModel** y el método **AverageSalary()**, asignando el valor al Label encargado de mostrar el salario promedio.

### Evento click del botón Cancelar

```
private void BtnCancel_Click(object sender, EventArgs e) => Close();
```

Simplemente cerramos el formulario **FrmEmployeeSalary**

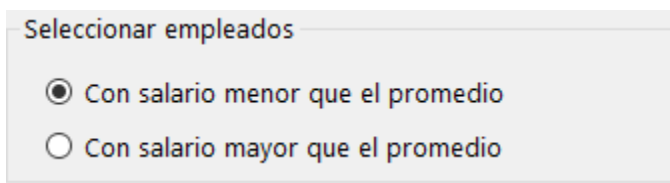
## Evento click del botón Apply

```
private void BtnApply_Click(object sender, EventArgs e)
{
    var averageSalary = EmployeeModel.Instance.AverageSalary();
    Predicate<Employee> condition;
    if (RadioBtnGreaterThanAvg.Checked)
        condition = emp => emp.Salary > averageSalary;
    else
        condition = emp => emp.Salary < averageSalary;

    var empCounter = EmployeeModel.Instance.SalaryIncrease(TxtPercentage.Value, condition);
    MessageBox.Show($"Se aumento el salario a {empCounter} empleados", "Información",
        MessageBoxButtons.OK, MessageBoxIcon.Information);

    (Owner as FrmMain)?.RefreshDataGridView();
    Close();
}
```

Primeramente se obtiene el salario promedio de todos los empleados en **averageSalary** luego en base al RadioButton que está seleccionado se crea el predicado o condición para la función **SalaryIncrease** de **EmployeeModel**



Seleccionar empleados

☒ Con salario menor que el promedio

☐ Con salario mayor que el promedio

1. El predicado sería Salario del empleado < Salario promedio
2. El predicado sería Salario del empleado > Salario promedio

Finalmente se pasa el salario promedio y el predicado a la función **SalaryIncrease** y esta devuelve el número de empleados beneficiado el cual se muestra en un MessageBox, se invoca el método **RefreshDataGridView()** para actualizar el modelo de datos de la tabla y se cierra el formulario de aumento de salario.

## Resumen de todos los métodos del formulario FrmEmployeeSalary

```
public partial class FrmEmployeeSalary : Form
{
    public FrmEmployeeSalary()
    {
        InitializeComponent();
    }

    private void FrmEmployeeSalary_Load(object sender, EventArgs e) =>
        LblAvgSalary.Text = $"Salario promedio: {EmployeeModel.Instance.AverageSalary()}";

    private void BtnCancel_Click(object sender, EventArgs e) => Close();

    private void BtnApply_Click(object sender, EventArgs e)
    {
        var averageSalary = EmployeeModel.Instance.AverageSalary();
        Predicate<Employee> condition;
        if (RadioBtnGreaterThanAvg.Checked)
            condition = emp => emp.Salary > averageSalary;
        else
            condition = emp => emp.Salary < averageSalary;

        var empCounter = EmployeeModel.Instance.SalaryIncrease(TxtPercentage.Value, condition);
        MessageBox.Show($"Se aumento el salario a {empCounter} empleados", "Información",
            MessageBoxButtons.OK, MessageBoxIcon.Information);

        (Owner as FrmMain)?.RefreshDataGridView();
        Close();
    }
}
```

## Ejecución del programa

AED Registro de Empleados - ZM1-CO

Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
------	---------------------	---------	-----------	---------	-------	------

Nuevo Editar Eliminar Aumento de salario Anterior Siguiete

AED Registro de Empleados - ZM1-CO

Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
<div><div>Registro de empleado</div><div><div>Nuevo Empleado</div><div><div><div>Foto</div><div>Foto</div></div><div><div>Número de cédula:</div><div>001-100203-1040T</div></div><div><div>Nombres:</div><div>Juan</div></div><div><div>Apellidos:</div><div>Lopez</div></div><div><div>Salario:</div><div>6000.00</div></div><div><div>Cantidad de hijos:</div><div>1</div></div><div><div>Sexo:</div><div><div><input checked="" type="radio"/> Masculino</div><div><input type="radio"/> Femenino</div></div></div><div><div>Cancelar</div><div>Guardar</div></div></div></div></div>						

Nuevo

Editar

Eliminar


Aumento de salario

Anterior

Siguiente

AED Registro de Empleados - ZM1-CO


Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
	001-100203-1040T	Juan	Lopez	6000.00	1	Male

Registro de empleado

Nuevo Empleado

Foto



Número de cédula:

020-100404-1040G

Nombres:

Maria

Apellidos:

Gutierrez

Salario:

7000.00

Cantidad de hijos:

20

Sexo:

☐ Masculino

☒ Femenino

Cancelar

Guardar

Nuevo

Editar

Eliminar



Aumento de salario

Anterior

Siguiente

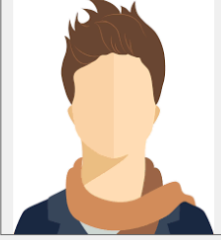
AED Registro de Empleados - ZM1-CO

### Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
	001-100203-1040T	Juan	Lopez	6000.00	1	Male
	020-100404-10				20	Female

Registro de empleado

#### Editar Empleado



Foto

Número de cédula: 001-100203-1040T

Nombres: Juan


Apellidos: Lopez


Salario: 6000.00


Cantidad de hijos: 1


Sexo: ☒ Masculino ☐ Femenino


Cancelar Guardar


 Nuevo

 Editar

 Eliminar



 Aumento de salario

 Anterior

 Siguiente

AED Registro de Empleados - ZM1-CO

### Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
	001-100203-1040T	Juan	Lopez	6000.00	1	Male
	020-100404-1040G	Maria	Gutierrez	7000.00	20	Female

Aumento de salario

#### Aumento de salario

Porcentaje: 0.10


Seleccionar empleados


☒ Con salario menor que el promedio


☐ Con salario mayor que el promedio


Salario promedio: 6500.00


Cancelar Aplicar


 Nuevo

 Editar

 Eliminar



 Aumento de salario

 Anterior

 Siguiente

AED Registro de Empleados - ZM1-CO

Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
	001-100203-1040T	Juan	Lopez	6000.00	1	Male
	020-100404-1040G	Maria	Gutierrez	7000.00	20	Female

Aumento de salario

Aumento de salario

Porcentaje: 0.10

Selección

Información

Se aumento el salario a 1 empleados

OK

Cancelar

Aplicar

Nuevo

Editar

Eliminar



Aumento de salario

Anterior

Siguiente

AED Registro de Empleados - ZM1-CO

Registro de Empleados

Foto	Cédula de identidad	Nombres	Apellidos	Salario	Hijos	Sexo
	001-100203-1040T	Juan	Lopez	6600.000	1	Male
	020-100404-1040G	Maria	Gutierrez	7000.00	20	Female

Nuevo

Editar

Eliminar

Aumento de salario

Anterior

Siguiente



## Resumen de todos los métodos de EmployeeModel

```
1  using AEDEmpleados.Entities;
2
3  namespace AEDEmpleados.Models
4  {
5      public class EmployeeModel
6      {
7          private static readonly EmployeeModel _instance = new();
8          public static EmployeeModel Instance { get => _instance; }
9
10         private EmployeeModel()
11         { }
12
13         private Employee[] _employees = [];
14         private int _size = 0, _quantity = 0;
15
16         public void Add(Employee employee) ...
17
18
19
20
21
22
23
24
25
26
27
28         public bool Update(Employee employee) ...
29
30
31
32
33
34
35
36
37
38
39
40         public bool Delete(string nationalID) ...
41
42
43
44
45
46
47
48
49
50
51
52
53
54         public Employee[] GetAll() => _employees.Take(_quantity).ToArray();
55
56         public Employee? Find(string nationalID) ...
57
58
59         public decimal AverageSalary() => _employees.Average(e => e.Salary);
60
61         public int SalaryIncrease(decimal percentage, Predicate<Employee> condition) ...
62
63
64
65
66
67
68
69
70
71
72
73
74
75     }
76 }
```