CALCULADORA

ANALISTA PROGRAMADOR

PROGRAMACIÓN II

Profesor Gonzalo Duarte

Octubre 2023

Karen Fernández
Valentina Benítez

Resumen

En este documento se detalla el proceso de desarrollo de una aplicación de tipo “Calculadora”, creada con el objetivo de funcionar como aplicación de escritorio para *Windows*. Se realizó a través del *IDE Visual Studio*, utilizando la interfaz o *API Windows Presentation Foundation (WPF)*, que forma parte de *.NET Framework*, esta permite diseñar y programar a la vez aplicaciones funcionales con diseño amigable para el usuario.

WPF aporta herramientas que simplifican el diseño, mediante lenguaje *XAML* (de etiquetas) y trabaja en conjunto con métodos definidos en lenguaje *C#* que determinan las funciones particulares de cada componente y del sistema en general.

Contenido

1. Introducción

1.1 .NET

Es una colección de diferentes plataformas de software de Microsoft que permite la creación y ejecución de todo tipo de software, desde aplicaciones de escritorio hasta aplicaciones web, rutinas de bases de datos, etc. En la plataforma de desarrollo se pueden utilizar una serie de lenguajes, implementaciones, herramientas y bibliotecas para el desarrollo de las aplicaciones. Los dos principales lenguajes de programación son *C#* y *Visual Basic .NET*.

Los componentes de la arquitectura *.NET* juegan un papel importante en el desarrollo de aplicaciones. Los podemos clasificar en:

- ☐ *.NET Framework*
- ☐ *.NET Core framework*
- ☐ *Xamarin*
- ☐ *Windows UWP.*

.NET Framework está dividido en diferentes subcategorías y categorías de programas y, por lo tanto, contiene diferentes modelos de ejecución entre los que el usuario debe elegir al desarrollar el software. La base del desarrollo es la biblioteca de clases, que ha estado disponible en general como fuente compartida desde 2014.
La llamada biblioteca de clases base permite el desarrollo de aplicaciones no sólo para entornos *Windows*, sino también para plataformas como *Android* o *MacOS.*

.NET Core generalmente se utiliza para desarrollar aplicaciones para la plataforma universal de *Windows UWP* y *Asp.NetCore,* aplicaciones en la nube. La biblioteca de clases *Core Ex* es compatible con *Windows*, *MacOs* y *Linux*.

Xamarin en la que se pueden desarrollar aplicaciones para *Android, iOS, tvOS, watchOS, macOS* y *Windows*. Ésta, dispone de herramientas y bibliotecas específicas.

Por último, ***UWP***, la plataforma universal de *Windows*. Aunque algunos desarrolladores la sitúan dentro de la plataforma *.NET Core* al compartir algunas bibliotecas de éste, Microsoft la considera una implementación independiente [1].

1.2 WPF - Windows Presentation Foundation

Es una serie de ensamblados y herramientas que forma parte de *.NET Framework*, destinado a proporcionar una *API* (*Application Programming Interface* - Interfaz de programación de aplicaciones) para crear interfaces de usuario enriquecidas y sofisticadas para *Windows*. *Windows Forms* es su predecesor, pero *WPF* incorpora nuevas funcionalidades como tipografía avanzada, Facetas como soporte para 3D, documentos similares al *PDF*, etc.

Sus principales características son:
- Interfaz gráfica declarativa que se realiza mediante el lenguaje de etiquetas *XAML* (*eXtensible Application Markup Language*). *Visual Studio*, así como los miembros de la familia de entornos de desarrollo de Microsoft como *Blend*, están preparados para generar código *XAML* de forma nativa. *XAML* proporciona un medio para que los diseñadores puedan colaborar estrechamente en la creación de aplicaciones de este tipo.
- Permite el diseño dinámico, facilitando el ajuste de los componentes.
- Sus gráficos se basan en vectores, lo cual permite que sean escalados sin deformaciones, además de ocupar menos espacio de almacenamiento. Aunque también admite imágenes rasterizadas.

Es una herramienta muy potente que puede ser usada eficazmente para múltiples fines, pero se debe destacar que es una excelente opción para la creación de aplicaciones de escritorio para *Windows* [2].

2. Desarrollo

Se realiza la descripción paso a paso de cómo se creó la interfaz y el funcionamiento de una aplicación de escritorio de tipo “Calculadora” con *WPF* y código *C#* a través del *IDE Visual Studio* (Figura 2.1).

El código se encuentra en: https://github.com/ValentiinaBenitez/calculadora/tree/main

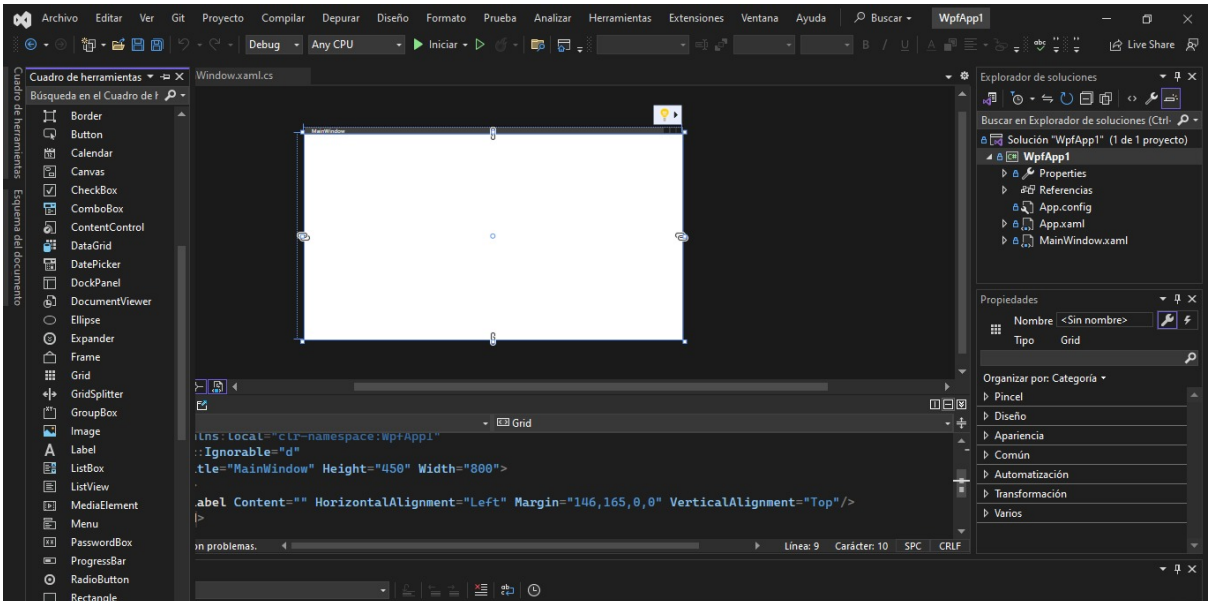

Figura 2.1: Inicio de proyecto *WPF* en *Visual Studio*.

2.1 Diseño de la interfaz

Para comenzar se crea el diseño de la interfaz visual de la calculadora. El primer paso fue ajustar el tamaño de la calculadora a uno acorde, luego se fueron agregando los elementos necesarios desde el cuadro de herramientas, en este caso un elemento *Label* y 19 elementos de tipo *Button* (Figura 2.2).

El panel de propiedades permite modificar y dar estilo a cada elemento (Figura 2.3).

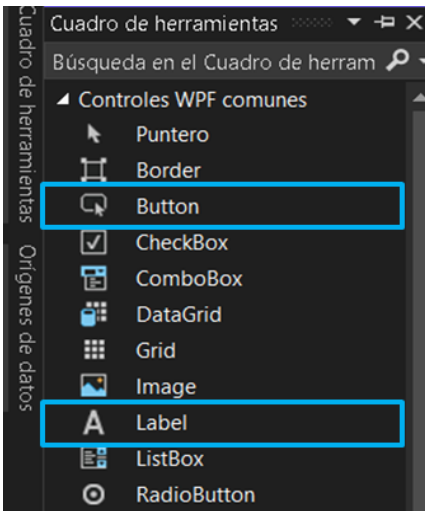

Figura 2.2: Cuadro de herramientas.

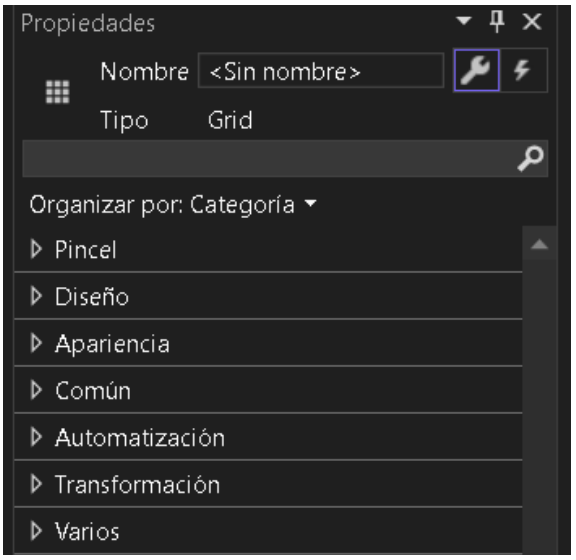

Figura 2.3: Panel de propiedades.

El color de fondo del proyecto fue personalizado con la herramienta Pincel y la opción de "Color sólido" en el Panel de propiedades (Figura 2.4).

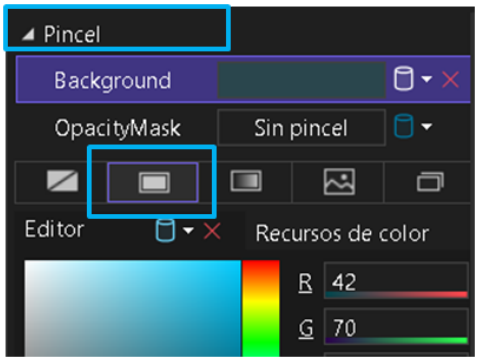

Figura 2.4: Opción Pincel del panel de propiedades.

La herramienta *Label* agregada constituye la pantalla de la calculadora, donde se muestran las operaciones y resultados. Se colocó, se ajustó y se eligió un color de fondo (Figura 2.5).

Figura 2.5: Pantalla de la calculadora.

En la parte superior del panel de propiedades se identificó el elemento con un nombre descriptivo para que posteriormente se pueda reconocer de manera sencilla a la hora de crear su código (Figura 2.6).

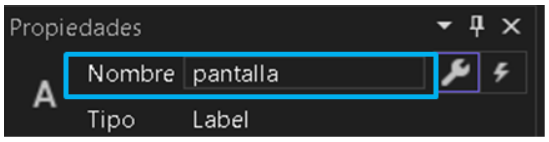

Figura 2.6: Sector en donde se identifica a los elementos de la aplicación.

Dentro de la opción “Común”/”Content” del panel de propiedades se permite agregar el contenido que estará a la vista por defecto. Allí se cambió ese valor del *Label*, en este caso se colocó un “0” (Figura 2.7).

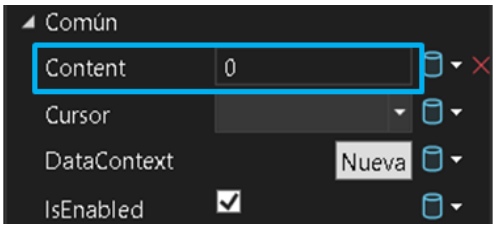

Figura 2.7: Opción en donde se cambia el contenido de la etiqueta *Label*.

En lo que respecta a los botones requeridos, se agregó y ajustó 19 de ellos correspondientes a: números del 0 al 9, suma, división, resta, multiplicación, símbolos como “=”, coma (,), porcentaje (%), cambio de signo (+/-) y limpiar pantalla (C). A cada uno se le destinó un nombre identificador, el contenido correspondiente a su valor y se les aplicó color y borde negro. Por último en la opción “Texto” del panel de propiedades se cambió el tipo de letra y el tamaño de todos los elementos (Figura 2.8).

Figura 2.8: Opción para dar formato a texto.

La siguiente imagen muestra el diseño final de la interfaz (Figura 2.9).

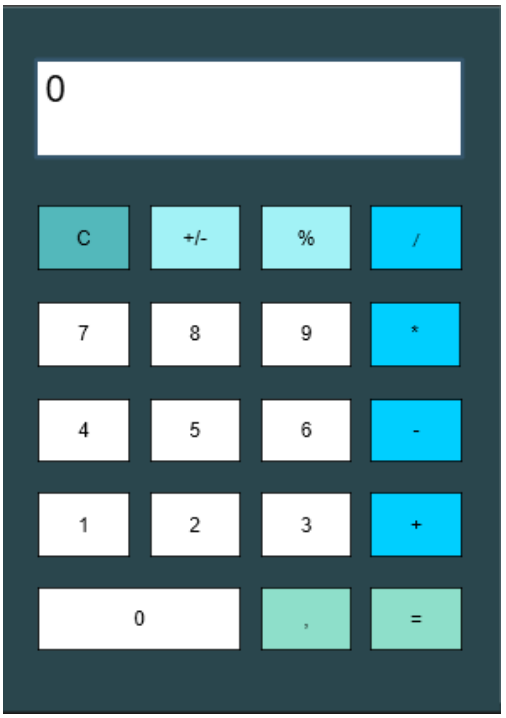

Figura 2.9: Interfaz de la calculadora.

2.2 Codificación del funcionamiento

Para aplicar funcionalidad a los botones se dió doble clic en cada uno, esto automáticamente creó una función por cada botón en la pestaña *MainWindow.xaml.cs,* entonces allí se desarrolló el funcionamiento correspondiente (Figura 2.10).

Figura 2.10: Pestañas del proyecto que contiene el diseño de interfaz y el código del funcionamiento.

Se comenzó con la declaración de la función de los botones que contienen los números, lo que corresponde a la agregación de estos en la pantalla de la calculadora (Figura 2.11).
A través de un método llamado AgregarNumero que recibe como parámetro un string equivalente al número seleccionado por el usuario, se valorará la siguiente acción.
.Content es una propiedad que se puede utilizar en varios elementos de la interfaz (botones, labels, etc.), que permite obtener o establecer el contenido de dicho elemento. En esta implementación, según el contenido existente en pantalla, o bien, se agrega el número solo, o lo que allí se encuentra y a continuación el número elegido (Figura 2.12).

```
0 referencias
private void uno_Click(object sender, RoutedEventArgs e)
{
    AgregarNumero("1");
}
```

Figura 2.11: Ejemplo de código que llevan los botones de los números.

```
void AgregarNumero (string numero)
{
    if ((string)pantalla.Content == "0")
    {
        pantalla.Content = numero;
    } else
    {
        pantalla.Content = pantalla.Content + numero;
    }
}
```

Figura 2.12: Método por el que se verifica cómo se coloca el número elegido en pantalla.

Se define el código de los botones que contienen a los operadores, se implementa la función AgregarOperador (explicada en la página 9) pasándole el que corresponde como un *string* por parámetro (Figura 2.13).

```
private void suma_Click(object sender, RoutedEventArgs e)
{
    AgregarOperador("+");
}
```

Figura 2.13: Ejemplo del código de los operadores.

Se creó la función del botón “C” que representa la acción borrar o limpiar pantalla. Al hacer clic en este botón, el contenido de la pantalla será 0 (Figura 2.14).

```
private void borrar_Click(object sender, RoutedEventArgs e)
{
    pantalla.Content = "0";
}
```

Figura 2.14: Código del botón que limpia la pantalla.

Se continuó con la funcionalidad del botón ".", que corresponde a la coma de un número. Se crea una variable en donde se extrae la información de la pantalla en tipo string y en base a eso con una estructura *if/else* se pregunta cómo es la terminación de dicho valor, si el caracter final es un espacio, quiere decir que antes hay un signo matemático, por lo que se agrega "0," de lo contrario, quiere decir que hay un número, entonces se agrega "," dando lugar a seguir agregando números (Figura 2.15).

```
private void coma_Click(object sender, RoutedEventArgs e)
{
    string c = (string)pantalla.Content;
    if (!(c.EndsWith(",")))
    {
        if (c.EndsWith(" "))
        {
            pantalla.Content = pantalla.Content + "0,";
        }
        else
        {
            pantalla.Content = pantalla.Content + ",";
        }
    }
}
```

Figura 2.15: Código de la agregación de la coma en pantalla.

A propósito del desarrollo del funcionamiento del botón de cambio de signo, primero se guarda el contenido de la pantalla en una variable "c" de tipo string y, mediante la propiedad *StartsWith,* se determina con qué valor comienza. Si el primer valor es un signo negativo, se iguala el contenido de la pantalla a una función en donde a través de la variable c se aplica el método *.Substring(1, c.Lenght -1)*, éste toma la medida de la cadena de caracteres recibida y con los datos especificados dentro del paréntesis se corta generando una más pequeña, comenzando en un índice específico y, opcionalmente, agregando una longitud. En este caso se quiere eliminar el índice 0 de la cadena, el signo, entonces la cadena resultante comienza en índice 1 y *c.Length -1* se refiere a la longitud que toma, que incluye al resto de los números presentes.

Si el número es positivo, para convertirlo en negativo basta con concatenar el caracter “-” antes del contenido de la pantalla (Figura 2.16).

```
private void signos_Click(object sender, RoutedEventArgs e)
{
    string c = (string)pantalla.Content;
    if (c.StartsWith("-"))
    {
        pantalla.Content = c.Substring(1, c.Length -1);
    } else
    {
        pantalla.Content = "-" + pantalla.Content;
    }

}
```

Figura 2.16: Visualización del funcionamiento del cambio de signo de un número.

Para dar operatividad al botón porcentaje, se guarda en una variable el contenido de la pantalla y luego mediante if se pregunta si en el contenido existe un espacio, lo cual significa que hay una operación, se calcula el porcentaje del segundo número por lo que se crea otra variable, se extrae la cadena de caracteres y se busca el índice del último espacio hasta el final del segundo número, luego se convierte en un float con otra variable, se divide entre cien y se agrega el contenido de esa variable, convirtiéndola a string, al contenido de la pantalla que toma desde el índice 0 hasta el último índice en donde aparece un espacio. Pero si no contiene un espacio se crea una variable en donde se convierte de string a float, se divide entre cien y se agrega a la pantalla convirtiéndola de nuevo a string (Figura 2.17).

```
private void porcentaje_Click(object sender, RoutedEventArgs e)
{
    string c = (string)pantalla.Content;
    if (c.Contains(" "))
    {
        string ch = c.Substring(c.LastIndexOf(" ", c.Length - 1));
        float f = float.Parse(ch);
        f = f / 100;
        pantalla.Content = c.Substring(0, c.LastIndexOf(" ") + 1) + f.ToString();
    } else
    {
        float f = float.Parse(c);
        f = f / 100;
        pantalla.Content =f.ToString();
    }
}
```

Figura 2.17: Visualización de la aplicación porcentaje a un número.

A continuación se creó un método que permite agregar operadores. Recibe como parámetro el caracter correspondiente al operador elegido.

Se asigna el valor de la pantalla como string a una variable auxiliar “c”. Cuando se agrega un operador se pregunta si c termina en espacio, si ocurre esto significa que lo último que se introdujo fue un operador, por lo que a través del método *.Substring* se extrae la cadena de caracteres del primer número, comenzando en 0 y con *IndexOf* se busca el primer espacio para quedarse con esa porción, eliminar lo siguiente y así poder cambiarlo por el último operador seleccionado. En cambio, si no existe esta posibilidad significa que hay una operación, entonces se realiza esta con la función calcular() y se coloca el resultado seguido de espacio más el operador seleccionado más espacio (Figura 2.18).

```
void AgregarOperador(string op)
{
    string c = (string)pantalla.Content;
        if (c.EndsWith(" "))
        {
            pantalla.Content = c.Substring(0, c.IndexOf(" ")) + " " + op + " ";
        } else
        {
            if (c.Contains("+") || c.Contains("-") || c.Contains("*") || c.Contains("/"))
            {
                Calcular();
            }
            pantalla.Content = pantalla.Content + " " + op + " ";
        }
}
```

Figura 2.18: Método que implementa el operador y su operación asociada.

La función calcular realiza las operaciones en base al operador que se encuentra en la pantalla. Toma el contenido de la pantalla como cadena de caracteres, la separa en 3 partes buscando los espacios con el método .Split, guarda las partes como elementos en un array y toma la segunda parte, es decir el índice 1, para compararlos con los caracteres de operadores en una estructura if/else. Dependiendo de qué condición se cumpla, se ejecuta el cálculo correspondiente. El resultado que surge se guarda en una variable “resultado”, se convierte a string mediante el método .ToString y se muestra en pantalla (Figura 2.19).

```
void Calcular()
{
    string c = (string)pantalla.Content;
    string[] partes = c.Split(' ');
    float a = float.Parse(partes[0]);
    float b = float.Parse(partes[2]);
    float resultado = 0;
    if (partes[1] == "+")
    {
        resultado = a + b;
    } else if (partes[1] == "-")
    {
        resultado = a - b;
    } else if (partes[1] == "*")
    {
        resultado = a * b;
    } else if (partes[1] == "/")
    {
        resultado = a / b;
    }
    pantalla.Content = resultado.ToString();
}
```

Figura 2.19: Método que realiza el cálculo correspondiente al contenido de la pantalla.

Por último se desarrolló el código del botón de resultado "=", donde se toma la cadena de caracteres de la pantalla en una variable string y se comienza preguntando si termina con un espacio, en ese caso se coloca en la pantalla la cadena de caracteres desde el índice 0 hasta el primer espacio, de modo que deja el contenido de la pantalla igual, pero de no ser así se aplica directamente la función Calcular() (Figura 2.20).

```
private void igual_Click(object sender, RoutedEventArgs e)
{
    string c = (string)pantalla.Content;
    if (c.EndsWith(" "))
    {
            pantalla.Content = c.Substring(0, c.IndexOf(" "));
    }
    else
    {
        Calcular();
    }
}
```

Figura 2.20: Funcionalidad del botón resultado (=).

Puntos a tener en cuenta al momento del uso de la calculadora:

1. Siempre dos operandos generan un resultado, luego del primero se podrá continuar trabajando sobre ese resultado, y cada vez que se presione un operador, se aplica al resultado anterior.

2. Se puede cambiar de operador si se seleccionó uno incorrecto, no es necesario que se borre todo. Al ingresar dos operadores correlativos, se realiza la operación con el último.

3. El porcentaje se calcula directamente al número ingresado como operando.

3. Resultados

-> Para diseñar la interfaz se utilizaron elementos desde el cuadro de herramientas (Figura 3.1) y fueron modificados desde el panel de propiedades (Figura 3.2)

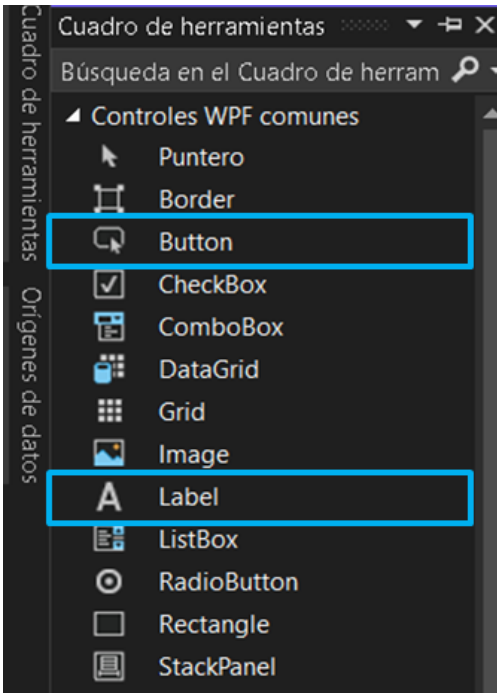

Figura 3.1

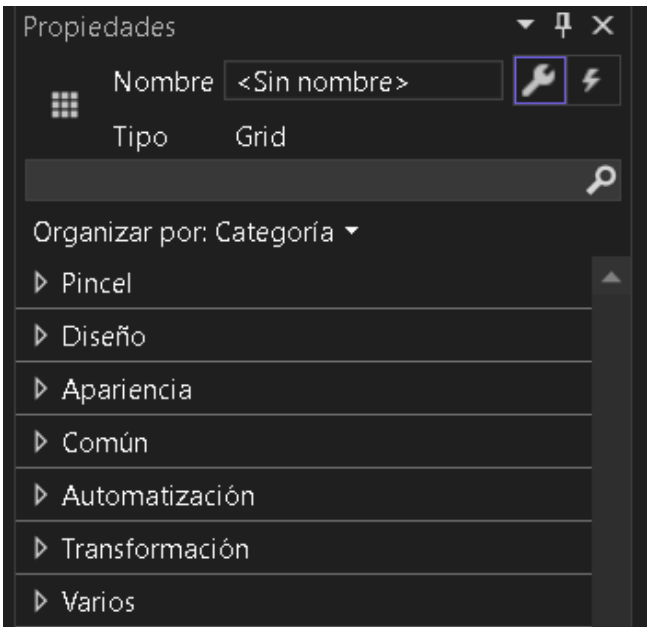

Figura 3.2

-> Diseño final de Interfaz de la calculadora (Figura 3.3)

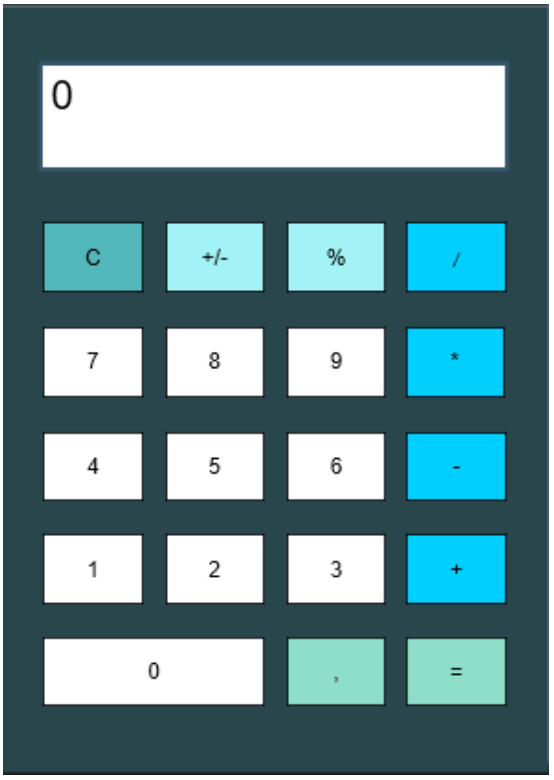

Figura 3.3

-> Método para agregar operadores básicos (Figura 3.4)

```
void AgregarOperador(string op)
{
    string c = (string)pantalla.Content;
        if (c.EndsWith(" "))
        {
            pantalla.Content = c.Substring(0, c.IndexOf(" ")) + " " + op + " ";
        } else
        {
            if (c.Contains("+") || c.Contains("-") || c.Contains("*") || c.Contains("/"))
            {
                Calcular();
            }
            pantalla.Content = pantalla.Content + " " + op + " ";
        }
}
```

Figura 3.4

-> Método para realizar cálculos básicos (Figura 3.5)

```
void Calcular()
{
    string c = (string)pantalla.Content;
    string[] partes = c.Split(' ');
    float a = float.Parse(partes[0]);
    float b = float.Parse(partes[2]);
    float resultado = 0;
    if (partes[1] == "+")
    {
        resultado = a + b;
    } else if (partes[1] == "-")
    {
        resultado = a - b;
    } else if (partes[1] == "*")
    {
        resultado = a * b;
    } else if (partes[1] == "/")
    {
        resultado = a / b;
    }
    pantalla.Content = resultado.ToString();
}
```

Figura 3.5

-> Método que da funcionamiento al botón "%" para calcular el porcentaje de un número (Figura 3.6)

```
private void porcentaje_Click(object sender, RoutedEventArgs e)
{
    string c = (string)pantalla.Content;
    if (c.Contains(" "))
    {
        string ch = c.Substring(c.LastIndexOf(" ", c.Length - 1));
        float f = float.Parse(ch);
        f = f / 100;
        pantalla.Content = c.Substring(0, c.LastIndexOf(" ") + 1) + f.ToString();
    } else
    {
        float f = float.Parse(c);
        f = f / 100;
        pantalla.Content =f.ToString();
    }
}
```

Figura 3.6

-> Método que da funcionamiento al botón “+/-” para cambiar el signo del número en pantalla (Figura 3.7)

```
private void signos_Click(object sender, RoutedEventArgs e)
{
    string c = (string)pantalla.Content;
    if (c.StartsWith("-"))
    {
        pantalla.Content = c.Substring(1, c.Length -1);
    } else
    {
        pantalla.Content = "-" + pantalla.Content;
    }

}
```

Figura 3.7

4. Conclusiones

La utilización de *WPF* simplifica el diseño de aplicaciones mediante su interfaz y el sistema de código de etiquetas *XAML*. Es capaz de recibir la definición de dichas etiquetas mediante código escrito y también permite la auto-generación de éstas al utilizar directamente las herramientas disponibles, como el *Button* y *Label* manejadas en el desarrollo de la calculadora.

Al comenzar un proyecto se generan automáticamente dos archivos *XAML* relacionados entre sí, uno destinado al diseño y otro al código, éste último recibirá la definición de los métodos necesarios para el funcionamiento del sistema. Al correr la aplicación, éstos dos archivos se fusionan dando como resultado una interfaz agradable para el usuario y a su vez funcional, en este caso podrá ser utilizada para realizar varios tipos de cálculos.

Este tipo de proyectos deja abierta la posibilidad de ser extendidos y modificados en el futuro con una amplia variedad de posibilidades gracias a las potentes herramientas de WPF.

5. Referencias

[1] - https://www.cursosaula21.com/que-es-net/

[2] - https://programarfacil.com/blog/programacion-net-blog/que-es-wpf/#:~:text=WPF%20permite%20crear%20interfaces%20de,potente%20que%20el%20propio%20HTML