



# TECNOLÓGICO NACIONAL DE MÉXICO INSTITUTO TECNOLÓGICO DE TEPIC



*Tepic Nayarit; 01 de Noviembre 2021*

**UNIDAD 2:** Desarrollo de interfaces.

**ACTIVIDAD 2:** MicroProyecto AutoMobile.

**MATERIA:** DESARROLLO DE APLICACIONES HÍBRIDAS

**DOCENTE:** FRANCISCO IBARRA CARLOS

## **Equipo "Los Híbridos"**

**Becerra Enríquez Carlos Alberto** 17400952

**Burgara Ortega Brenda Karime** 17400957

**González Guzmán Liliana Elizabeth** 16400919

**Tovar Andrade Juan Eduardo** 17401087

**Varela García Jorge Jorge** 16401013

**Verdín Carreño Ximena** 17401092

**SEMESTRE:** 9no

**GRUPO:** A

**HORA:** 20:00 a 21:00

**CARRERA:** INGENIERÍA EN SISTEMAS COMPUTACIONALES



## Introducción

En este presente documento se estará abordando todo el procedimiento de la actividad 2 de la materia aplicaciones híbridas. Dicha actividad será realizada con todos los conocimientos adquiridos de las pasadas actividades de anteriores unidades y unidad actual, por ende, se considera esta actividad como un microproyecto ya que se estaremos utilizando clases con constructores, disposición grid, el uso de objetos y el uso de ciclos.

Orientada principal mente en una empresa de carros donde podremos guardar los datos del automóvil como lo es la marca y el modelo, en dicha información existen una serie de restricciones las cuales indican que no debe de haber espacios en blanco para la marca y el modelo y que el año de registro del automóvil no sea menor a 2000 y de igual manera que no sea mayor a dos años de la fecha actúa, por ultima condición es indicar que no se acepta valores nulos.

Como podemos ver el objetivo de esta actividad es abordar los conocimientos anteriormente adquiridos para el manejo y uso de objetos codificados, nuevamente apoyándonos de las herramientas que Xamarin nos ofrece y desarrollándolo mediante Visual Studio Code.

Al igual como se ha ido desarrollando en entregas anteriores, se tomará una captura de pantalla con el desarrollo del programa y mediante se vaya codificando y desarrollando se dará su debida narración mencionando como fueron los pasos que fueron tomando para lo que se está solicitando, anexándolo junto con las capturas de pantallas.

También cabe mencionar que en el trabajo realizado en práctica se suman, diversos collages de imágenes que represan talas ideas principales de lo que se menciona y se desarrolla, de igual manera se anexan palabras clave en los encabezados. También otra particularidad que tendrá en presente documento es que contara con un glosario de términos, los cuales, por parte del equipo, se consideran de los conceptos más importantes comprendidos en el desarrollo de esta actividad.

Por último, se hacen mención de algunas referencias que fueron consultadas a lo largo de este proceso, para poder ofrecer un acceso sencillo a la información consultada. Esperando a que en esta presente entrega sirva de lectura de referencia al lector.

## Contenido

Introducción.....	1
Contenido .....	3
A) Crea la clase “FicAutoMobile” dentro del directorio “Class/Auto”. Esta clase deberá tener las siguientes propiedades: Make (marca), Model (modelo) y Year (año). .....	4
B) Crear el constructor de la clase con la que crearemos los Autos.....	5
C) Al crear un objeto sobre esta clase debemos validar en el constructor lo siguiente: 1) La marca y el modelo no deben aceptar valores en nulo. 2) La marca y el modelo no pueden contener espacios. 3) El año del auto no debe ser menor a al año 2000 ni tampoco 2 años mayor al año actual. 4) En caso de que no se cumpla con estas validaciones no se deberá permitir crear el objeto (auto) y se deberán mandar las excepciones correspondientes. ....	5
D) Crear la vista “FicViAutosList.xalm” de tipo “Content Page” y su respectivo constructor. ....	7
E) Crear al menos 5 autos (objetos) y mandar su información a consola con su respectiva evidencia (esta evidencia en consola es por cada objeto).....	9
F) Crear los autos (objetos) necesarios para validar todas y cada una de las excepciones descritas y que debieron ser programadas en el inciso C. Mostrar evidencia de cada una de estas validaciones (al menos 6 evidencias enumeradas y con su descripción detallada y su salida en consola). ....	11
G) Realice la codificación necesaria tanto en la clase como en la vista para que cada que se cree un auto (5 objetos del inciso [E]) este sea agregado a una lista de tipo clase “FicAutoMobile” con el nombre de “FicAllAutoMobileList”.....	14
H) Diseña una vista con sus respectivo controles como son: Titulo (Label), buscador (SearchBar) y su grid (SfDataGrid) en donde despliegues o muestres la información de los 5 objetos (autos) que fueron agregados a la lista “FicAllAutoMobileList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.....	15
Conclusión .....	19
Glosario .....	20
Referencias .....	22

## Contenido

3.4 Listas.

3.5 Navegación.

3.6 Modales.

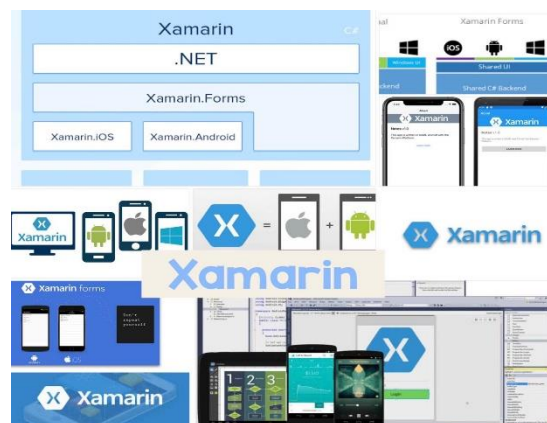
### Actividad 2. MicroProyecto AutoMobile.

- A) Crea la clase “FicAutoMobile” dentro del directorio “Class/Auto”. Esta clase deberá tener las siguientes propiedades: Make (marca), Model (modelo) y Year (año).
- B) Crear el constructor de la clase con la que crearemos los Autos.
- C) Al crear un objeto sobre esta clase debemos validar en el constructor lo siguiente:
  - 1. La marca y el modelo no deben aceptar valores en nulo.
  - 2. La marca y el modelo no pueden contener espacios.
  - 3. El año del auto no debe ser menor a al año 2000 ni tampoco 2 años mayor al año actual.
  - 4. En caso de que no se cumpla con estas validaciones no se deberá permitir crear el objeto (auto) y se deberán mandar las excepciones correspondientes.
- D) Crear la vista “FicViAutosList.xalm” de tipo “Content Page” y su respectivo constructor.
- E) Crear al menos 5 autos (objetos) y mandar su información a consola con su respectiva evidencia (esta evidencia en consola es por cada objeto).
- F) Crear los autos (objetos) necesarios para validar todas y cada una de las excepciones descritas y que debieron ser programadas en el inciso C. Mostrar evidencia de cada una de estas validaciones (al menos 6 evidencias enumeradas y con su descripción detallada y su salida en consola).
- G) Realice la codificación necesaria tanto en la clase como en la vista para que cada que se cree un auto (5 objetos del inciso [E]) este sea agregado a una lista de tipo clase “FicAutoMobile” con el nombre de “FicAllAutoMobileList”.
- H) Diseña una vista con sus respectivo controles como son: Titulo (Label), buscador (SearchBar) y su grid (SfDataGrid) en donde despliegues o muestres la información de los 5 objetos (autos) que fueron agregados a la lista “FicAllAutoMobileList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.

*Nota: Recuerda que para cada inciso debes enviar evidencia detallada en la que expliques el proceso de desarrollo y los motivos del porque es necesaria cada bloque o línea de código. No olvides aplicar el formato de tareas al 100% las imágenes de evidencia de tu desarrollo no sustituyen a los colash de imágenes que se piden en formato de tareas.*

Modular programming it's a methodology adopted by developers to avoid long and complex programs. For this is the code becomes into modules or subprograms controlled by a principal module, this a loud to make it more legible and easier to handle.

Equipo “Los Híbridos”  
Desarrollo de Aplicaciones Híbridas.  
Unidad #3: Desarrollo de Interfaces.  
Docente: Francisco Ibarra Carlos  
ISC. Instituto Tecnológico de Tepic.



**A) Crea la clase “FicAutoMobile” dentro del directorio “Class/Auto”. Esta clase deberá tener las siguientes propiedades: Make (marca), Model (modelo) y Year (año).**

**Paso 1: creación del proyecto AutoMobile.**

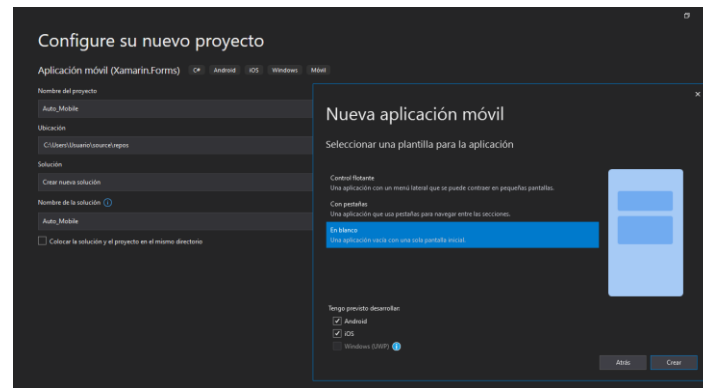


Ilustración 1: Paso 1, creación del proyecto.

**Paso 2: clase FicAutoMobile.** Tal como en proyectos anteriores, damos clic derecho sobre la solución, *Agregar, Nueva Carpeta*, y la nombramos *Class*. En esta carpeta, damos clic derecho, *Agregar, Nueva Carpeta* y esta es nombrada *Auto*. Finalmente, clic derecho en dicha carpeta, *Agregar, Nuevo Elemento, Clase* y la nombramos *FicAutoMobile*, quedando todos los movimientos de la siguiente manera:

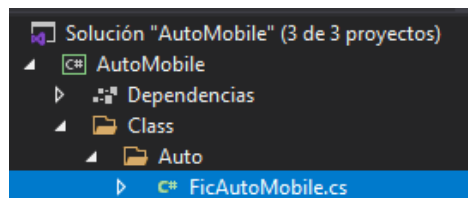


Ilustración 2: Paso 2, clase FicAutoMobile.

**Paso 3: propiedades.** Dentro de la clase, agregaremos las propiedades *Make*, de tipo *String* que representará la marca del auto, *Model*, de tipo *String* que representará el modelo del auto y *Year*, de tipo *int* que representará el año del auto.

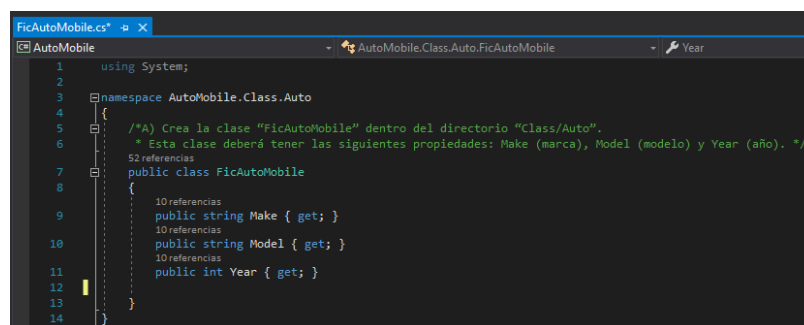


Ilustración 3: Paso 3, propiedades.



## B) Crear el constructor de la clase con la que crearemos los Autos.

**Paso 1: constructor.** Creamos el constructor de la clase, el cual recibirá tres parámetros. El primero, MakeD de tipo *String* que recibirá la marca del auto, ModelD de tipo *String* que recibirá el modelo del auto y, finalmente, YearD de tipo *int* que recibirá el año del auto.

```
/*B) Crear el constructor de la clase con la que crearemos los Autos. */
22 referencias
public FicAutoMobile(string MakeD, string ModelD, int YearD)
{
    ...
}
```

Ilustración 4: Paso 1, constructor.

The C# programs are formed by one or more files, that may contain namespaces. Those namespaces contain structures, classes, and other useful items. In this project we work with the namespaces and the classes, this to create the object and to give them an instance in the program.



**C) Al crear un objeto sobre esta clase debemos validar en el constructor lo siguiente: 1) La marca y el modelo no deben aceptar valores en nulo. 2) La marca y el modelo no pueden contener espacios. 3) El año del auto no debe ser menor a al año 2000 ni tampoco 2 años mayor al año actual. 4) En caso de que no se cumpla con estas validaciones no se deberá permitir crear el objeto (auto) y se deberán mandar las excepciones correspondientes.**

**Paso 1: espacios y nulos en marca.** Para resolver los espacios y los nulos en el parámetro MakeD que recibirá el constructor, se utiliza un condicional *if-else if*.

La primera condición que se resuelve es el parámetro nulo. Si *MakeD* es nulo, enviará una excepción de tipo *ArgumentNullException* con el mensaje “The Make can’t be null”, que significa “La marca no puede ser nula”.

De no enviar esa excepción, entrará a la validación *String.IsNullOrEmpty(MakeD)* para verificar los espacios en blanco. Si entra en esta excepción, enviará un mensaje de tipo *ArgumentException* con el mensaje *"The Make can't be empty and with whitespace."*, que significa "La marca no puede estar vacía y con espacios en blanco".

Si no envía ninguna de las excepciones, se asigna la marca del registro a la que recibió el constructor con el comando *this.Make=MakeD*.

```
/*C) Al crear un objeto sobre esta clase debemos validar en el constructor lo siguiente:
 *     La marca y el modelo no deben aceptar valores en nulo.
 *     La marca y el modelo no pueden contener espacios.*/

if (MakeD == null)
    throw new ArgumentNullException("The Make can't be null");
else if (String.IsNullOrEmpty(MakeD))
    throw new ArgumentException("The Make can't be empty and with whitespace.");
this.Make = MakeD;
```

Ilustración 5: Paso 1, espacios y nulos en marca.

**Paso 2: espacios y nulos en modelo.** Para resolver los espacios y los nulos en el parámetro *ModelD* que recibirá el constructor, se utiliza un condicional *if-else if*.

La primera condición que se resuelve es el parámetro nulo. Si *ModelD* es nulo, enviará una excepción de tipo *ArgumentNullException* con el mensaje *"The Model can't be null"*, que significa "El modelo no puede ser nula".

De no enviar esa excepción, entrará a la validación *String.IsNullOrEmpty(ModelD)* para verificar los espacios en blanco. Si entra en esta excepción, enviará un mensaje de tipo *ArgumentException* con el mensaje *"The Model can't be empty and with whitespace."*, que significa "La marca no puede estar vacía y con espacios en blanco".

Si no envía ninguna de las excepciones, se asigna la marca del registro a la que recibió el constructor con el comando *this.Model=ModelD*.

```
if(ModelD == null)
    throw new ArgumentNullException("The Model can't be null");
else if (String.IsNullOrEmpty(ModelD))
    throw new ArgumentException("The Model can't be empty and with whitespace.");
this.Model = ModelD;
```

Ilustración 6: Paso 2, espacios y nulos en modelo.

**Paso 3: rango de año.** Para resolver el rango de validez del año, siendo que no puede ser de antes del año 2000 ni posterior a 2 años del actual, en un condicional *if*. En este, especificamos que, si el parámetro *YearD* no incumpla con alguna de las dos condiciones con un "OR". Si la validación es verdadera (es decir, si las condiciones son infringidas), se

enviará una excepción de tipo *ArgumentException* con el mensaje *"The Year is out of range."*, que significa *"El Año está fuera del rango"*.

En caso contrario, se asigna el año del registro al que recibió el constructor con el comando *this.Year=YearD*.

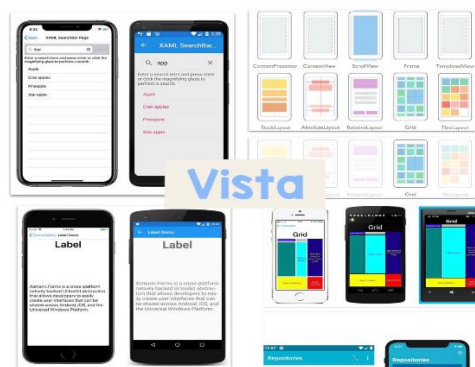
```
/* El año del auto no debe ser menor a al año 2000 ni tampoco 2 años mayor al año actual.
 * En caso de que no se cumpla con estas validaciones no se deberá permitir crear el objeto
   (auto) y se deberán mandar las excepciones correspondientes. */

if (YearD < 2000 || YearD > DateTime.Now.Year + 2)
    throw new ArgumentException("The Year is out of range.");
this.Year = YearD;
```

Ilustración 7: Paso 3, rango de año.

#### D) Crear la vista "FicViAutosList.xaml" de tipo "Content Page" y su respectivo constructor.

The MVP model keeps being very useful in the implementation of the all app, this through the components define in the xaml, the class that define the object and the code in the main class that bring all to function.



**Paso 1: vista FicViAutosList.** Tal como en proyectos anteriores, damos clic derecho sobre la solución, *Agregar, Nueva Carpeta*, y la nombramos *Views*. En esta carpeta, damos clic derecho, *Agregar, Nueva Carpeta* y esta es nombrada *Auto*. Finalmente, clic derecho en dicha carpeta, *Agregar, Nuevo Elemento, Página de Contenido* y la nombramos *FicViAutosList* quedando todos los movimientos de la siguiente manera:

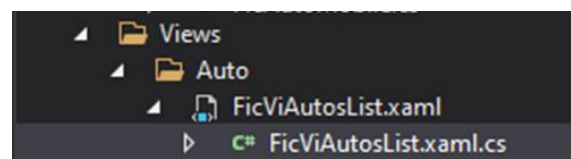
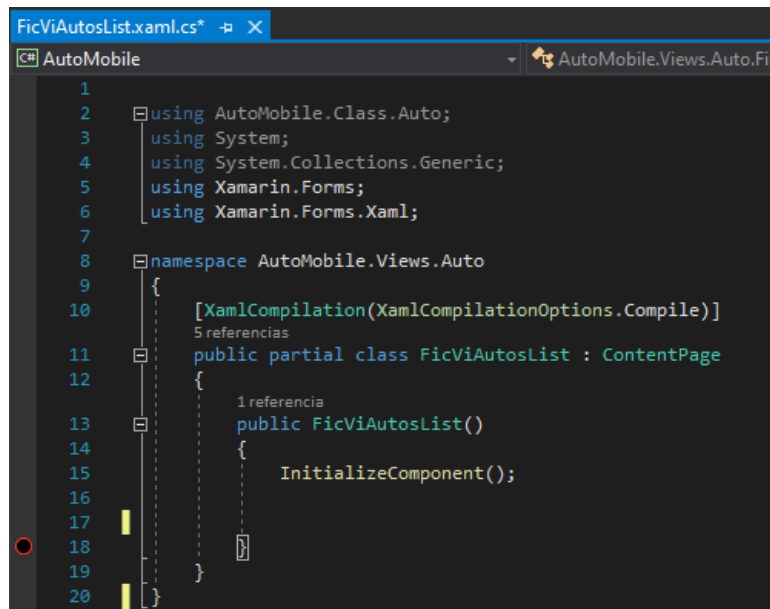


Ilustración 8: Paso 1, vista FicViAutosList.

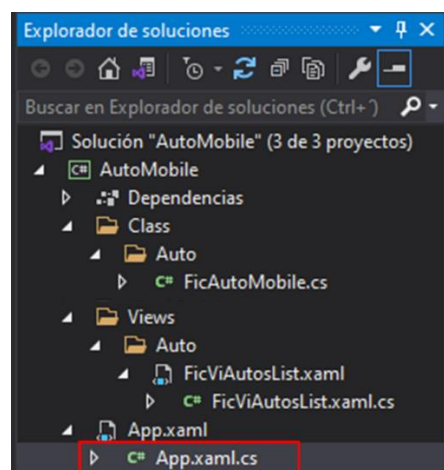


**Paso 2: constructor.** En el archivo *FicViAutosList.xaml.cs*, creamos el constructor de la clase de la siguiente manera, para, en este, se realicen las codificaciones para los pasos posteriores.



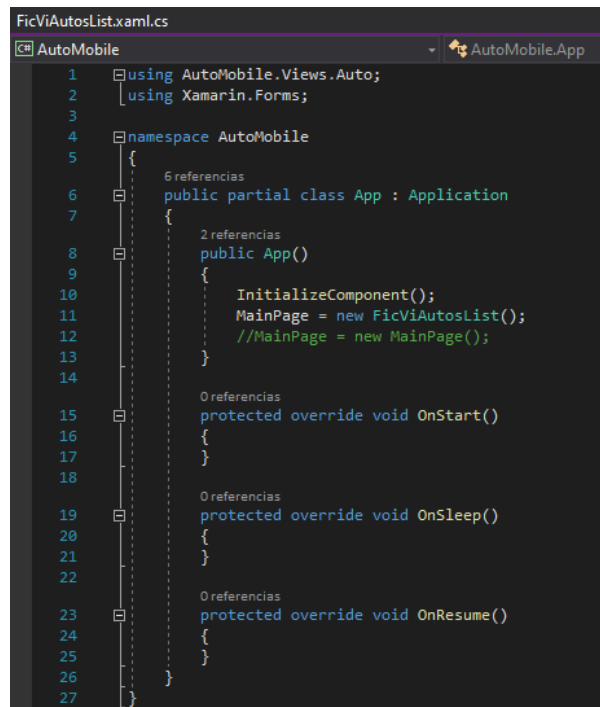
*Ilustración 9: Paso 2, constructor.*

**Paso 3: MainPage.** Para que, al ejecutar la aplicación, sea mostrada la vista *FicViAutosList.xaml*, debemos hacer una modificación dentro del archivo *App.xaml.cs*.



*Ilustración 10: Paso 3, MainPage.*

**Paso 4: cambio en el constructor.** En el constructor de *App.xaml.cs* comentamos la línea *MainPage = new MainPage();*, reemplazándola con la línea *MainPage = new FicViAutosList();*. Con este cambio, *FicViAutosList.xaml* será la pantalla que aparezca al ejecutar la aplicación.



```
1 using AutoMobile.Views.Auto;
2 using Xamarin.Forms;
3
4 namespace AutoMobile
5 {
6     public partial class App : Application
7     {
8         public App()
9         {
10             InitializeComponent();
11             MainPage = new FicViAutosList();
12             //MainPage = new MainPage();
13         }
14
15         protected override void OnStart()
16         {
17         }
18
19         protected override void OnSleep()
20         {
21         }
22
23         protected override void OnResume()
24         {
25         }
26     }
27 }
```

Ilustración 11: Paso 4, cambio en el constructor.

**E) Crear al menos 5 autos (objetos) y mandar su información a consola con su respectiva evidencia (esta evidencia en consola es por cada objeto).**

The methods in C# may be instance or static. The static method is already defined like a set of code that need a type to be called and the instance one needs an object to be summoned. It's important to acknowledge the complexity to define them, within parameters and the types that may accept and return like an output.



**Paso 1: clase FicViAutosList.** Dentro de la carpeta *Auto*, desplegando el archivo *FicViAutosList.xaml*, encontraremos la clase de dicho archivo *FicViAutosList.xaml.cs*.

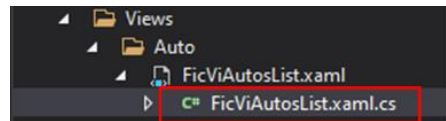


Ilustración 12: Paso 1, clase *FicViAutosList*.

**Paso 2: creación de los objetos.** En la clase mencionada en el paso previo, crearemos cinco objetos mandando llamar el constructor *FicAutoMobile* con los tres parámetros, e imprimiendo cada resultado en consola.

```
1 referencia
public FicViAutosList()
{
    InitializeComponent();

    ///E.Crear al menos 5 autos (objetos) y mandar su información a consola con su
    ///respectiva evidencia (esta evidencia en consola es por cada objeto).
    FicAutoMobile Auto1 = new FicAutoMobile("Audi", "A8", 2020);
    Console.WriteLine(
        $"Make: {Auto1.Make} " +
        $"Model: {Auto1.Model} " +
        $"Year: {Auto1.Year} "
    );

    FicAutoMobile Auto2 = new FicAutoMobile("Chevrolet", "Aveo", 2021);
    Console.WriteLine(
        $"Make: {Auto2.Make} " +
        $"Model: {Auto2.Model} " +
        $"Year: {Auto2.Year} "
    );

    FicAutoMobile Auto3 = new FicAutoMobile("Jeep", "Wrangler", 2022);
    Console.WriteLine(
        $"Make: {Auto3.Make} " +
        $"Model: {Auto3.Model} " +
        $"Year: {Auto3.Year} "
    );

    FicAutoMobile Auto4 = new FicAutoMobile("Rolls-Royce", "Wraith", 2022);
    Console.WriteLine(
        $"Make: {Auto4.Make} " +
        $"Model: {Auto4.Model} " +
        $"Year: {Auto4.Year} "
    );

    FicAutoMobile Auto5 = new FicAutoMobile("Lexus", "IS", 2021);
    Console.WriteLine(
        $"Make: {Auto5.Make} " +
        $"Model: {Auto5.Model} " +
        $"Year: {Auto5.Year} "
    );
}
```

Ilustración 13: Paso 2, creación de los objetos, código.

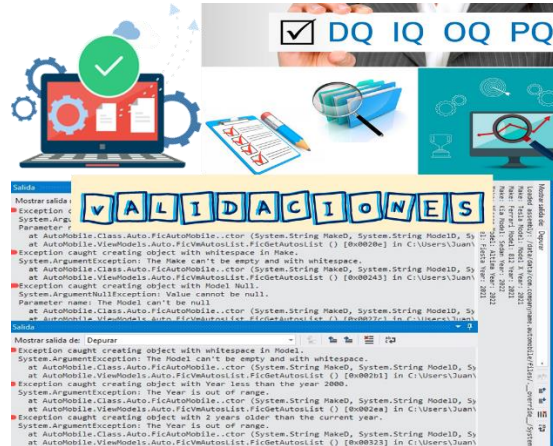
*Salida en consola.*

```
Salida
Mostrar salida de: Depurar
Make: Audi Model: A8 Year: 2020
Make: Chevrolet Model: Aveo Year: 2021
Make: Jeep Model: Wrangler Year: 2022
Make: Rolls-Royce Model: Wraith Year: 2022
Make: Lexus Model: IS Year: 2021
```

Ilustración 14: Paso 2, creación de los objetos, salida.

**F) Crear los autos (objetos) necesarios para validar todas y cada una de las excepciones descritas y que debieron ser programadas en el inciso C. Mostrar evidencia de cada una de estas validaciones (al menos 6 evidencias enumeradas y con su descripción detallada y su salida en consola).**

The validation in the parameters of method is really helpful when we need to create objects in a secure way. This through POO and the method “set” are very useful and easy to use with some considerations, an example are the methods that avoid non valid years in the cars of this app.



**Paso 1: Make null.** La primera validación que se verifica es la de una marca null. Para ello, creamos una variable “validación1” enviando los parámetros de un auto cuya marca detallamos como “null”. A continuación, el código y el resultado en consola:

*Código:*

```
//VALIDACIÓN 1: Make null
try..
{
    FicAutoMobile validacion1 = new FicAutoMobile(null,"A110",2021);
}..
catch (ArgumentNullException e)
{
    Console.WriteLine("Exception caught creating object with Make Null.");
    Console.WriteLine(e.ToString());
}
```

Ilustración 15: Paso 1, Make null, código.

*Salida en consola:*

```
Salida
Mostrar salida de: Depurar
Exception caught creating object with Make Null.
System.ArgumentNullException: Value cannot be null.
Parameter name: The Make can't be null
at Automobile.Class.Auto.FicAutoMobile..ctor (System.String MakeD, System.St
\Auto\FicAutoMobile.cs:22
at Automobile.Views.Auto.FicViAutosList..ctor () [0x0021c] in C:\Users\Usuar
```

Ilustración 16: Paso 1, Make null, consola.

**Paso 2: Make con espacios en blanco.** La segunda validación que se verifica es la de una marca con espacios. Para ello, creamos una variable “validación2” y enviaremos los parámetros de un auto cuya marca detallamos como “ ”. A continuación, el código y el resultado en consola:

*Código:*

```
////VALIDACIÓN 2: Make with whitespace
try
{
    FicAutoMobile validacion2 = new FicAutoMobile("Aston Martin", "Mulsanne", 2022);
}
catch (ArgumentException e)
{
    Console.WriteLine("Exception caught creating object with whitespace in Make.");
    Console.WriteLine(e.ToString());
}
```

*Ilustración 17: Paso 2, Make con espacios en blanco, código.*

*Salida en consola:*

*Ilustración 18: Paso 2, Make con espacios en blanco, consola.*

**Paso 3: Model null.** La tercer validación que se verifica es la de un modelo null. Para ello, creamos una variable “validación3” enviando los parámetros de un auto cuyo modelo detallamos como “null”. A continuación, el código y el resultado en consola:

*Código:*

```
////VALIDACIÓN 3: Model null
try
{
    FicAutoMobile validacion3 = new FicAutoMobile("BMW",null, 2021);
}
catch (ArgumentNullException e)
{
    Console.WriteLine("Exception caught creating object with Model Null.");
    Console.WriteLine(e.ToString());
}
```

*Ilustración 19: Paso 3, Model null, código.*

*Salida en consola:*

*Ilustración 20: Paso 3, Model null, consola.*



**Palabras Clave:** Verificación de Validaciones, Código, Salida en Consola, Espacios en Blanco, Rango de Años.

**Actividad #2: 01/11/2021**

Página 13 de 22

**Paso 4: Model con espacios en blanco.** La cuarta validación que se verifica es la de un modelo con espacios. Para ello, creamos una variable “validación4” y enviaremos los parámetros de un auto cuyo modelo detallamos como “ ”. A continuación, el código y el resultado en consola:

*Código:*

```
////VALIDACIÓN 4: Model with whitespace
try
{
    FicAutoMobile validacion4 = new FicAutoMobile("Bentley","Motors Limited", 2022);
}
catch (ArgumentException e)
{
    Console.WriteLine("Exception caught creating object with whitespace in Model.");
    Console.WriteLine(e.ToString());
}
```

*Ilustración 21: Paso 4, Model con espacios en blanco, código.*

*Salida en consola:*

A screenshot of a Windows console window titled "Salida". The window shows the output of a C# application. The first line is "Exception caught creating object with whitespace in Model." followed by a detailed exception message: "System.ArgumentException: The Model can't be empty and with whitespace." The stack trace indicates the error occurred in the constructor of the "FicAutoMobile" class at line 30 of "FicAutoMobile.cs".

*Ilustración 22: Paso 4, Model con espacios en blanco, consola.*

**Paso 5: año menor al 2000.** La siguiente validación verifica el año del auto, siendo este anterior al 2000. Para ello, creamos una variable “validacion5” y enviamos los parámetros de un auto cuyo año detallamos como “1987”. A continuación, el código y el resultado en consola:

*Código:*

```
////VALIDACIÓN 5: The car is less than the year 2000.
try
{
    FicAutoMobile validacion5 = new FicAutoMobile("Ferrari", "F40", 1987);
}
catch (ArgumentException e)
{
    Console.WriteLine("Exception caught creating object with Year less than the year 2000.");
    Console.WriteLine(e.ToString());
}
```

*Ilustración 23: Paso 5, año menor al 2000, código.*

*Salida en consola:*

A screenshot of a Windows console window titled "Salida". The window shows the output of a C# application. The first line is "Exception caught creating object with Year less than the year 2000." followed by a detailed exception message: "System.ArgumentException: The Year is out of range." The stack trace indicates the error occurred in the constructor of the "FicAutoMobile" class at line 38 of "FicAutoMobile.cs".

*Ilustración 24: Paso 5, año menor al 2000, consola.*

**Palabras Clave:** Verificación de Validaciones, Código, Salida en Consola, Espacios en Blanco, Rango de Años.

**Actividad #2: 01/11/2021**

Página 14 de 22

**Paso 6: año mayor a dos años del actual.** La última validación verifica el año del auto, siendo este más de dos años posterior al actual (2021 en este caso, el límite debe ser 2023). Para ello, creamos una variable “validacion6” y enviamos los parámetros de un auto cuyo año detallamos como “2025”. A continuación, el código y el resultado en consola:

*Código:*

```
////VALIDACIÓN 6: The car is 2 years older than the current year.
try
{
    FicAutoMobile validacion6 = new FicAutoMobile("Ford", "EcoSport", 2025);
}
catch (ArgumentException e)
{
    Console.WriteLine("Exception caught creating object with 2 years older than the current year.");
    Console.WriteLine(e.ToString());
}
```

*Ilustración 25: Paso 6, año mayor a dos años del actual, código.*

*Salida en consola:*

```
Salida
Mostrar salida de: Depurar
Exception caught creating object with 2 years older than the current year.
System.ArgumentException: The Year is out of range.
   at AutoMobile.Class.Auto.FicAutoMobile..ctor (System.String MakeD, System.Str
   \Auto\FicAutoMobile.cs:38
   at AutoMobile.Views.Auto.FicViAutosList..ctor () [0x00331] in C:\Users\Usuari
```

*Ilustración 26: Paso 6, año mayor a dos años del actual, consola.*

**G) Realice la codificación necesaria tanto en la clase como en la vista para que cada que se cree un auto (5 objetos del inciso [E]) este sea agregado a una lista de tipo clase “FicAutoMobile” con el nombre de “FicAllAutoMobileList”.**

The each namespace in XAML, includes concepts and elements that a loud us to identify them, like the x:Key that’s the way for us to get across different programs, the x:Class it’s the bridge between the view and the programming behind it, finally the x:Name give us the opportunity to modify the XAML after being processed.



Equipo “Los Híbridos”  
Desarrollo de Aplicaciones Híbridas.  
Unidad #3: Desarrollo de Interfaces.  
Docente: Francisco Ibarra Carlos  
ISC. Instituto Tecnológico de Tepic.

**Paso 1: lista para la clase.** En la clase, haremos la lista *FicAllAutoMobileList* de tipo clase *FicAutoMobile*.

```
/*G) Realice la codificación necesaria tanto en la clase como en la vista para que cada que se cree un auto  
(5 objetos del inciso [E]) este sea agregado a una lista de tipo clase "FicAutoMobile" con el nombre  
de "FicAllAutoMobileList". */  
  
List<FicAutoMobile> FicAllAutoMobileList = new List<FicAutoMobile>();
```

Ilustración 27: Paso 1, lista para la clase.

**Paso 2: objetos.** Para añadir los autos (objetos) creados en el inciso E, usamos el comando *“Add”* de la lista, pasando el parámetro (el auto).

```
FicAllAutoMobileList.Add(Auto1);  
FicAllAutoMobileList.Add(Auto2);  
FicAllAutoMobileList.Add(Auto3);  
FicAllAutoMobileList.Add(Auto4);  
FicAllAutoMobileList.Add(Auto5);
```

Ilustración 28: Paso 2, objetos.

**Paso 3: lista observable.** Cada que hagamos un cambio en la lista de la clase, esta debe verse reflejada en la vista. Para ello, necesitamos una variable de tipo *ObservableCollection* que usaremos en el siguiente inciso para reflejar los cambios.

```
//Lista local de tipo ObservableCollection<FicAutoMobile>  
var FicAutosCollection = new ObservableCollection<FicAutoMobile>();
```

Ilustración 29: Paso 3, lista observable.

**H) Diseña una vista con sus respectivos controles como son: Título (Label), buscador (SearchBar) y su grid (SfDataGrid) en donde despliegues o muestres la información de los 5 objetos (autos) que fueron agregados a la lista “FicAllAutoMobileList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.**

The get settings are the main way for us to retrieve information about the objects, this allowed us to bring the information to the XAML and print it in the SynFusion component, this avoids the use and reuse of unnecessary code.



**Paso 1: FicViAutosList.** Para crear los controles de la vista, deberemos ir al archivo *FicViAutosList.xaml*, dentro de la carpeta *Views -> Auto*.

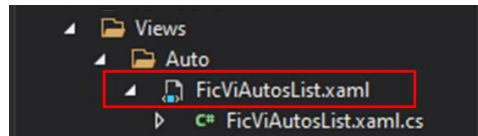


Ilustración 30: Paso 1, FicViAutosList.

**Paso 2: Grid.** Deberemos modificar tanto la creación como los atributos del Grid que usaremos para la pantalla. Para ello, se hace uso de los siguientes comandos, dentro de los cuales también se han incluido el número de columnas y de filas.

```
<ContentPage.Content>
<!--
  H) Diseña una vista con sus respectivos controles.
-->
<Grid
  Padding="5"
  ColumnSpacing="5"
  RowSpacing="5"
  BackgroundColor="Red"
  HorizontalOptions="FillAndExpand"
  VerticalOptions="FillAndExpand"
>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>

  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <Label x:Name="FicTitulo_Label"
    Grid.Row="0" Grid.Column="0"
    VerticalOptions="CenterAndExpand"
    HorizontalOptions="CenterAndExpand"
    Text="Auto Mobile List"
    FontSize="Medium"
    TextColor="White"
    FontAttributes="Bold"
  />
</Grid>
```

Ilustración 31: Paso 2, Grid.

**Paso 3: primera y segunda columnas.** En el primer renglón del Grid, debemos indicar el título que tendrá nuestra aplicación, en el segundo, incluiremos una barra de búsqueda. Ambos bloques de comandos se muestran a continuación:

```
<Label x:Name="FicTitulo_Label"
  Grid.Row="0" Grid.Column="0"
  VerticalOptions="CenterAndExpand"
  HorizontalOptions="CenterAndExpand"
  Text="Auto Mobile List"
  FontSize="Medium"
  TextColor="White"
  FontAttributes="Bold"
/>

<SearchBar x:Name="FicBuscador_SearchBar"
  Grid.Row="1" Grid.Column="0"
  Placeholder="Filtrar información..."
  HorizontalOptions="FillAndExpand"
  BackgroundColor="Green"
  TextColor="Black"
/>
```

Ilustración 32: Paso 3, primera y segunda columnas.

**Paso 4: DataGrid.** Para mostrar los datos que serán obtenidos desde la clase creada, debemos utilizar un componente DataGrid. Este se codifica de la siguiente manera. En esta ocasión, también se han incluido los detalles correspondientes al SyncFusion (del paquete NuGet *Syncfusion.Xamarin.SfDataGrid*):

```
<FicSfDataGrid:SfDataGrid
    ItemsSource="{Binding FicAutosList_ItemSource_SfDataGrid}"
    x:Name="FicAutosList_SfDataGrid"
    Grid.Row="2" Grid.Column="0"
    VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand"
    ColumnSizer="Star"
    AutoGenerateColumns="False"
    AllowSorting="False"
    BackgroundColor="#800080"
    SelectionMode="Single"
>
    <FicSfDataGrid:SfDataGrid.Columns x:TypeArguments="syncfusion:Columns">
        <FicSfDataGrid:GridTextColumn
            HeaderFontAttribute="Bold" HeaderText="* Make *"
            MappingName="Make" ColumnSizer="None"
            CellTextSize="10"
        >
    </FicSfDataGrid:GridTextColumn>
        <FicSfDataGrid:GridTextColumn
            HeaderFontAttribute="Bold" HeaderText="* Model *"
            MappingName="Model" ColumnSizer="None"
            CellTextSize="10"
        >
    </FicSfDataGrid:GridTextColumn>
        <FicSfDataGrid:GridTextColumn
            HeaderFontAttribute="Bold" HeaderText="* Year *"
            MappingName="Year" ColumnSizer="None"
            CellTextSize="10"
        >
    </FicSfDataGrid:GridTextColumn>
    </FicSfDataGrid:SfDataGrid.Columns>
</FicSfDataGrid:SfDataGrid>
```

Ilustración 33: Paso 4, DataGrid.

**Paso 5: herencia.** En FicViAutosList.cs, debemos indicarle a la clase que hará herencia de la clase INotifyPropertyChanged de la siguiente manera:

```
public partial class FicViAutosList : INotifyPropertyChanged
```

Ilustración 34: Paso 5, herencia.

**Paso 6: sobreescritura.** Al final de la clase FicViAutosList.cs, se sobrescribe, a través de los métodos de INotifyPropertyChanged, los cambios que se han realizado. Esto permite actualizar las propiedades de los controles.

```
INotifyPropertyChanged
```

Ilustración 35: Paso 6, sobreescritura.

**Paso 7: ForEach.** A través de un ciclo ForEach, alimentamos la lista observable de los datos pertenecientes a la lista de la clase (FicAutoMobileList)

```
//Uso de un ciclo para recorrer la lista de objetos.
foreach (FicAutoMobile FicItem in FicAllAutoMobileList)
{
    FicAutosCollection.Add(FicItem);
}
```

Ilustración 36: Paso 7, ForEach.



**Palabras Clave:** Inciso H, DataGrid, Herencia, Sobreescritura, ForEach, Ejecución.

**Actividad #2: 01/11/2021**

Página 18 de 22

**Paso 8: traspaso de datos.** Indicamos, para el DataGrid, que por cada elemento dentro de la lista de cuentas, debe de existir uno en la lista observable recientemente creada.

```
//Asignamos los registros de la coleccion local a la coleccion de enlace.  
FicAutosList_ItemSource_SfDataGrid = FicAutosCollection;
```

*Ilustración 37: Paso 8, traspaso de datos.*

**Paso 9: ejecución.** Finalmente, ejecutamos la aplicación para verificar que todo funciona correctamente.



The screenshot shows a mobile application interface with a red header bar containing the title 'Auto Mobile List'. Below the header is a green search bar with a magnifying glass icon and the text 'Filtrar información...'. Underneath the search bar is a table with three columns: '\* Make \*', '\* Model \*', and '\* Year \*'. The table contains five rows of data:

* Make *	* Model *	* Year *
Audi	A8	2020
Chevrolet	Aveo	2021
Jeep	Wrangler	2022
Bentley	Wraith	2022
Lexus	IS	2021

*Ilustración 38: Paso 9, ejecución.*

## Conclusión

En la segunda actividad de esta unidad continuamos con temas que anteriormente habíamos visto en la unidad 2, así de esta forma podemos reforzar lo aprendido además de comprender de una mejor forma lo adquirido y aplicado en la unidad 3. Con la actividad anterior se encontraron algunas dudas en el equipo las cuales fueron resueltas y para esta actividad resultado más fácil si realización ya que comprendíamos un poco más sobre el tema.

Con el reforzamiento de los conocimientos, resolver un problema con las diversas herramientas que nos ofrece la plataforma de código abierto Xamarin, se convierte en un proceso bastante práctico, aprendiendo a codificar aplicaciones que parecieran complejas en un lapso de tiempo muy corto.

Haciendo nuevamente uso de los `contentPage`, logramos crear la vista que se nos solicitaba, la cual heredaba sus datos de una clase diferente, con los objetos que hacían referencia a los autos que, en esta ocasión, contaban con una serie de validaciones más puntuales que en actividades anteriores, pero no por ello suponían un problema o reto mayor a los manejados y enfrentados previamente.

Este micro proyecto se pudo conseguir gracias a un trabajo colaborativo realizado por parte de todos los integrantes del equipo. Las notas proporcionadas por el profesor para la materia, fueron parte fundamental para el desarrollo de esta actividad pues, si bien incluyen el paso a paso de las acciones que debíamos poner en práctica la actividad pasada, varios de esos pasos fueron una guía crucial para lograr la finalidad de este proyecto.

Gracias a la documentación y la explicación previa a la actividad se pudo lograr el entendimiento individual de cada miembro del equipo, hubo pequeños obstáculos así que fue necesario también realizar unas pequeñas investigaciones para que se cumpliera al cien por ciento el objetivo, fuera de eso la actividad pudo realizarse de manera efectiva.

Nos quedó más clara la sintaxis de las etiquetas que debemos manejar, el cómo utilizar las listas y como se realizan para poder mandarlas a las vistas, con la realización de esta práctica se aprendió un poco más sobre Xamarin y cómo manejar sus vistas con listas enlazadas y el uso del paquete NuGet para SyncFusion.

## **Glosario**

### *C*

#### **Condicionales.**

son la serie de instrucciones o reglas que se necesitan para ejecutar un programa en cualquier lenguaje de programación. Las sentencias son las pautas para que un ejercicio a un programa pueda ser leído o interpretado por un ordenador ya que este interpreta los datos que el usuario ingresa.

#### **Consola.**

Una consola de programación es un dispositivo de comunicación con la centralita electrónica del vehículo, ya que no es posible acceder fácilmente, ni sirve cualquier dispositivo para extraer información que contiene la ECU, y aún menos modificarla.

#### **Constructor.**

Un constructor es un método cuyo nombre es igual que el nombre de su tipo. Su firma del método incluye solo el nombre del método y su lista de parámetros; no incluye un tipo de valor devuelto.

### *D*

#### **Directorio.**

En informática, un directorio es un contenedor virtual en el que se almacenan una agrupación de archivos informáticos y otros subdirectorios, atendiendo a su contenido, a su propósito o a cualquier criterio que decida el usuario.

### *F*

#### **Foreach.**

Foreach (de la palabra inglesa for each = para cada uno) es un bloque constructivo de los lenguajes de programación para recorrer los elementos de una colección. Foreach se utiliza por lo general en lugar de una norma para la declaración.

### *I*

#### **Interface.**

Se denomina interfaz al conjunto de elementos de la pantalla que permiten al usuario realizar acciones sobre el Sitio Web que está visitando.

## *L*

### **Label.**

La Label se utiliza para mostrar texto, tanto de una o varias líneas. Las etiquetas pueden tener decoraciones de texto, texto en color y usar fuentes personalizadas (familias, tamaños y opciones).

## *O*

### **Objeto.**

Se trata de un ente abstracto usado en programación que permite separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras.

## *V*

### **Validación.**

En las ciencias de la computación, validación de datos es el proceso de asegurar que un programa funcione en datos limpios, correctos y útiles. Las reglas pueden implementarse a través de las instalaciones automatizadas de un diccionario de datos, o mediante la inclusión de una lógica de validación explícita.

### **Vista.**

En una base de datos, una vista es el conjunto de resultados de una consulta almacenada en los datos. Es una consulta que se presenta como una tabla a partir de un conjunto de tablas en una base de datos relacional. Las vistas tienen la misma estructura que una tabla: filas y columnas.

## Referencias

¿Qué es una Interfaz? (s.f.). Obtenido de Guía Digital Beta:  
<https://www.guiadigital.gob.cl/articulo/que-es-una-interfaz.html>

Ibarra Carlos, F. (26 de Mayo de 2021). 31. T3. *Desarrollo de Vistas (Interfaces)*. Obtenido de Evernote: <https://www.evernote.com/shard/s94/client/snv?noteGuid=5423f264-4951-bfee-3615-3adbec54ac4a&noteKey=eb5d9d94de4c257ce0181fe8fbe7bfbfd&sn=https%3A%2F%2Fwww.evernote.com%2Fshard%2Fs94%2Fsh%2F5423f264-4951-bfee-3615-3adbec54ac4a%2Feb5d9d94de4c257ce0181fe8fbe7>

Ibarra Carlos, F. (20 de Mayo de 2021). 32. T3. *Desarrollo de ViewModel*. Obtenido de Evernote: <https://www.evernote.com/shard/s94/client/snv?noteGuid=8f83beb3-b698-2258-7121-7c0991f8fe49&noteKey=75d2659de56a35b56819bf76ac8623b5&sn=https%3A%2F%2Fwww.evernote.com%2Fshard%2Fs94%2Fsh%2F8f83beb3-b698-2258-7121-7c0991f8fe49%2F75d2659de56a35b56819bf76ac86>

Ibarra Carlos, F. (2021). 4. Tema 3. *Desarrollo de Interfaces*. Obtenido de Evernote: <https://www.evernote.com/shard/s94/client/snv?noteGuid=0dc3f9c2-ac81-4e37-bc19-55de23ca5c05&noteKey=56baafd7946f9c6698447e6a3eed55c4&sn=https%3A%2F%2Fwww.evernote.com%2Fshard%2Fs94%2Fsh%2F0dc3f9c2-ac81-4e37-bc19-55de23ca5c05%2F56baafd7946f9c6698447e6a3eed>

Microsoft. (28 de Mayo de 2020). ¿Qué es Xamarin? Obtenido de Microsoft Docs: <https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin>

Microsoft. (2021). *Vistas de colección en Xamarin.iOS*. Obtenido de Microsoft Docs: <https://docs.microsoft.com/es-es/xamarin/ios/user-interface/controls/uicollectionview>

Microsoft. (2021). *Xamarin.Forms Label*. Obtenido de Microsoft Docs: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/text/label>

Microsoft. (2021). *Xamarin.Forms SearchBar*. Obtenido de Microsoft Docs: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/searchbar>

Microsoft. (s.f.). *ContentPage Class*. Obtenido de Microsoft Docs: <https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.contentpage?view=xamarin-forms>