INSTITUTO TECNOLOGICO DE MEXICO

INSTITUTO TECNOLOGICO DE TLAXIACO

CARRERA:

ING. EN SISTEMAS COMPUTACIONALES

ASIGNATURA:

TOPICOS AVANZADOS DE PROGRAMACIÓN

DOCENTE:

ING. JOSE ALFREDO ROMAN CRUZ

NOMBRE DE LAS ALUMNAS:

ABIGAIL CORONEL SANTIAGO

PRACTICA 5:

DISEÑO DE CONTROL PERSONALIZADO

SEMESTRE:

4US

CICLO ESCOLAR:

FEBRERO 2022 – JUNIO 2022

TLAXIACO, OAXACA A 23 DE MARZO DEL 2022

Índice:

Tabla de ilustraciones:

Objetivo:

El presente trabajo tiene como objetivo que el estudiante aprenda acerca de la personalización de botones en visual studio, debe realizar una practica 5 para realizar la implementación de un componente personalizado para su implementación como control en Visual Studio.

Materiales:

- laptop
- internet
- Visual Studio

Procedimiento:

1. Diseño de botón personalizado

1. Primero crearemos un nuevo **proyecto de Windows Form** con Visual C#, lo llamaré **CustomControls**

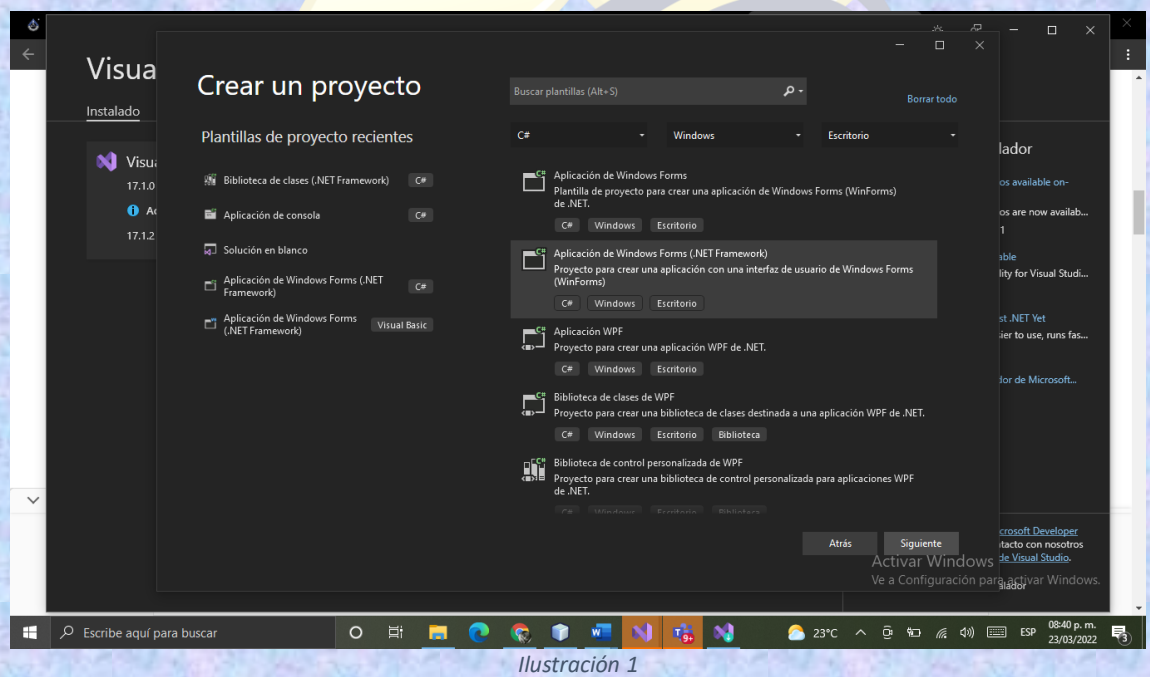

Ilustración 1

2. Una vez creados el proyecto dentro de este mismo agregaremos una carpeta de nombre **RJControls** para agrupar todos los controles personalizados en un solo espacio.

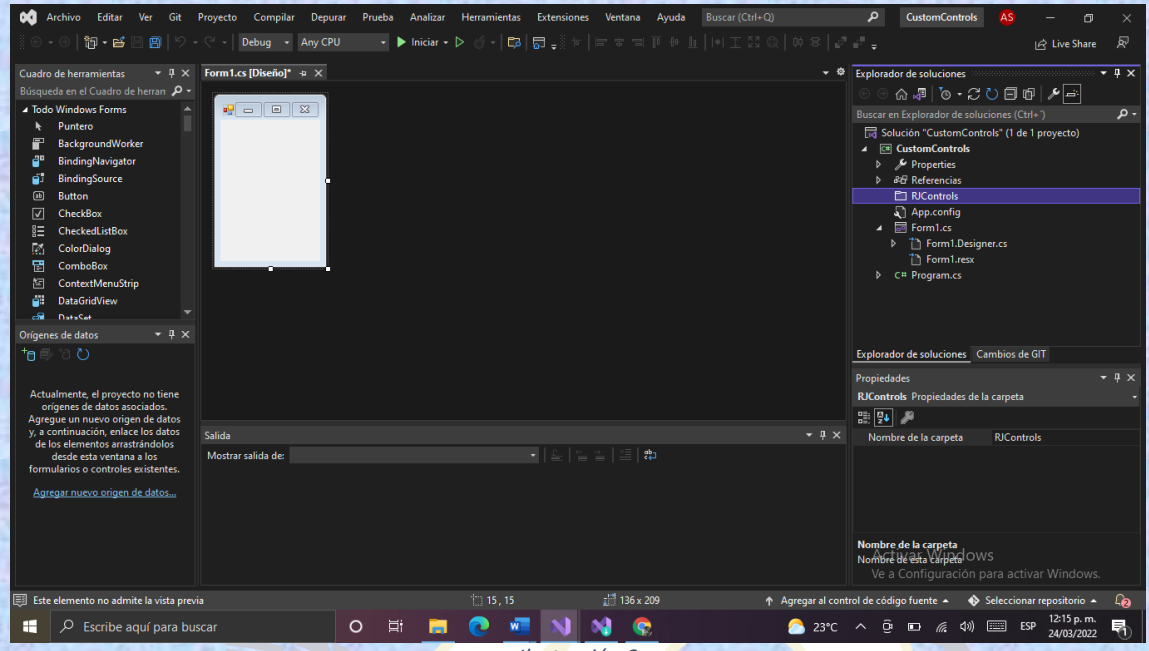

Ilustración 2

3. Seguidamente **agregamos una clase** de nombre **RJToggleButton**, o el nombre que desean.

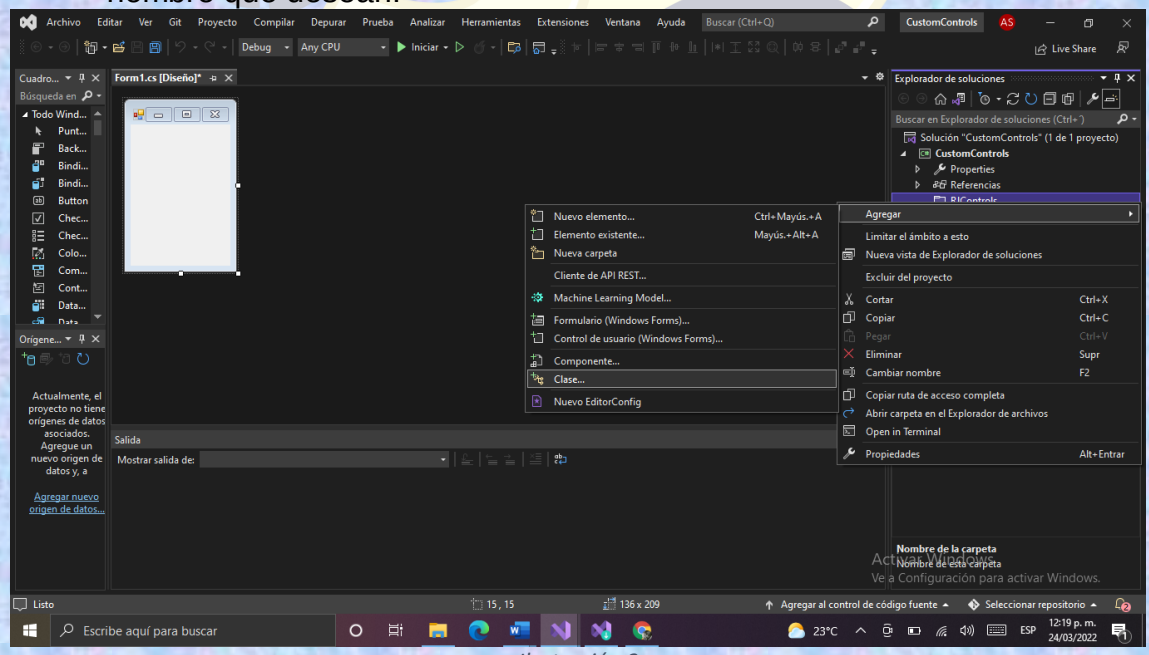

Ilustración 3

4. Seleccionamos la clase le damos en **Agregar**.

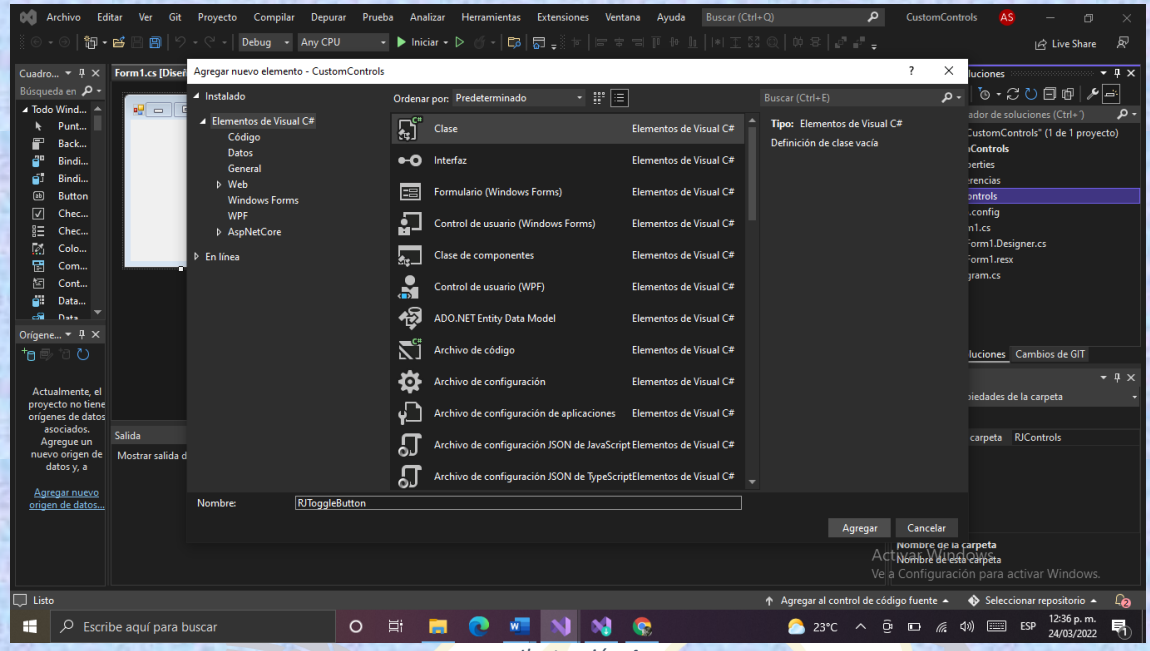

Ilustración 4

5. Importamos **la librería Windows Forms** para usar los controles tradicionales de Windows, e importamos la librería de dibujos (**Drawing**) y la libreria **ComponentModel** para implementar atributos.

```
6
7   using System.Windows.Forms;
8   using System.Drawing;
9   using System.Drawing.Drawing2D;
10  using System.ComponentModel;
11
```

Ilustración 5

6. En la clase RJToggleButton, **heredamos la clase (Control) CheckBox** de la librería de Windows Form, ya que es el más parecido y conveniente para este caso. Estos tipos de controles se conocen como **controles extendidos**.

```
13  {
        3 referencias
14      public class RJToggleButton : CheckBox
15      {
```

Ilustración 6

7. Declaramos campos para la apariencia del control ToggleButton y asignar sus valores predeterminados, por ejemplo, color de fondo y palanca en estado encendido y apagado.

```
17      private Color onBackColor = Color.MediumSlateBlue;
18      private Color onToggleColor = Color.WhiteSmoke;
19      private Color offBackColor = Color.Gray;
20      private Color offToggleColor = Color.Gainsboro;
21      private bool solidStyle = true;
```

Ilustración 7

8. Generamos propiedades para exponer los campos anteriores declarados.

```
24      [Category("RJ Code Advance")]
        1 referencia
25      public Color OnBackColor
26      {
27          get
28          {
29              return onBackColor;
30          }
31
32          set
33          {
34              onBackColor = value;
35              this.Invalidate();
36          }
37      }
38
39      [Category("RJ Code Advance")]
        1 referencia
40      public Color OnToggleColor
41      {
42          get
43          {
44              return onToggleColor;
45          }
46
47          set
48          {
49              onToggleColor = value;
50              this.Invalidate();
51          }
52      }
53
```

Ilustración 8

```
54          [Category("RJ Code Advance")]
            1 referencia
55          public Color OffBackColor
56          {
57              get
58              {
59                  return offBackColor;
60              }
61
62              set
63              {
64                  offBackColor = value;
65                  this.Invalidate();
66              }
67          }
68
69          [Category("RJ Code Advance")]
            1 referencia
70          public Color OffToggleColor
71          {
72              get
73              {
74                  return offToggleColor;
75              }
76
77              set
78              {
79                  offToggleColor = value;
80                  this.Invalidate();
81              }
82          }
83
```

Ilustración 9

```
84          [Browsable(false)]
            0 referencias
85          public override string Text
86          {
87              get
88              {
89                  return base.Text;
90              }
91
92              set
93              {
94
95              }
96          }
97
98          [Category("RJ Code Advance")]
99          [DefaultValue(true)]
            0 referencias
100         public bool SolidStyle
101         {
102             get
103             {
104                 return solidStyle;
105             }
106
107             set
108             {
109                 solidStyle = value;
110                 this.Invalidate();
111             }
112         }
113
```

Ilustración 10

9. En el constructor puedes inicializar las propiedades que deseas, por ejemplo, la propiedad MinimunSize, propio del control CheckBox.

```
113
114         //Constructor
            1 referencia
115         public RJToggleButton()
116         {
117             this.MinimumSize = new Size(45, 22);
118         }
119
```

Ilustración 11

10. El método **GetFigurePath** se encarga de **crear la forma del control**, en este caso una figura con **lados redondeados**, para ello **se agregan 2 arcos** para el **lado izquierdo** (En el eje X=0 y eje Y=0 con tamaño arcSize, comenzando en un ángulo de 90 grados con un rango de 180 grados) y **lado derecho** (En el eje X=[Ancho de control – arcSize -2] y eje Y=0 con tamaño arcSize, comenzando en un ángulo de 270 grados con un rango de 180 grados). Debemos tener en cuenta que la dirección de los ángulos del círculo unitario en C# o Visual Basic son en sentido de las agujas del reloj (Hacia la derecha).

```
121     private GraphicsPath GetFigurePath()
122     {
123         int arcSize = this.Height - 1;
124         Rectangle leftArc = new Rectangle(0, 0, arcSize, arcSize);
125         Rectangle rightArc = new Rectangle(this.Width - arcSize - 2, 0, arcSize, arcSize);
126
127         GraphicsPath path = new GraphicsPath();
128         path.StartFigure();
129         path.AddArc(leftArc, 90, 180);
130         path.AddArc(rightArc, 270, 180);
131         path.CloseFigure();
132
133         return path;
134     }
```

Ilustración 12

11. Anular el evento Paint del control para volver a dibujar la apariencia del control a nuestra manera. En este caso, básicamente se dibuja el control en estado encendido o apagado.

```
136         protected override void OnPaint(PaintEventArgs pevent)
137         {
138             int toggleSize = this.Height - 5;
139             pevent.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
140             pevent.Graphics.Clear(this.Parent.BackColor);
141
142             if (this.Checked) //ON
143             {
144                 //Draw the control surface
145                 if (solidStyle)
146                     pevent.Graphics.FillPath(new SolidBrush(onBackColor), GetFigurePath());
147                 else pevent.Graphics.DrawPath(new Pen(onBackColor, 2), GetFigurePath());
148                 //Draw the toggle
149                 pevent.Graphics.FillEllipse(new SolidBrush(onToggleColor),
150                     new Rectangle(this.Width - this.Height + 1, 2, toggleSize, toggleSize));
151             }
152             else //OFF
153             {
154                 //Draw the control surface
155                 if (solidStyle)
156                     pevent.Graphics.FillPath(new SolidBrush(offBackColor), GetFigurePath());
157                 else pevent.Graphics.DrawPath(new Pen(offBackColor, 2), GetFigurePath());
158                 //Draw the toggle
159                 pevent.Graphics.FillEllipse(new SolidBrush(offToggleColor),
160                     new Rectangle(2, 2, toggleSize, toggleSize));
161             }
162         }
163     }
```

Ilustración 13

12. Ahora nos dirigimos a nuestra formulario en el cuadro de herramientas nos deberá aparecer el nombre de nuestro diseño personalizado en este caso tiene el nombre de **RJToggleButton.**

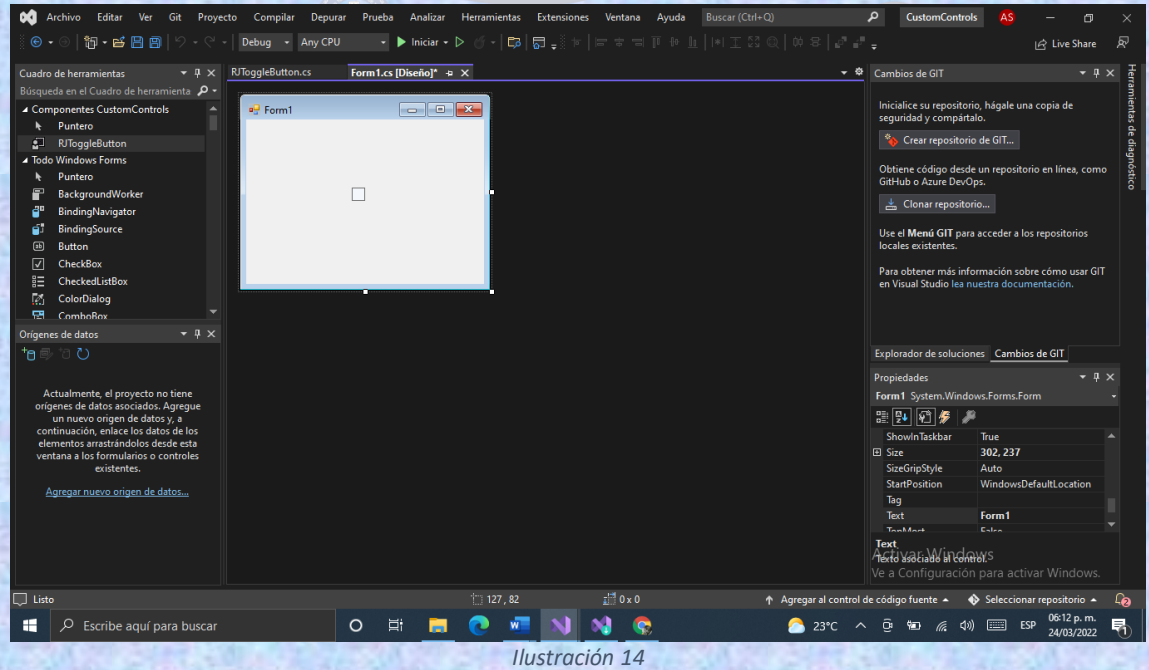

Ilustración 14

13. Seleccionamos y arrastramos a nuestro formulario, aquí es donde podemos cambiar las propiedades de nuestro Boton personalizado

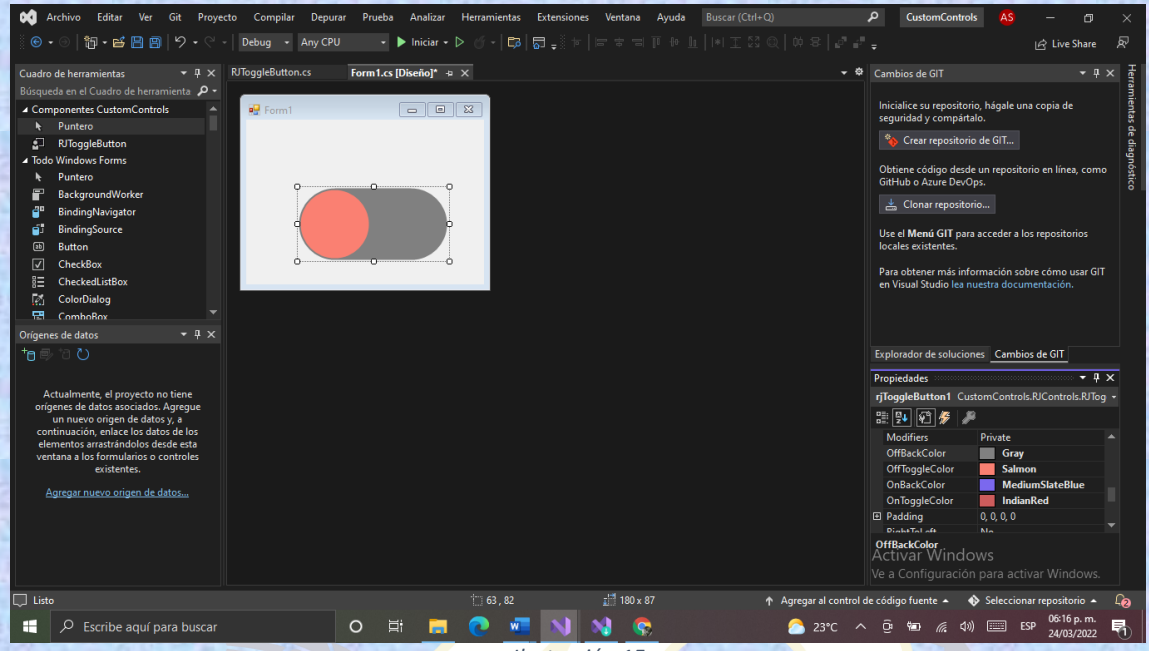

Ilustración 15

14. Compilamos nuestro formulario y nos saldrá el siguiente recuadro donde podemos encender y apagar nuestro botón.

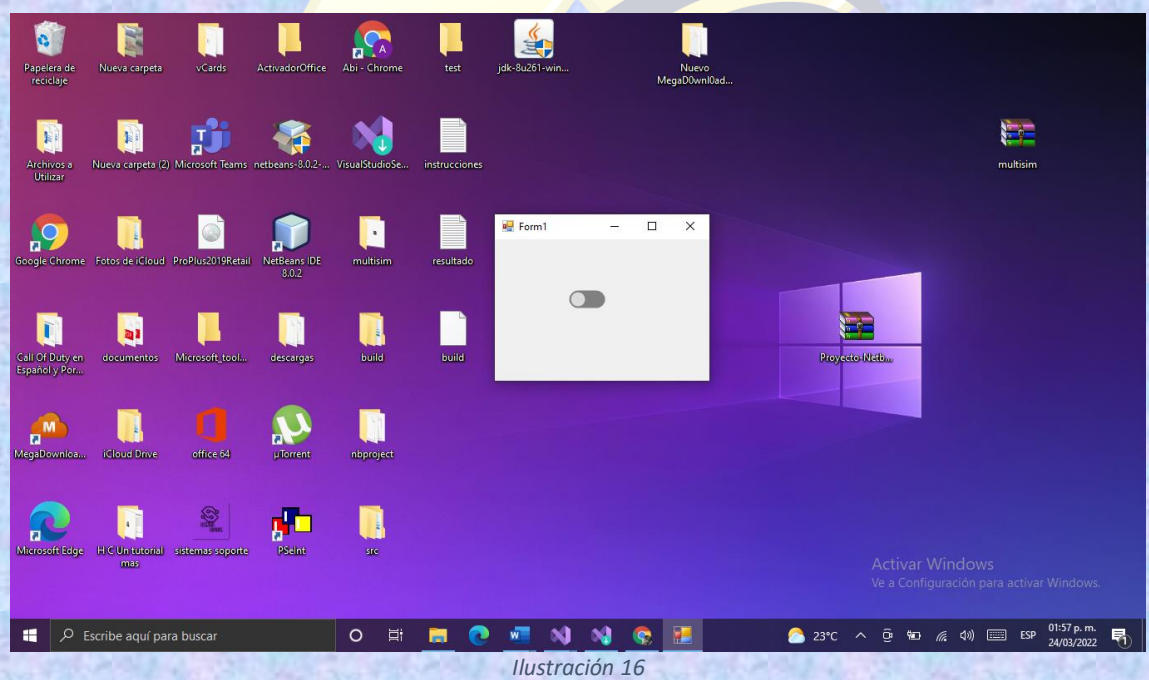

Ilustración 16

2. Diseñar una aplicación de escritorio implementando en botón personalizado.

1. Para crear una aplicación donde podamos implementar el botón personalizado nos dirigimos a nuestro formulario que anteriormente teníamos abierto

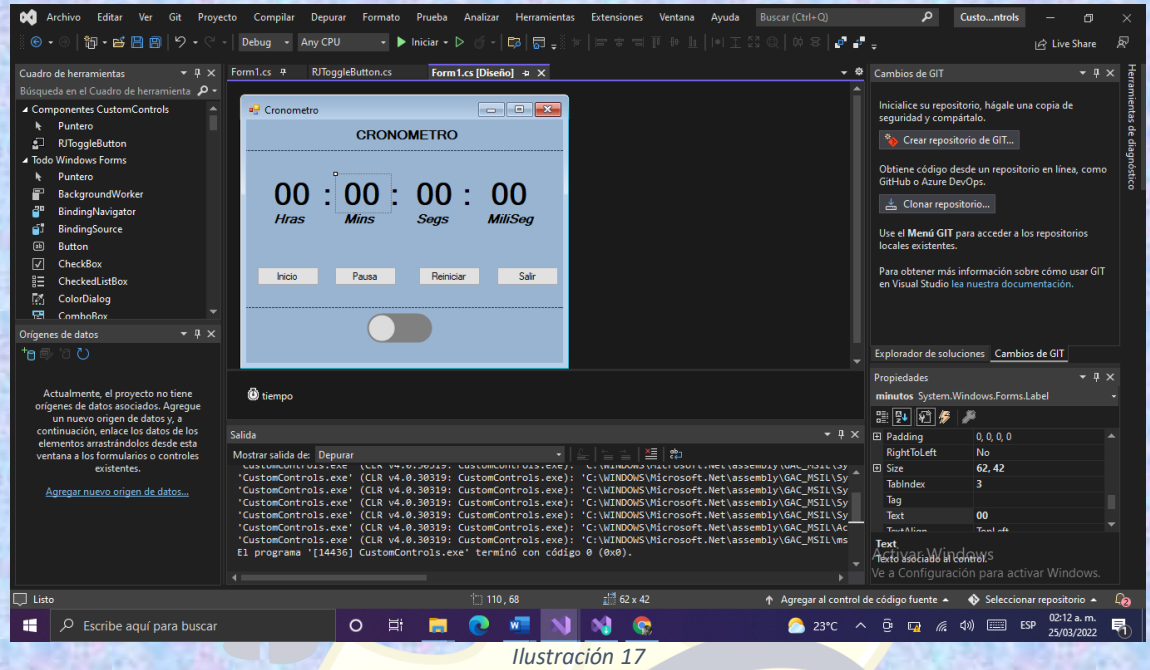

Ilustración 17

2. Ahora buscamos en el cuadro de herramientas un Label seleccionamos y arrastramos hacia nuestro formulario ahora solo cambiaremos sus propiedades su texto en este caso tengo ocho Label de los cuales tiene por texto 00, Hras, Mins, Segs, MiliSeg.

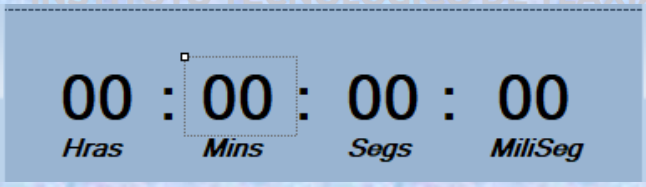

Ilustración 18

3. Buscamos en el cuadro de herramientas un Botton seleccionamos y arrastramos y tendrán por nombre Iniciar, Pausar, Reiniciar y Salir.

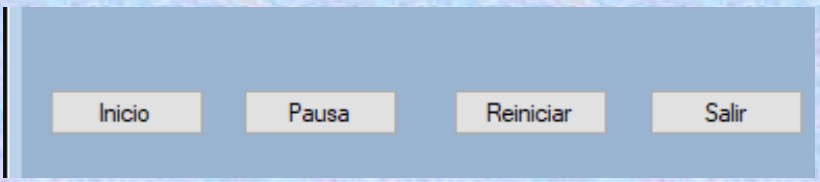

Ilustración 19

4. Ahora buscamos el botón que personalizamos y que tiene por nombre RJToggleButton seleccionamos y arrastramos, también buscaremos en el cuadro de herramientas un Timer.

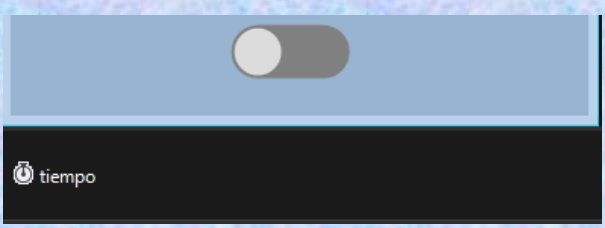

Ilustración 20

5. Ahora solo para que nuestro cronómetro funcione de manera correcta tendremos que programarlo para programar los Milisegundos implementaremos el siguiente código:

```
1 referencia
private void timer1_Tick(object sender, EventArgs e)
{
    int aux = Convert.ToInt32(milisegundos.Text);
    int aux1 = Convert.ToInt32(segundos.Text);
    int aux2 = Convert.ToInt32(minutos.Text);
    int aux3 = Convert.ToInt32(horas.Text);
    aux = aux + 1;
    milisegundos.Text = aux.ToString();
    if (aux > 59) //para los milisegundos
    {
        aux1 = aux1 + 1;
        segundos.Text = aux1.ToString();
        aux = 0;
        milisegundos.Text = aux.ToString();

    }
```

Ilustración 21

6. Ahora para que los segundas corran utilizaremos el siguiente código:

```
if (aux > 59) //para los segundos
{
    aux2 = aux2 + 1;
    minutos.Text = aux2.ToString();
    aux1 = 0;
    segundos.Text = aux1.ToString();

}
```

Ilustración 22

7. El código para los minutos sería el siguiente

```
    if (aux2 > 59) //para los minutos
    {
        aux3 = aux3 + 1;
        horas.Text = aux3.ToString();
        aux2 = 0;
        minutos.Text = aux3.ToString();

    }
}
```

Ilustración 23

8. Ahora para que nuestros botones de Iniciar, Pausar, Reanudar y Salir funcionen implementaremos el siguiente código

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    tiempo.Start(); //iniciando cronometro

}
```

Ilustración 24

```
0 referencias
private void button2_Click(object sender, EventArgs e)
{
    tiempo.Stop();//detener objeto "timer"
}

0 referencias
private void btnReiniciar_Click(object sender, EventArgs e)
{
    horas.Text = "00"; //reiniciando horas
    minutos.Text = "00";
    segundos.Text = "00";
    milisegundos.Text = "00";
}

0 referencias
private void btnSalir_Click(object sender, EventArgs e)
{
    Close(); //boton salir
}
```

Ilustración 25

9. Ahora para que nuestro botón personalizado funcione implementaremos el siguiente código

```
    0 referencias
    private void rjToggleButton1_CheckedChanged(object sender, EventArgs e)
    {
        panel1.BackColor = Color.BlueViolet;

    }
}
```

Ilustración 26

10. Y por último solo compilamos nuestro proyecto y verificamos que esté funcionando correctamente en este caso si quedo como yo quería

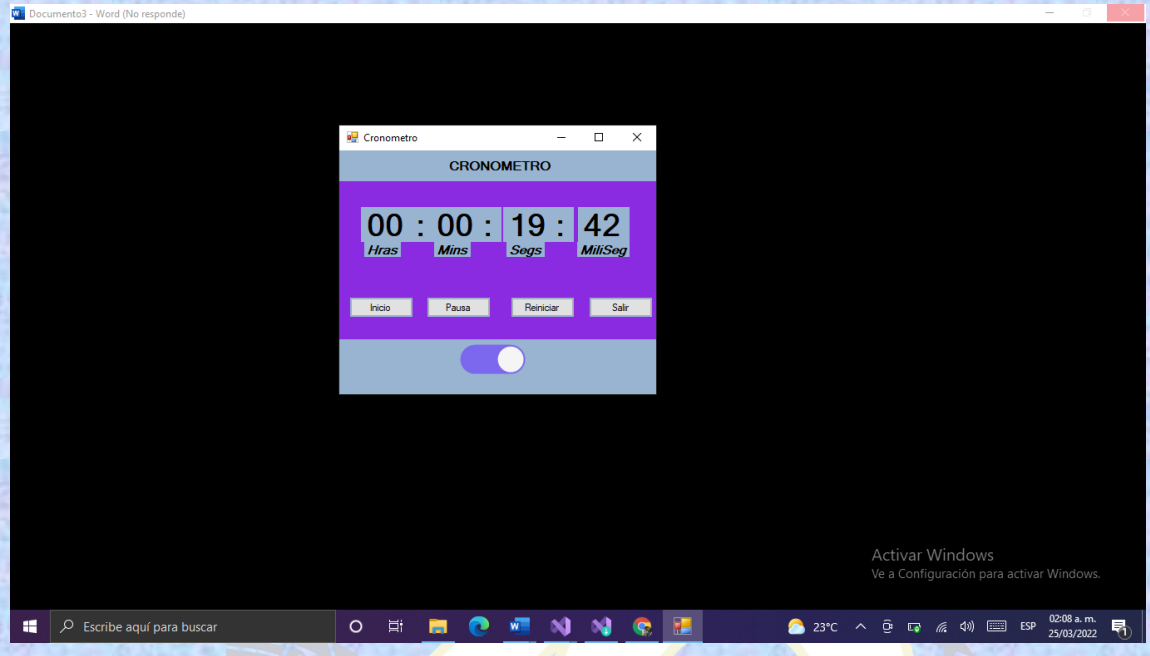

Ilustración 27

Conclusión:

Seguramente alguna vez has intentado darles un toque personal a tus aplicaciones con botones personalizados ya sea con imágenes o solo dándoles una forma no convencional, pero claro, te has topado con la restricciones de Windows para estos casos, y al final has escogió alguna de las opciones más comunes, a un botón le agregas una imagen o creas tu propio botón en base a imágenes superpuestas, pero ambas aunque no quieras aceptarlo tienen el mismo problema, el área activa siempre es **RECTANGULAR**.

Pero el reinado de las formas rectangulares ha llegado a su fin, y ahora podrás agregar botones de cualquier forma a tus formularios. Pero no solo es apariencia, la respuesta a los eventos del mouse solamente ocurren dentro la forma del botón, no en la clásica área rectangular.

Bibliografía:

https://rjcodeadvance.com/toggle-button-custom-controls-winform-c/

https://www.youtube.com/watch?v=a5SM5mrqb5s