



TECNOLOGICO NACIONAL DE MÉXICO INSTITUTO TECNOLÓGICO DE TEPIC



Tepic Nayarit: 27 de Octubre 2021

UNIDAD 2: Desarrollo de interfaces.

ACTIVIDAD 1: MicroProyecto Bank Account.

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 el presente documento se mostrará una serie de pasos que constituirán una aplicación desarrollada en Visual Studio Code. Esta actividad se buscará poder realizar una aplicación donde se estarán utilizando la disposición de la pantalla utilizando los controladores vista que Xamarin nos aportan, utilizando funciones de la disposición Grid y dándole sus respectivos parámetros de posicionamiento en la pantalla.

Podemos saber que en el desarrollo de esta actividad se van integrando cada vez más las herramientas posibles que nos ofrece esta plataforma de desarrollo, esto con el objetivo de ir las integrando al conocimiento general para poder codificar de manera correcta la aplicación del proyecto final integral del semestre.

Mediante una serie de pasos que se estarán realizando para la actividad es como se obtendrá dichos conocimientos para la futura realización del mencionado proyecto, estos puntos estarán abordando todo el concepto del FrontEnd. Cabe resaltar que existen varios controles que nos ayudan a ordenar los demás controles de la vista que deseamos mostrar en ejecución, esto para que no se muestren de cierta manera juntos o amontonados o de igual manera desalineados o en su peor de los casos en un desorden total.

Como buenos desarrolladores tenemos que tener en cuenta que la vista de nuestras aplicaciones tiene que estar en óptimas condiciones, para que el usuario que este destinado a manejar la aplicación no se le complique o se le dificulte la correcta utilización, en Xamarin Forms los controles más utilizados en la ayuda para la ordenación de nuestros objetos en pantalla son StackLayout y Grid.

Un StackLayout organiza las vistas secundarias en una pila unidimensional, ya sea horizontal o verticalmente. De forma predeterminada, está orientado verticalmente. Además, se puede utilizar como un diseño primario que contiene otros diseños secundarios. Grid, por otro lado, es un diseño que organiza sus hijos en filas y columnas, que pueden tener tamaños proporcionales o absolutos. De forma predeterminada, contiene una fila y una columna.

Una vez tomando en cuenta todo lo anteriormente dicho podemos comenzar a desarrollar en el apartado de FrontEnd la dicha actividad. Añadiendo al trabajo realizado en la práctica, se suman diversos collages de imágenes que representan las ideas principales de lo que se presenta, también se anexan palabras clave en los encabezados. Otra de las características que contiene este documento es un glosario donde se muestran términos considerados relevantes en esta actividad.

Finalmente se presentarán algunas referencias que fueron consultadas a lo largo de este proceso para ofrecer un acceso sencillo a la información. Esperando que el documento sirva de lectura de referencia al lector.

Contenido

Introducción.....	1
Contenido	3
A) Crear la vista “FicViBankAccountList.xalm” de tipo “Content Page” y su respectivo constructor.....	4
B) Realice la codificación necesaria tanto en la clase como en la vista para que cada que se cree una cuenta bancaria (al menos 6 objetos) estas sea agregadas a una lista de tipo clase “FicBankAccount” con el nombre de “FicAllBankAccountList”.....	6
C) Diseña una vista con sus respectivos controles como son: Titulo (Label), buscador (SearchBar) y su grid (SfDataGrid) en donde despliegues o muestres la información de las cuentas bancarias que fueron agregados a la lista “FicAllBankAccountList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.....	8
D) 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 las transacciones realizadas sobre una cuenta bancaria (objeto de tipo “FicBankAccount”) las cuales fueron agregados a la lista “FicAllTransactionList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.....	13
Conclusión.....	19
Glosario	20
Referencias	22

Contenido

3.4 Listas.

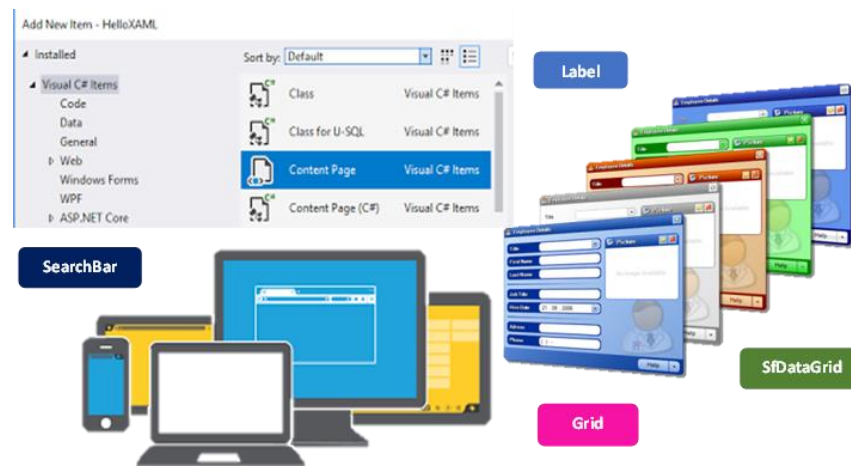
3.5 Navegación.

3.6 Modales.

Actividad 1. MicroProyecto Bank Account.

- A) Crear la vista “FicViBankAccountList.xaml” de tipo “Content Page” y su respectivo constructor.
- B) Realice la codificación necesaria tanto en la clase como en la vista para que cada que se cree una cuenta bancaria (al menos 6 objetos) estas sean agregadas a una lista de tipo clase “FicBankAccount” con el nombre de “FicAllBankAccountList”.
- C) Diseña una vista con sus respectivos controles como son: Titulo (Label), buscador (SearchBar) y su grid (SfDataGrid) en donde despliegues o muestres la información de las cuentas bancarias que fueron agregados a la lista “FicAllBankAccountList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.
- D) 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 las transacciones realizadas sobre una cuenta bancaria (objeto de tipo “FicBankAccount”) las cuales fueron agregados a la lista “FicAllTransactionList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.

The use of Xaml comes greatly for friendly design on mobile apps, with this in mind and the whole library set from Xamarin gives free roam to new ways to program and design. In this scenario, the Oriented program paradigm it's a great way for us to plan and build our ideas and get them to good use. Also C# comes hand and is very easy to use even for the programmers related to Java.



A) Crear la vista “FicViBankAccountList.xaml” de tipo “Content Page” y su respectivo constructor.

Paso 1: Proyecto Anterior. El primer paso es abrir el proyecto anterior, correspondiente a la Actividad 2, de la unidad 2. Esto puede hacerse directamente con un clic al proyecto en el menú de “Recientes” que se abre al iniciar la aplicación.

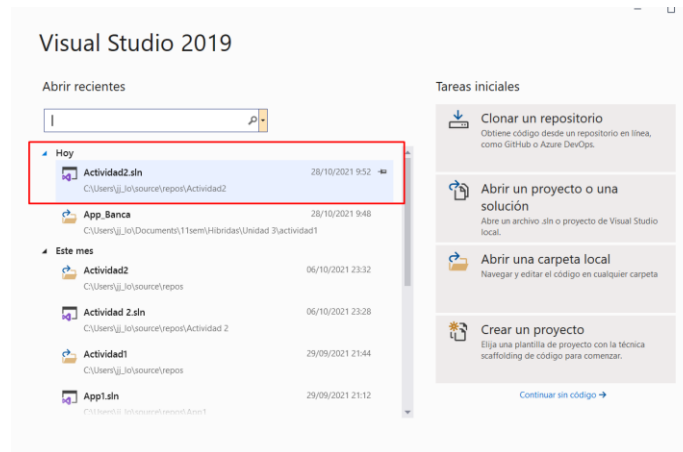


Ilustración 1: Paso 1, abrir la actividad anterior.

Paso 2: carpetas “Views” y “Bank”. Verifica que dentro de la actividad exista la carpeta Views y, dentro de esta, la carpeta Bank. Recuerda que estas fueron realizadas con anterioridad en la actividad anterior.

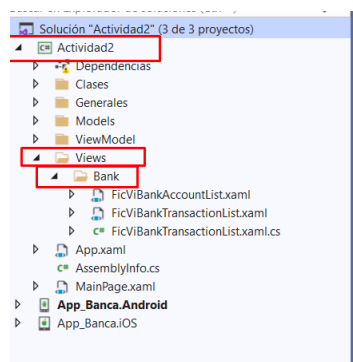


Ilustración 2: Paso 2, verificación de carpetas.

Paso 2: “FicViBankAccountList”. Dentro de la carpeta Bank, es necesario tener una página de contenido con el nombre FicViBankAccountList.xaml. Esta *también* fue creada en la actividad anterior, solamente hay que verificar su existencia, pues es allí desde donde trabajaremos.

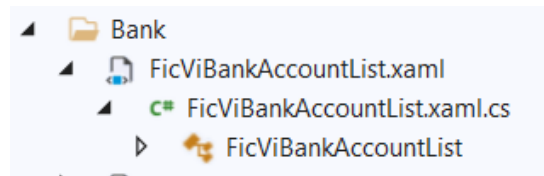


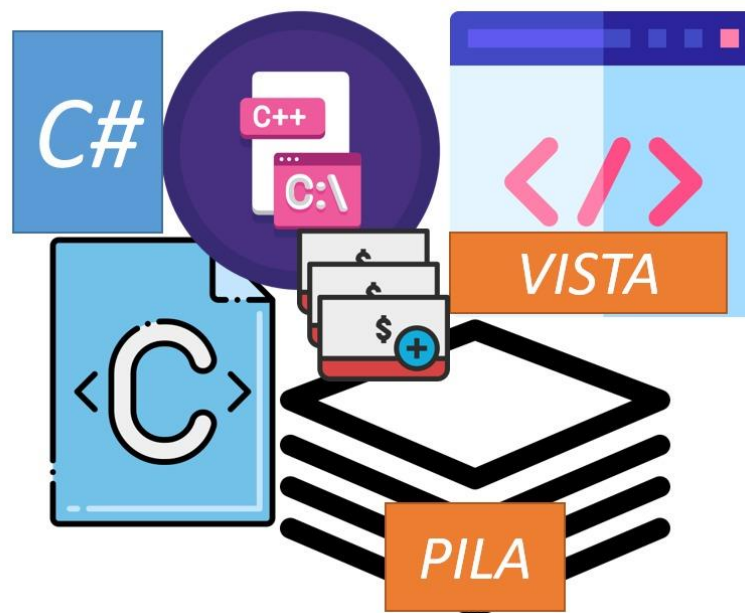
Ilustración 3: Paso 3, FicViBankAccountList

Paso 3: constructor. Finalmente, dentro del archivo *FicViBankAccountList.xaml.cs*, es necesario crear el constructor de la siguiente manera:

```
4 {  
5  
6 4 referencias  
7 public partial class FicViBankAccountList : ContentPage  
8 {  
9 0 referencias  
10 public FicViBankAccountList()  
11 {  
12     InitializeComponent();  
13 }  
14 }
```

Ilustración 4: Paso 3, constructor

The model, view and controller it's very useful in this case scenario, this is more visible within the interface with the data displayed and in use, the only keep on mind consideration is about the links between these 3 parts.



B) Realice la codificación necesaria tanto en la clase como en la vista para que cada que se cree una cuenta bancaria (al menos 6 objetos) estas sean agregadas a una lista de tipo clase “FicBankAccount” con el nombre de “FicAllBankAccountList”.

Paso 1: carpeta ViewModel y FicVmBankAccountList. Desplegando el explorador de soluciones de la Actividad, deberemos crear una nueva carpeta, llamada *ViewModels*. En esta, deberemos añadir una clase llamada *FicVmBankAccountList.cs*, en la cual realizaremos la codificación necesaria.

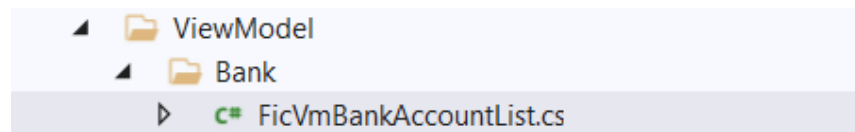


Ilustración 5: Paso 1, resultado de la creación de la carpeta y el archivo.

Paso 2: constructor de la clase. Dentro de la clase, crearemos un constructor como en el **paso 3, inciso A:**

```
//Inciso B Paso 2- Creamos el constructor de la clase.
0 referencias
public FicVmBankAccountList()
{
}
```

Ilustración 6: Paso 2, constructor.

Paso 3: FicMetGetBankAccountList. Es necesario crear un método para posteriormente añadir los objetos a nuestra aplicación. Es un método de tipo void, ya que no retornará ningún valor, y quedará de la siguiente manera:

```
//Inciso B Paso 3- Crearemos el metodo FicMetGetBankAccountList
1 referencia
public void FicMetGetBankAccountList()
{
}
```

Ilustración 7: Paso 3: Método Void

Este método, lo instanciaremos en el constructor de la clase de la siguiente manera:

```
//Inciso B Paso 4- Creamos el constructor de la clase.
0 referencias
public FicVmBankAccountList()
{
    //Inciso B Paso 4- Creado el metodo lo instanciamos en el constructor.
    FicMetGetBankAccountList();
}
```

Ilustración 8: Paso 3: instanciar el método en el constructor.

Paso 4: lista de cuentas bancarias. Para poder visualizar las cuentas bancarias existentes en la aplicación, deberemos crear una variable de tipo Lista para almacenar estos datos.

Equipo “Los Híbridos”

Desarrollo de Aplicaciones Híbridas.

Unidad #3: Desarrollo de Interfaces.

Docente: Francisco Ibarra Carlos

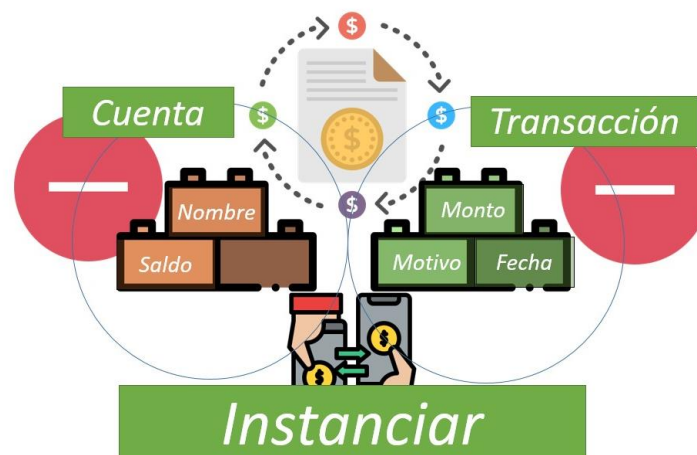
ISC. Instituto Tecnológico de Tepic.

```
//Inciso B Paso 4- Crearemos la lista donde almacenaremos la cuentas bancarias.  
List<FicBankAccount> FicAllBankAccountList = new List<FicBankAccount>();  
..
```

Paso 5: Instancias de objetos. Para tener los objetos a mostrar listos, debemos instanciarlos, así como añadirlos a la lista. Esto se logra a través de los siguientes comandos:

```
using FicBankAccount = FicAllBankAccountList = new List<FicBankAccount>();  
//-----  
//Inciso A Paso 5- Instanciamos los objetos y los agregamos a la lista.  
//FicBankAccount FicAccountObj = new FicBankAccount();  
var FicAccount = new FicBankAccount("Carlos Alberto Becerra Enriquez", 1000);  
FicAllBankAccountList.Add(FicAccount);  
  
//FicBankAccount FicAccountObj = new FicBankAccount();  
var FicAccount2 = new FicBankAccount("Ximena Verdín Carreño", 2000);  
FicAllBankAccountList.Add(FicAccount2);  
  
//FicBankAccount FicAccountObj = new FicBankAccount();  
var FicAccount3 = new FicBankAccount("Brenda Karime Burgara Ortega", 3000);  
FicAllBankAccountList.Add(FicAccount3);  
  
//FicBankAccount FicAccountObj = new FicBankAccount();  
var FicAccount4 = new FicBankAccount("Lilian Elizabeth González Guzmán", 4000);  
FicAllBankAccountList.Add(FicAccount4);  
  
//FicBankAccount FicAccountObj = new FicBankAccount();  
var FicAccount5 = new FicBankAccount("Juan Eduardo Tovar Andrade", 5000);  
FicAllBankAccountList.Add(FicAccount5);  
  
//FicBankAccount FicAccountObj = new FicBankAccount();  
var FicAccount6 = new FicBankAccount("Jorge Jorge Varela García", 6000);  
FicAllBankAccountList.Add(FicAccount6);
```

The object account, needs to be created with 2 things, the name of the owner and, the amount of initial money within the count. When this happens it's automatically set the seed from the account, this is what is called instance an object.



C) Diseña una vista con sus respectivos controles como son: Titulo (Label), buscador (SearchBar) y su grid (SfDataGrid) en donde despliegues o muestres la información de las cuentas bancarias que fueron agregados a la lista “FicAllBankAccountList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.

Paso 0: FicViBankAccountList.xaml. En la carpeta de Views, tendremos el archivo FicViBankAccountList.xaml, el cual modificaremos para darle la apariencia solicitada a la pantalla que veremos en la aplicación.

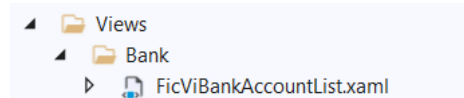


Ilustración 9: Paso 0, FicViBankAccountList.xaml.

Paso 1: Grid. Debemos modificar tanto la creación como los atributos del Grid que usaremos para la pantalla. Para ello, se hace uso de los siguientes comandos:

```
<!--Inciso C paso 1: Indicaremos la creación del Grid-->
<Grid
    Padding="5"
    ColumnSpacing="5"
    RowSpacing="5"
    BackgroundColor="#blue"
    HorizontalOptions="FillAndExpand"
    VerticalOptions="FillAndExpand"
/>
```

Ilustración 10: Paso 1, Grid

Paso 2: Numero de columnas. Dado que el Grid necesita los parámetros de columnas y filas (o renglones) primero definimos las columnas. Para el caso de la actividad, solo es 1.

```
<!--Inciso C paso 2 Indicamos el número de columnas, en este caso 1-->
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
```

Ilustración 11: Paso 2, columnas

Paso 3: Numero de renglones. De igual manera, indicamos el número de renglones del Grid, en este caso, 3.

```
<!--Inciso C paso 3: Indicamos el número de renglones, en este caso 3-->
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
```

Ilustración 12: Paso 3, renglones

Paso 4: título. En el primer renglón del Grid, debemos indicar el título que tendrá nuestra aplicación, así como una serie de parámetros tales como el nombre de la etiqueta, la posición, el acomodo dentro de la pantalla, el color del texto, entre otros datos.

```
<!--Inciso C paso 4: Indicamos la linea de titulo en el primer renglon.-->
<Label x:Name="FicTitulo_Label"
Grid.Row="0" Grid.Column="0"
VerticalOptions="CenterAndExpand"
HorizontalOptions="CenterAndExpand"
Text="Bank Account List"
FontSize="Medium"
TextColor="Yellow"
FontAttributes="Bold"
/>
```

Ilustración 13: Paso 4, titulo

Paso 5: barra de búsqueda. En el segundo renglón del Grid, definiremos una barra de búsqueda para la aplicación. A este objeto también debemos especificarle una serie de atributos, los cuales se listan en el siguiente código:

```
<!--Inciso C paso 5: Indicamos la creación de la barra de busqueda en el segundo renglon.-->
<SearchBar x:Name="FicBuscador_SearchBar"
Grid.Row="1" Grid.Column="0"
Placeholder="Filtrar información..."
HorizontalOptions="FillAndExpand"
BackgroundColor="Orange"
TextColor="Blue"
/>
```

Ilustración 14: Paso 5, barra de búsqueda

Paso 6: DataGrid. Para mostrar los datos que serán obtenidos desde la clase creada en el inciso B, debemos utilizar un componente DataGrid. Este se codifica de la siguiente manera:

```
<!--Inciso C paso 6: Creamos el componente DataGrid para mostrar los datos en el tercer renglon-->
<FicSfDataGrid:SfDataGrid
x:Name="FicBankAccountList_SfDataGrid"
Grid.Row="2" Grid.Column="0"
VerticalOptions="FillAndExpand"
HorizontalOptions="FillAndExpand"
ColumnSizer="Star"
AutoGenerateColumns="False"
AllowSorting="False"
BackgroundColor="GreenYellow"
SelectionMode="Single"
ItemsSource="{Binding FicBankAccountList_ItemSource_SfDataGrid}"
/>
```

Ilustración 15: Paso 6, DataGrid

Xaml work with a typed oriented design, this comes useful when all properties may be used repeatedly and work with modularity. Also many new components may be added within new libraries through the management of NuGet packages.



Paso 7: NuGet. En los paquetes de NuGet, es necesario buscar e instalar *Syncfusion.Xamarin.SfDataGrid* para utilizar el objeto anteriormente creado, así como otras características importantes que ofrece esta paquetería.

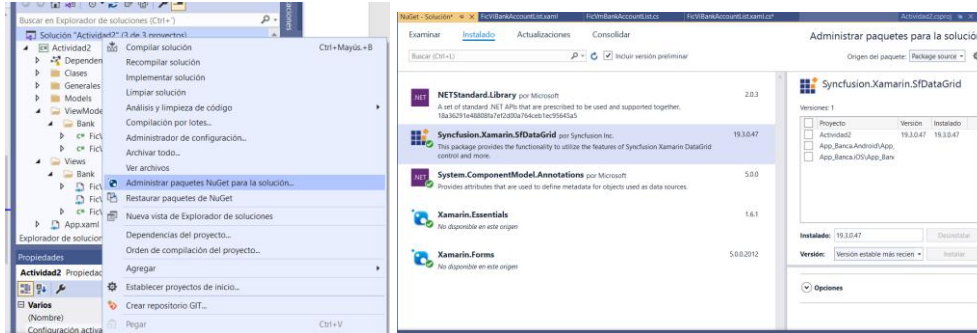


Ilustración 16: Paso 7, NuGet

Posteriormente, deberemos indicarle a la aplicación que hará uso de la paquetería recientemente instalada:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:SfDataGrid="clr-namespace:Syncfusion.SfDataGrid.XForms;assembly=Syncfusion.SfDataGrid.XForms"
             xmlns:ViewModel="clr-namespace:App_Banca.ViewModel.Bank"
             x:DataType="ViewModel:FicVmBankTransactionList"
             x:Class="App_Banca.Views.Bank.FicViBankTransactionList">
```

Paso 8: SyncFusion. Deberemos indicarle al objeto SyncFusion el cómo se distribuirán las columnas dentro del mismo. Para ello, usaremos los siguientes comandos:

```
<!--Inciso C paso 8: Indicaremos el como se repartiran las columnas dentro del Objeto SyncFusion-->
<FicSfDataGrid:SfDataGrid.Columns x:TypeArguments="syncfusion:Columns">
    <FicSfDataGrid:GridTextColumn
        HeaderFontAttribute="Bold" HeaderText="#Account"
        MappingName="Number" ColumnSize="SizeToHeader"
        CellTextSize="10"
    >
</FicSfDataGrid:GridTextColumn>
    <FicSfDataGrid:GridTextColumn
        HeaderFontAttribute="Bold" HeaderText="#Owner"
        MappingName="Owner" ColumnSize="LastColumnFill"
        CellTextSize="10"
    >
</FicSfDataGrid:GridTextColumn>
    <FicSfDataGrid:GridTextColumn
        HeaderFontAttribute="Bold" HeaderText="$ Balance"
        MappingName="Balance" ColumnSize="SizeToHeader"
        CellTextSize="10"
    >
</FicSfDataGrid:GridTextColumn>
</FicSfDataGrid:SfDataGrid.Columns>
</FicSfDataGrid:SfDataGrid>
</Grid>
```

Ilustración 17: Paso 8, SyncFusion

Paso 9: fuente de información. Finalmente, le indicamos al DataGrid desde dónde tomará la información para ser llenado:

```
<!---Inciso C paso 6: Creamos el componente DataGrid para mostrar los datos en el tercer renglon-->
<FicSfDataGrid:SfDataGrid
    x:Name="FicBankAccountList_SfDataGrid"
    Grid.Row="2" Grid.Column="0"
    VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand"
    ColumnSizer="Star"
    AutoGenerateColumns="False"
    AllowSorting="False"
    BackgroundColor="#GreenYellow"
    SelectionMode="Single"
    ItemsSource="{Binding FicBankAccountList_ItemSource_SfDataGrid}"
/>
<!---Inciso C paso 9:Le indicaremos el onjeto que rellenara el componente-->
```

Ilustración 18: Paso 9, fuente de información

Paso 10: herencia. En FicVmBankAccountList.cs, debemos indicarle a la clase que hará herencia de la clase INotifyPropertyChanged de la siguiente manera:

```
public class FicVmBankAccountList : INotifyPropertyChanged
//Inciso C Paso 10- Indicamo que la clase FicVmBankAccountList  hara herencia de la clase INotifyPropertyChanged
{
```

Ilustración 19. Paso 10, herencia.

Paso 11: sobrescritura. Al final de la clase `FicVmBankAccountList`, se sobrescribe, a través de los métodos de `INotifyPropertyChanged`, los cambios que se han realizado. Esto permite actualizar las propiedades de los controles.

```
//Inciso C Paso 11: Sobreescribimos los metodos propios de INotifyPropertyChanged para evitar conflicto e indicar que realizara.
#region INotifyPropertyChanged
```

Ilustración 20: Paso 11, sobreescritura

Paso 12: lista observable. Para poder pasar los datos entre a clase y el xaml, debemos crear un objeto de tipo ObservableCollection.

```
//Inciso C Paso 12: Creamos la lista observable que nos permitira pasar datos a la siguiente lista en el xml.
var FicBankAccountCollection = new ObservableCollection<FicBankAccount>();
```

Ilustración 21: Paso 12, lista observable.

Paso 13: ForEach. A través de un ciclo ForEach, alimentamos la lista observable de los datos pertenecientes a la lista de la clase (FicAllBankAccountList)

```
//Inciso C Paso 13: Le indicamos que para cada elemento dentro de la lista de cuenta se pasaran a esta lista observable.
foreach (FicBankAccount FicItem in FicAllBankAccountList)
{
    FicBankAccountCollection.Add(FicItem);
}
```

Ilustración 22: Paso 13, ForEach

Paso 14: 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.

```
//Inciso D Paso 14: Le indicamos que para cada elemento dentro de la lista de cuenta se pasaran a esta lista observable.  
FicBankTransactionList_ItemSource_SfDataGrid = FicBankTransactionCollection;
```

Ilustración 23: Paso 14, traspaso de datos

Paso 15: ejecución. Finalmente, ejecutamos la aplicación para verificar que todo funciona correctamente.



#Account	-Owner-	\$ Balance
555666001	Carlos Alberto Becerra Enriquez	1000
555666002	Ximena Verdin Carreño	2000
555666003	Brenda Karime Burgara Ortega	3000
555666004	Lilien Elizabeth González Guzmán	4000
555666005	Juan Eduardo Tovar Andrade	5000
555666006	Jorge Jorge Varela Garcia	6000

Ilustración 24: Paso 15, ejecución

When we talk about objects is simple to get it into imagination, but when it comes to the creation, we need to give the instance about it. The constructor exists of this reason also the methods like “get” and “set” properly used. So, if we used the constructor properties correctly, the results are as amazing as the showed interface.



D) 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 las transacciones realizadas sobre una cuenta bancaria (objeto de tipo “FicBankAccount”) las cuales fueron agregados a la lista “FicAllTransactionList” esto a través de un “foreach” para el llenado de la colección y asignación a la fuente de datos y enlazador.

Paso 1: FicVmBankTransactionList. En la carpeta *ViewModels* deberemos añadir una clase llamada *FicVmBankTransactionList.cs*, en la cual realizaremos la codificación necesaria.

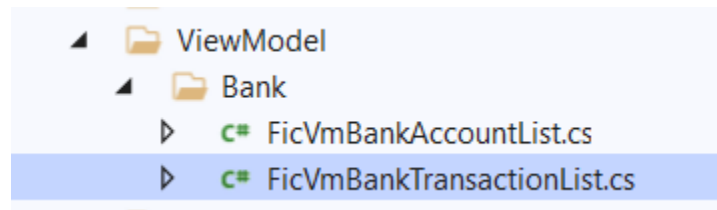


Ilustración 25: Paso 1, resultado de la creación de la carpeta y el archivo.

Paso 2: constructor de la clase. Dentro de la clase, crearemos un constructor para la clase.

```
//Inciso B Paso 2- Creamos el constructor de la clase.  
0 referencias  
public FicVmBankTransactionList()  
{  
    ...  
}
```

Ilustración 26: Paso 2, constructor.

Paso 3: FicMetGetBankTransactionList. Es necesario crear un método para posteriormente añadir los objetos a nuestra aplicación. Es un método de tipo void, ya que no retornará ningún valor, y quedará de la siguiente manera:

```
1 referencia  
public void FicMetGetBankTransactionList()  
{  
    ...  
}
```

Ilustración 27: Paso 3: Método Void

Este método, lo instanciaremos en el constructor de la clase de la siguiente manera:


```
0 referencias
public FicVmBankTransactionList()
{
    FicMetGetBankTransactionList();
}
```

Ilustración 28: Paso 3: instanciar el método en el constructor.

Paso 4: lista de transacciones bancarias. Para poder visualizar las transacciones bancarias existentes en la aplicación, deberemos crear una variable de tipo Lista para almacenar estos datos.

```
//Inciso D Paso 4- Crearemos la lista donde almacenaremos las transacciones bancarias.
List<FicTransaction> FicAllBankTransactionList = new List<FicTransaction>();
//
```

Paso 5: Instancias de objetos. Para tener los objetos a mostrar listos, debemos instanciarlos, así como añadirlos a la lista. Esto se logra a través de los siguientes comandos:

```
//
//Inciso D Paso 5- Instanciamos los objetos y los agregamos a la lista.

var FicTransaction = new FicTransaction(4000, DateTime.Now, "Initial Balance");
FicAllBankTransactionList.Add(FicTransaction);

var FicTransaction2 = new FicTransaction(-580, DateTime.Now, "Food Payment");
FicAllBankTransactionList.Add(FicTransaction2);

var FicTransaction3 = new FicTransaction(-1900, DateTime.Now, "Rent Payment");
FicAllBankTransactionList.Add(FicTransaction3);

var FicTransaction4 = new FicTransaction(200, DateTime.Now, "Amazon Return");
FicAllBankTransactionList.Add(FicTransaction4);

//
```

Paso 6: FicViBankTransactionList.xaml. En la carpeta de Views, tendremos el archivo FicViBankTransactionList.xaml, el cual modificaremos para darle la apariencia solicitada a la pantalla que veremos en la aplicación.

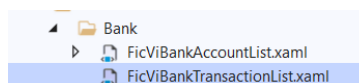


Ilustración 29: FicViBankTransactionList.xaml.

The Grid is a layout that organizes its children into rows and columns, which can have proportional or absolute sizes. By default, a Grid contains one row and one column. In the other hand, a StackLayout organizes child views in a one-dimensional stack, either horizontally or vertically. By default, a StackLayout is oriented vertically. In addition, a StackLayout can be used as a parent layout that contains other child layouts.



Paso 7: 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, pues es similar a lo explicado en el inciso C.

```
<!--Inciso D paso 1: Indicaremos la creación del Grid-->
<Grid
  Padding="5"
  ColumnSpacing="5"
  RowSpacing="5"
  BackgroundColor="#blue"
  HorizontalOptions="FillAndExpand"
  VerticalOptions="FillAndExpand"
>
  <!--Inciso D paso 2 Indicamos el número de columnas, en este caso 1-->
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <!--Inciso D paso 3: Indicamos el número de renglones, en este caso 3-->
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  </Grid>
```

Ilustración 30: Paso 7, Grid

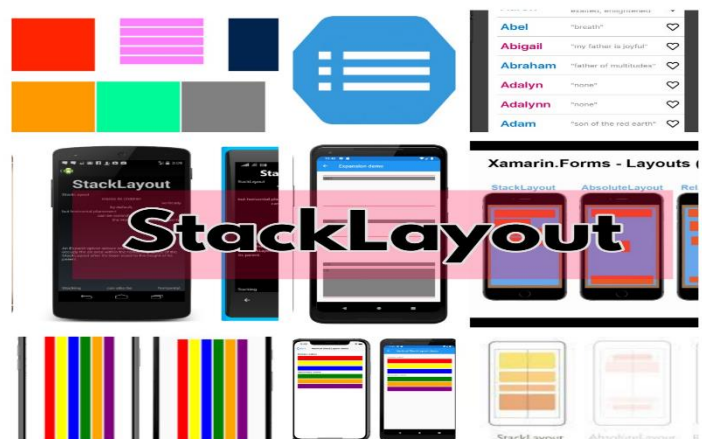
Paso 8: primera y segunda columnas. En el primer renglón del Grid, debemos indicar el título que tendrá nuestra aplicación, en el segundo, incluiremos una barra de búsqueda. Ambos bloques de comandos son similares a los presentados en el inciso C.

```
<!--Inciso D paso 4: Indicamos la línea de título en el primer renglon.-->
<Label x:Name="FicTitulo_Label"
  Grid.Row="0" Grid.Column="0"
  VerticalOptions="CenterAndExpand"
  HorizontalOptions="CenterAndExpand"
  Text="Bank Transactions List"
  FontSize="Medium"
  TextColor="#Yellow"
  FontAttributes="Bold"
/>

<!--Inciso D paso 5: Indicamos la creación de la barra de búsqueda en el segundo renglon.-->
<SearchBar x:Name="FicBuscador_SearchBar"
  Grid.Row="1" Grid.Column="0"
  Placeholder="Filtrar información..."
  HorizontalOptions="FillAndExpand"
  BackgroundColor="#Orange"
  TextColor="#Blue"
/>
```

Ilustración 31: Paso 8, primera y segunda columnas

The transaction log, relates with an object from the account class, but are different objects. For us to create a log about a transaction first needs to exist an account for it to be related. This a very important relation to consider about when we instance this object. If this instance was succesfull, the GridLayout will be filled in the correct way, if it isn't, then something happened in the middle of the instance.



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

Paso 9: 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 :

```
<!--Inciso D paso 6: Creamos el componente DataGrid para mostrar los datos en el tercer renglon-->
<FicSfDataGrid:SfDataGrid
    x:Name="FicBankTransactionList_SfDataGrid"
    Grid.Row="2" Grid.Column="0"
    VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand"
    ColumnSizer="Star"
    AutoGenerateColumns="False"
    AllowSorting="False"
    BackgroundColor="#90EE90"
    SelectionMode="Single"
    ItemsSource="{Binding FicBankTransactionList_ItemSource_SfDataGrid}"
>
<!--Inciso D paso 9:Le indicaremos el objeto que rellenara el componente-->

<!--Inciso D paso 7: Agregamos el paquete SyncFusion-->
<!--Inciso D paso 8: Indicaremos el como se repartiran las columnas dentro del Objeto SyncFusion-->
<FicSfDataGrid:SfDataGrid.Columns x:TypeArguments="syncfusion:Columns">
    <FicSfDataGrid:GridTextColumn
        HeaderFontAttribute="Bold" HeaderText="--Monto--"
        MappingName="Amount" ColumnSizer="SizeToHeader"
        CellTextSize="10"
    >
</FicSfDataGrid:GridTextColumn >
    <FicSfDataGrid:GridTextColumn
        HeaderFontAttribute="Bold" HeaderText="--Fecha--"
        MappingName="Date" ColumnSizer="LastColumnFill"
        CellTextSize="10"
    >
</FicSfDataGrid:GridTextColumn >
    <FicSfDataGrid:GridTextColumn
        HeaderFontAttribute="Bold" HeaderText="--Notas--"
        MappingName="Notes" ColumnSizer="SizeToHeader"
        CellTextSize="10"
    >
</FicSfDataGrid:GridTextColumn >
</FicSfDataGrid:SfDataGrid.Columns>
</FicSfDataGrid:SfDataGrid>
```

Ilustración 32: Paso 9, DataGrid

Paso 10: herencia. En FicVmBancTransactionList.cs, debemos indicarle a la clase que hará herencia de la clase INotifyPropertyChanged de la siguiente manera:

```
public class FicVmBankAccountList : INotifyPropertyChanged
{
    //Inciso C Paso 10- Indicamos que la clase FicVmBankAccountList hara herencia de la clase INotifyPropertyChanged
}
```

Ilustración 33. Paso 10, herencia.

Paso 11: sobreescritura. Al final de la clase FicVmBankTransactionList, se sobrescribe, a través de los métodos de INotifyPropertyChanged, los cambios que se han realizado. Esto permite actualizar las propiedades de los controles.

```
//Inciso D Paso 11: Sobreescibimos los metodos propios de INotifyPropertyChanged para evitar conflicto e indicar que realizara.
INotifyPropertyChanged
}
```

Ilustración 34: Paso 11, sobreescritura



The movement between objects across the list it's very useful when is necessary to show, cause it's already a data structure with their own methods like the for each, to show the whole set of elements.

Paso 12: lista observable. Para poder pasar los datos entre a clase y el xaml, debemos crear un objeto de tipo ObservableCollection.

```
//Inciso D Paso 12: Creamos la lista observable que nos permitira pasar datos a la siguiente lista en el xml.  
var FicBankTransactionCollection = new ObservableCollection<FicTransaction>();
```

Ilustración 35: Paso 12, lista observable.

Paso 13: ForEach. A través de un ciclo ForEach, alimentamos la lista observable de los datos pertenecientes a la lista de la clase (FicAllBankTransactionList)

```
//Inciso D Paso 13: Le indicamos que para cada elemento dentro de la lista de cuenta se pasaran a esta lista observable.  
foreach (FicTransaction FicItem in FicAllBankTransactionList)  
{  
    FicBankTransactionCollection.Add(FicItem);  
}
```

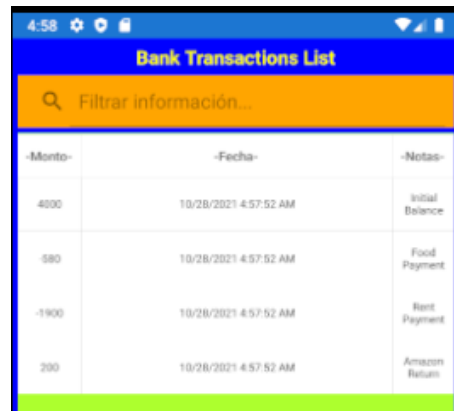
Ilustración 36: Paso 13, ForEach

Paso 14: 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.

```
//Inciso D Paso 14: Le indicamos que para cada elemento dentro de la lista de cuenta se pasaran a esta lista observable.  
FicBankTransactionList_ItemSource_SfDataGrid = FicBankTransactionCollection;
```

Ilustración 37: Paso 14, traspaso de datos

Paso 15: ejecución. Finalmente, ejecutamos la aplicación para verificar que todo funciona correctamente.



-Monto-	-Fecha-	-Notas-
4000	10/28/2021 4:57:52 AM	Initial Balance
-580	10/28/2021 4:57:52 AM	Food Payment
-1900	10/28/2021 4:57:52 AM	Rent Payment
200	10/28/2021 4:57:52 AM	Amazon Return

Ilustración 38: Paso 15, ejecución

Conclusión

Resolver un problema con la cooperación de las diversas herramientas que nos ofrece la plataforma de código abierto Xamarin, se convierte en un proceso bastante práctico, ya que contiene funcionalidades con las que podemos codificar páginas complejas en un lapso de tiempo muy corto.

Trabajar con Xamarin es una gran alternativa a los desarrollos nativos puros en la mayoría de escenarios habituales: optimizamos costes y tiempos de entrega en el desarrollo multiplataforma obteniendo un resultado 100% nativo.

En esta ocasión la actividad la llevamos a cabo por medio de Xamarin de la mano de la herramienta de la plataforma una página llamada “contentPage”, esta es el tipo de página más simple y común, que cuenta con una única vista donde pudimos añadir contenido para desarrollar esta actividad. Logramos crear distintas vistas que contienen diferentes clases, objetos, referentes a la información sobre cuentas bancarias que fueron agregadas a la lista creada en la clase principal.

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 incluyen el paso a paso de las acciones que debíamos poner en práctica 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.

Glosario

C

Colección.

Las vistas de colección permiten mostrar contenido mediante diseños arbitrarios. Permiten crear fácilmente diseños de tipo cuadrícula de forma rápida, al tiempo que admiten diseños personalizados.

Componente.

Representa un fragmento en lugar de una respuesta completa. Incluye las mismas ventajas de separación de conceptos y capacidad de prueba que se encuentran entre un controlador y una vista. Puede tener parámetros y lógica de negocios.

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.

Content Page.

Esta es una página que muestra una sola vista, a menudo un contenedor como StackLayout o ScrollView .

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).

Lista.

En informática, una lista o secuencia es un tipo de datos abstractos que representa un número contable de valores ordenados, donde el mismo valor puede aparecer más de una vez.

S

SearchBar.

SearchBar es un control de entrada de usuario que se usa para iniciar una búsqueda. El SearchBarcontrol admite texto de marcador de posición, entrada de consultas, ejecución de búsqueda y cancelación.

V

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>

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>