Desenvolupament d'interfícies

Unitat 4. Més components visuals. Organització de l'aplicació

Índex

1. Introducció

En aquest punt continuarem estudiant els controls WPF més comuns i començarem a utilitzar altres que ens seran molt útils per a posar un poc d'ordre i trellat a aplicacions amb moltes opcions i més complexitat que s'han de resoldre amb una única finestra.

És a dir, contemplarem l'ús de menús, pestanyes, etc...

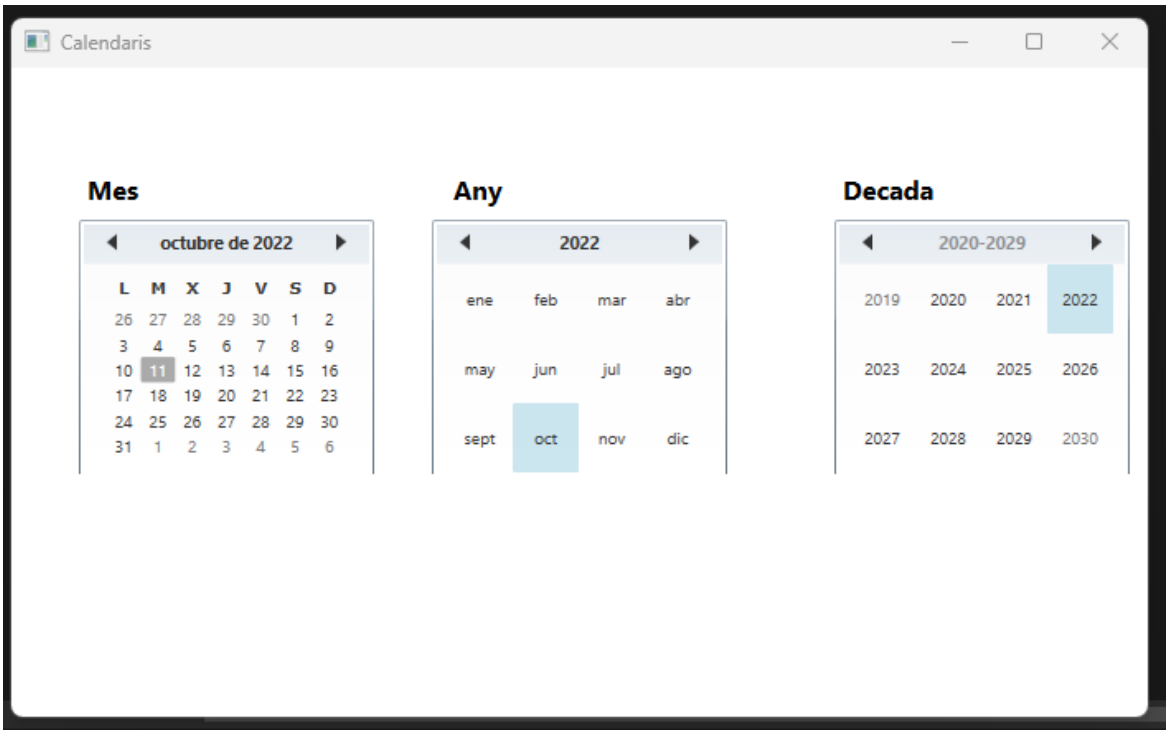
I recordeu, que quan menys text haja d'escriure l'usuari, menys errors de dades es produiran.

2. Introducció de dates

2.1. **Calendaris.**

Aquest component ens serveix per a triar **dates** en un calendari sense haver d'escriure-les. Disposa de nombroses opcions i formes de funcionament. Enumerarem els principals:

- **Display Modes**: indica la forma de visualitzar un calendari. Els valors poden ser: *month, year* o *decade*. La apariència per defecte és **month**.



- **Selection Modes**: indica la quantitat de dates que es poden seleccionar. Les maneres són els següents:
  - *None*: no es permet cap selecció
  - *SingleDate*: solament es pot seleccionar una data. **Opció per defecte**.
  - *SingleRange*: selecciona un rang de dates
  - *MultipleRange*: pot seleccionar diversos rangs de dates

- **BlackoutDates**: representa un rang de dates que estan deshabilitades per a la seua selecció. Apareixen ratllades


```
<Calendar.BlackoutDates>
      <CalendarDateRange Start="10/3/2022" End="10/7/2022"/>
      <CalendarDateRange Start="10/13/2022" End="10/20/2022"/>
      <CalendarDateRange Start="11/13/2022" End="11/20/2022"/>
</Calendar.BlackoutDates>
```

Atenció: les dates estan en format mm/dd/yyyy

```
1.  private void SetBlackOutDates()
2.  {
3.      MonthlyCalendar.BlackoutDates.Add(new CalendarDateRange(
4.          new DateTime(2010, 3, 1),
5.          new DateTime(2010, 3, 7)
6.          ));
7.      MonthlyCalendar.BlackoutDates.Add(new CalendarDateRange(
8.          new DateTime(2010, 3, 8),
9.          new DateTime(2010, 3, 8)
10.         ));
11.     MonthlyCalendar.BlackoutDates.Add(new CalendarDateRange(
12.       new DateTime(2010, 3, 15),
13.       new DateTime(2010, 3, 15)
14.       ));
15.     MonthlyCalendar.BlackoutDates.Add(new CalendarDateRange(
16.       new DateTime(2010, 3, 22),
17.       new DateTime(2010, 3, 22)
18.       ));
19.     MonthlyCalendar.BlackoutDates.Add(new CalendarDateRange(
20.       new DateTime(2010, 3, 29),
21.       new DateTime(2010, 3, 29)
22.       ));
23. }
```

- **DisplayDateStart**: permet establir la data inicial que mostrarà el calendari.
- **DisplayDayEnd**: la propietat anterior es completa amb aquesta, que ens indicarà la data final que es visualitzarà.
- **FirstDayOfWeek**: Ens permet canviar el primer dia de la setmana. Per exemple, podem posar com a primer dia de la setmana el dijous, això farà que el nostre primer dia de la setmana serà dijous. Per defecte, si no especifiquem el llenguatge, el primer dia és el diumenge.
- **IsTodayHighlighted**: indica si la data actual és destacada o ombrejada.
- **SelectedDate**: en aquesta propietat s'emmagatzema la data seleccionada. **Per defecte, el component no té cap data seleccionada**.
- **SelectedDates**: si la selecció múltiple està activada, llavors en aquesta propietat s'emmagatzemarà una llista amb les dates seleccionades.

Esdeveniments de Calendar

A part dels esdeveniments generals de qualsevol control, aquest té alguns específics que calen destacar:

- **DisplayDateChanged**: s'activa en canviar la propietat *DisplayDate*. Aquesta propietat li indica al calendari la data que ha de mostrar inicialment. Per exemple, si posem *DisplayDate="10/05/2016"* el calendari ens mostrarà inicialment el mes que correspon amb aquesta data. Cal destacar que la **data es posa en format anglés**: primer el mes, després el dia i finalment l'any.
- **DisplayModeChanged**: s'activa en canviar la propietat *DisplayMode*. És a dir, quan canvia la forma de visualització: per mes, any o dècada.
- **SelectedDatesChanged**: s'activa en canviar la propietat *SelectedDate*. És a dir, quan es canvia la selecció de la data. També s'activa amb *SelectedDates* quan es canvia o afig una data a la llista.

2.2. **DatePicker.**

Aquest component és similar a l'anterior perquè serveix per a triar una data des d'un calendari, no obstant això té l'aspecte d'un camp de text que polsant sobre ell podem triar la data en un calendari.

Per defecte es proposa la data actual i per a canviar-la utilitzarem la propietat **SelectedDate**.

La resta de propietats són similars a les que disposa el control **Calendar**.

3. Selecció d'opcions

Per a seleccionar diferents opcions tenim dos tipus de controls fonamentalment: **RadioButton** i **Casella de selecció**.

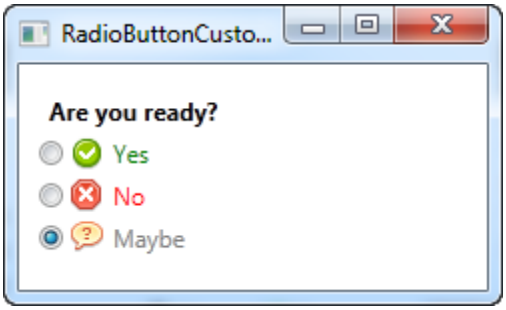
Tots dos ens permeten seleccionar opcions, encara que existeix el següent matís: el control *RadioButton* s'utilitza per a triar entre diverses opcions que són excloents i que per tant solament podrem triar una en un moment determinat. Per contra, la component *Casella de selecció* permet triar diverses opcions simultàniament.

3.1. **Radio Button.**

Com s'ha indicat anteriorment, s'utilitzen per a seleccionar opcions incompatibles en un moment donat. Perquè els RadioButton funcionen de forma sincronitzada (en marcar un es desmarca l'anterior, hem d'incorporar-los a un grup. Això s'especifica en la propietat **GoupName**.

Per defecte no apareix cap botó seleccionat, si volem fer-ho hem d'activar la propietat **IsChecked** en el RadioButton que volem que aparega seleccionat.

També podem tindre icones al costat del text posant panells dins del component.




```
<RadioButton>
    <WrapPanel>
        <Image Source="/WpfTutorialSamples;component/Images/accept.png" Width="16"
Height="16" Margin="0,0,5,0" />
        <TextBlock Text="Yes" Foreground="Green" />
    </WrapPanel>
</RadioButton>
<RadioButton Margin="0,5">
    <WrapPanel>
        <Image Source="/WpfTutorialSamples;component/Images/cancel.png" Width="16"
Height="16" Margin="0,0,5,0" />
        <TextBlock Text="No" Foreground="Red" />
    </WrapPanel>
</RadioButton>
<RadioButton IsChecked="True">
    <WrapPanel>
        <Image Source="/WpfTutorialSamples;component/Images/question.png" Width="16"
Height="16" Margin="0,0,5,0" />
        <TextBlock Text="Maybe" Foreground="Gray" />
    </WrapPanel>
</RadioButton>
```

Cal destacar també la propietat **IsChecked** per a saber si un RadioButton està seleccionat o no.

Finalment afegir que l'esdeveniment que sol utilitzar-se en aquests components és **Checked**. Es tracta de l'esdeveniment que s'activa en seleccionar el RadioButton.

3.2. **Casella de selecció.**

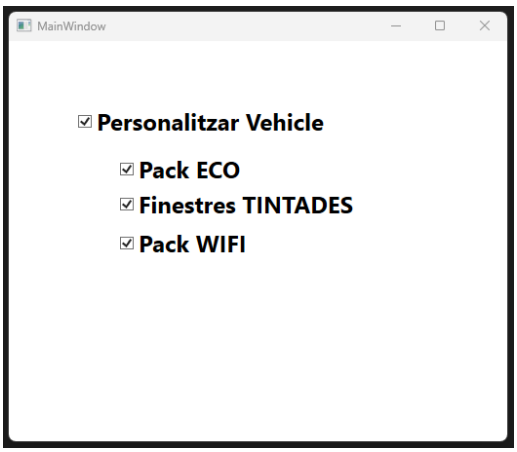
Aquest component és el complement a l'anterior lloc que ens permet seleccionar diverses opcions alhora.

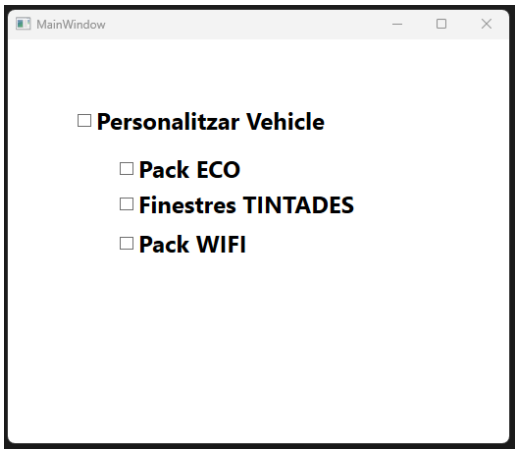
Podem personalitzar el nostre control afegint panells dins del mateix per a poder afegir icones. Per exemple:

Una de les propietats personals d'aquest control és **IsChecked** igual que succeïa amb l'anterior component.

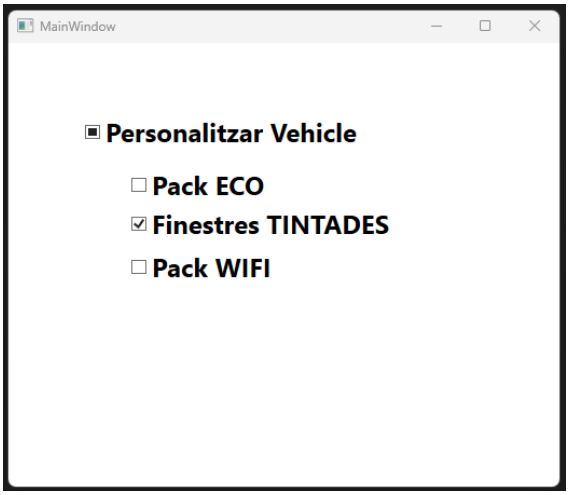
Aquest component té una propietat anomenada **IsThreeState** que quan està a true habilita un tercer estat. Habitualment, aquest control té dos estats (true or false) si està marcat o no. Amb aquesta propietat podem afegir un tercer estat indeterminat .

Aquest estat s'utilitza quan tenim una Casella de selecció que habilita o deshabilita a un conjunt de caselles de selecció que depèn d'ella. Si tenim tots els fills marcats, llavors el component principal apareixerà marcat, per contra, si els tenim desmarcat, llavors apareixerà desmarcat.





Però què ocorre quan solament tenim un dels fills seleccionat? en aquest cas el control principal tindrà un estat d'indeterminat.



Els esdeveniments utilitzats principalment pel control són **Checked** i **UnChecked** que s'activen en marcar o desmarcar el component.

Observa el codi que fa possible aquest comportament:

Fitxer .**xaml**:

```
<Grid>
        <CheckBox x:Name="chkPersComplet" IsThreeState="True"
Checked="chkPersComplet_CheckedChanged" Unchecked="chkPersComplet_CheckedChanged"
Content="Personalitzar Vehicle" HorizontalAlignment="Left" VerticalContentAlignment="Center"
Margin="70,67,0,0" VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>
        <CheckBox x:Name="chkECO" Content="Pack ECO" Checked="chkPers_Checked"
Unchecked="chkPers_Checked" HorizontalAlignment="Left" VerticalContentAlignment="Center"
Margin="113,116,0,0" VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>
        <CheckBox x:Name="chkTINTADES" Checked="chkPers_Checked" Unchecked="chkPers_Checked"
IsChecked="True" Content="Finestres TINTADES" HorizontalAlignment="Left"
VerticalContentAlignment="Center" Margin="113,152,0,0" VerticalAlignment="Top" FontSize="24"
FontWeight="Bold"/>
        <CheckBox x:Name="chkWIFI" Checked="chkPers_Checked" Unchecked="chkPers_Checked"
Content="Pack WIFI" HorizontalAlignment="Left" VerticalContentAlignment="Center"
Margin="113,192,0,0" VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>

    </Grid>
```

Fitxer **.cs**:

```
private void chkPersComplet_CheckedChanged(object sender, RoutedEventArgs e)
        {
            bool nouValor = (chkPersComplet.IsChecked == true);
            this.chkECO.IsChecked = nouValor;
            this.chkTINTADES.IsChecked = nouValor;
            this.chkWIFI.IsChecked = nouValor;
        }

private void chkPers_Checked(object sender, RoutedEventArgs e)
        {
            chkPersComplet.IsChecked = null;

             if ((chkECO.IsChecked == true) && (chkTINTADES.IsChecked == true) &&
             (chkWIFI.IsChecked == true))
                chkPersComplet.IsChecked = true;

             if ((chkECO.IsChecked == false) && (chkTINTADES.IsChecked == false) &&
             (chkWIFI.IsChecked == false))
                chkPersComplet.IsChecked = false;
        }
```

4. Llistes desplegables

Tenen la grandària d'un botó i solament quan es seleccionen despleguen la llista de valors a seleccionar. Per això el seu nom.

Per a afegir els elements utilitzarem en aquest cas l'etiqueta **ComboBoxItem** si el que volem és fer-ho en xaml. Per codi s'afegeixen mitjançant llistes.

L'esdeveniment principal també és **SelectionChanged**.

Hi ha una propietat que no es troba en el component anterior i és **Text.** Aquesta propietat mostra un missatge en el component tipus ***Selecciona una opció*** que no es tracta d'un element sinó de simplement un missatge informatiu.

Perquè aquest missatge siga **visible hem de col·locar a true** les propietats **IsReadOnly** i **IsEditable**, en cas contrari no podrem visualitzar-lo.

4.1. **Carrega de dades.**

Aquest control visualitza una llista d'elements que poden seleccionar-se. Depenent de la propietat **SelectionMode** que ens permet canviar el tipus de selecció d'elements de múltiple a simple.

Tenim dues maneres d'afegir dades a la nostra llista:

- **Mitjançant XAML:** afegint **ComboBoxItem** al control

```
<Grid>
        <ComboBox x:Name="cmbColors" HorizontalAlignment="Left" Margin="42,37,0,0"
VerticalAlignment="Top" Width="160" RenderTransformOrigin="0.438,0.004" FontSize="20">
            <ComboBoxItem>Roig</ComboBoxItem>
            <ComboBoxItem IsSelected="True">Verd</ComboBoxItem>
            <ComboBoxItem>Groc</ComboBoxItem>
        </ComboBox>
</Grid>
```

  En aquest exemple es mostraran tres opcions i la segona serà la seleccionada per defecte.

- **Amb codi C#:** associant una llista a la propietat **Items** o **ItemsSource**. Amb la propietat Items afegim element a element, mentre que amb la propietat ItemsSource podem afegir una llista sencera.

  Exemple 1:

```
cmbBegudes.ItemsSource = LoadListBoxData();

cmbBegudes.SelectedIndex = 3;

private ArrayList LoadListBoxData() {
      ArrayList itemsList = new ArrayList();
      itemsList.Add("Café");
      itemsList.Add("Té");
      itemsList.Add("Suc de taronja");
      itemsList.Add("Llet");
      itemsList.Add("Aigüa");
      return itemsList;
}
```

  Exemple 2:

```
    cmbBegudes2.Items.Add("Coffie");
    cmbBegudes2.Items.Add("Café");
    cmbBegudes2.Items.Add("Té");
    cmbBegudes2.Items.Add("Suc de taronja");
    cmbBegudes2.Items.Add("Llet");
    cmbBegudes2.Items.Add("Aigüa");

    cmbBegudes2.SelectedIndex = 1;
```

La propietat **Items** es tracta d'una col·lecció de C# per tant podem utilitzar tots els seus mètodes associats per a afegir, inserir, eliminar, buscar, ordenar, …

Igual que succeeix amb la majoria dels controls de WPF podem incorporar imatges en els nostres controls **ComboBoxItems** afegint un **layout** dins d'ell i col·locant els elements necessaris.

4.2. **Propietats de selecció**

Les propietats més importants d'aquest component són les relacionades amb la selecció dels elements. A continuació descriurem les més importants:

- **SelectedIndex**: indica l'índex de l'element seleccionat. Si la selecció múltiple està activada llavors retorna l'índex del primer element seleccionat. En cas de no tindre cap element seleccionat retornarà un **-1**.
- **SelectedItem**: indica l'element seleccionat. D'igual manera que l'anterior, si tenim activada la selecció múltiple, solament retornarà el primer element seleccionat. Retorna **null** en cas de no seleccionar res.
- **SelectedItems**: retorna la llista d'elements seleccionats en la selecció múltiple.
- **SelectedValue**: indica el valor de l'element seleccionat en funció de l'especificat en la propietat següent. Les anteriors propietats retornen un objecte, mentre que està propietat retorna el valor mostrat en la llista. Per defecte mostra una cadena amb el valor de la propietat **toString()** de l'objecte i el seu valor.
- **SelectedValuePath**: aquesta propietat indica quin valor retornarà la propietat anterior. Per exemple, si volem que retorne l'ID de l'objecte, llavors col·locarem l'etiqueta ID en aquesta propietat i en la propietat anterior ens retornarà l'ID de l'objecte. D'aquesta forma és possible afegir a la llista un Array mes complex amb mes d'un camp.

  Si no especifiquem res en aquesta propietat, llavors en la propietat **SelectedValue** es retornarà la funció **ToString** de l'objecte.

  Per exemple, si utilitzem ComboBoxItems, la qual cosa ens retorna **SelectedValue** serà la següent cadena de caràcters: *System.Windows.Controls.ListBoxItems: Valor*

  Si volguérem que solament ens retornara *Valor* hauríem d'especificar **Content** en la propietat **SelectedValuePath**.

- **SelectionMode**: indica si la selecció serà múltiple o simple. Per defecte està activada la simple.

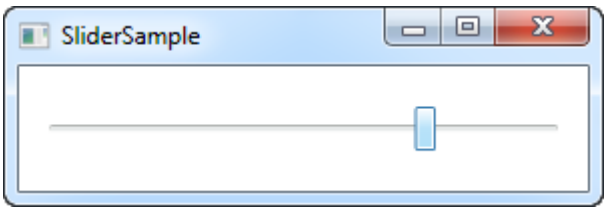
L'esdeveniment que sol utilitzar-se és SelectionChanged que s'activa en canviar d'elements seleccionat.

5. Controls de entrades numèriques

Fins ara hem vist components per a poder introduir dades de text, dates, opcions lògiques o elements seleccionats des d'una llista. En aquest punt es manegen controls per a introduir valors numèrics.

5.1. **Slider**

Ens permet seleccionar un valor numèric arrossegant marca al llarg d'una línia, que pot ser vertical o horitzontal.



S'estableix un valor mínim, situat en la part esquerra de la línia, i valor màxim situat en la part dreta.

Per a millorar la precisió a l'hora de moure la marca al llarg de la línia podem establir unes marques que ens ajuden. Aquestes marques reben el nom de **Ticks**.

Per a saber quin és el valor que hem seleccionat en moure la marca utilitzarem la propietat **Value**.

L'esdeveniment associat per a saber quan canvia el valor del Slider és **ValueChanged**.

Xaml

```
<Grid>
      <Slider x:Name="sldNum" ValueChanged="sldNum_ValueChanged"
      HorizontalAlignment="Left" Margin="91,336,0,0" VerticalAlignment="Top" Width="393"
      Height="25" Maximum="100" TickPlacement="BottomRight" TickFrequency="5"
      IsSnapToTickEnabled="True" SmallChange="1" AutoToolTipPlacement="TopLeft"/>

      <Label x:Name="lblNum" Content="0" HorizontalAlignment="Left"
       Margin="261,157,0,0" VerticalAlignment="Top" FontSize="48" FontWeight="Bold"/>
       <Label x:Name="lblNum" Content="0" HorizontalAlignment="Left" Margin="261,157,0,0"
       VerticalAlignment="Top" FontSize="48" FontWeight="Bold"/>

</Grid>
```

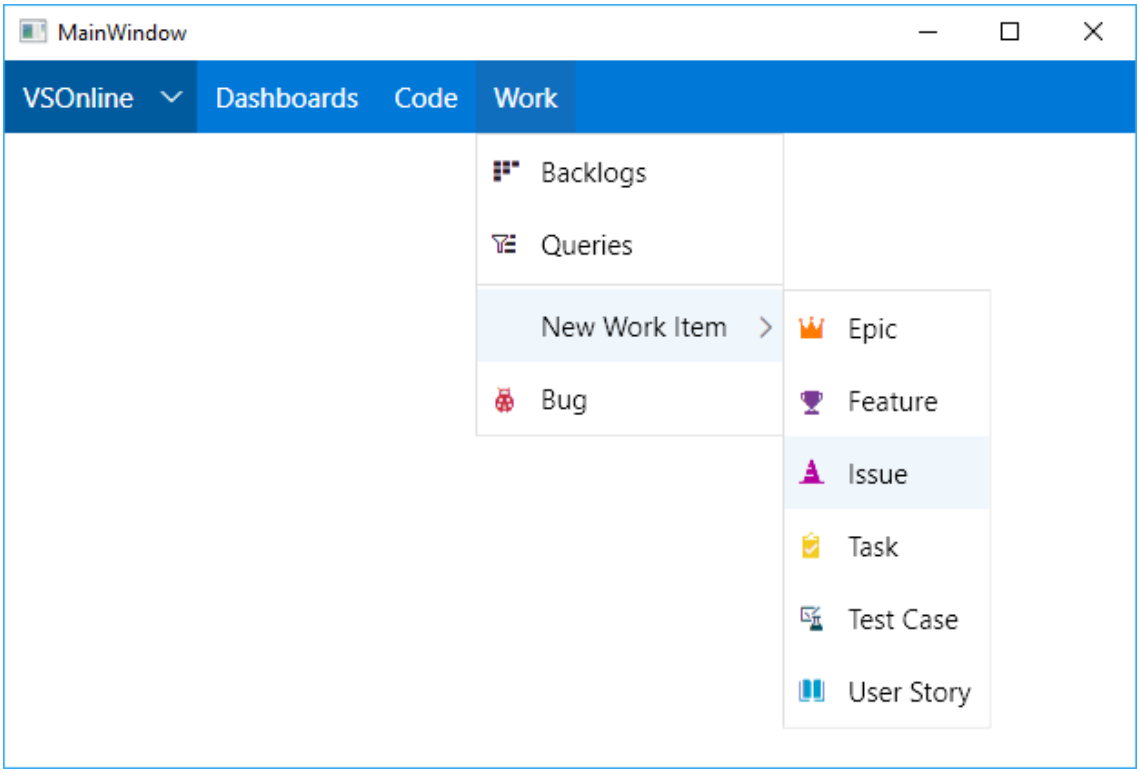
C#

```
    private void sldNum_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)
    {
            this.lblNum.Content = sldNum.Value.ToString();
    }
```

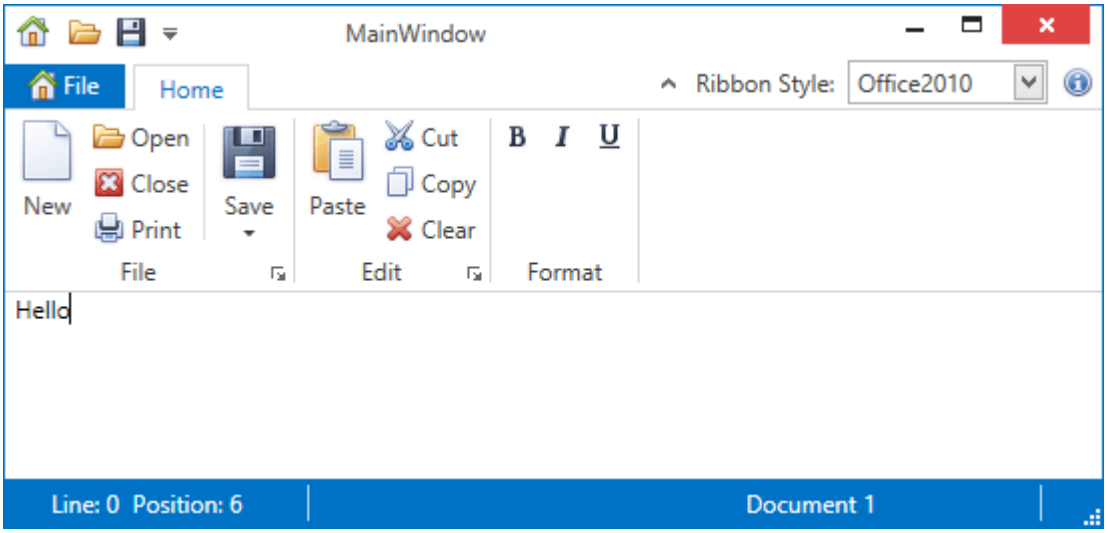
6. Organització finestra principal

En aquest punt tractarem dels elements que solen construir una interfície principal per a organitzar totes les opcions i possibilitats que puga tindre.

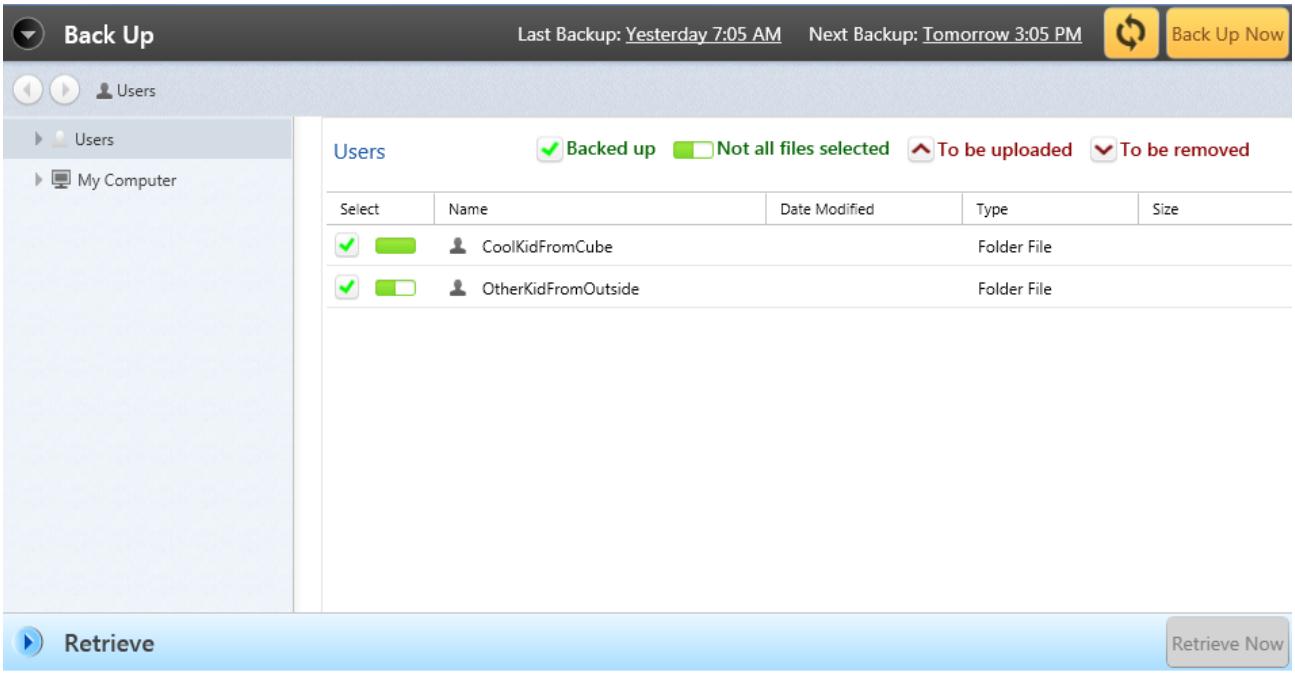
Analitzarem les principals, encara que sempre pot haver-hi variacions d'un framework a un altre. Totes elles poden combinar-se i d'aquesta manera aconseguir una interfície més rica i flexible.



Aquests components poden seguir un enfocament més clàssic com una **barra de menús** que té associada una **barra d'eines**, o poden tindre un enfocament més modern com la famosa cinta o ribbon de Microsoft.

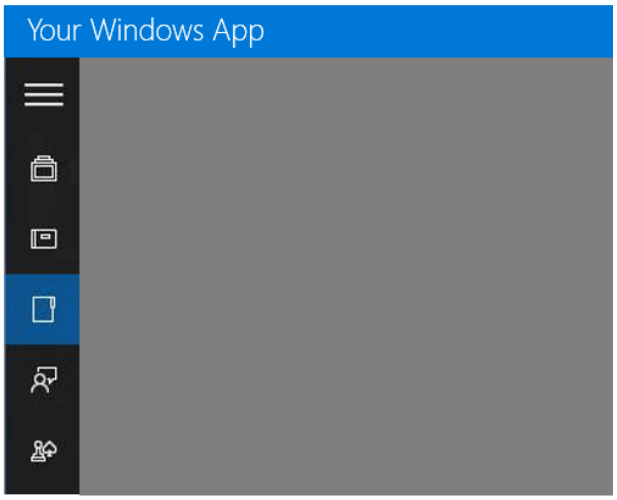


També podem orientar la nostra aplicació a un estil més semblant a una web, amb un menú o un element acordió en el qual albergar les principals opcions.



Una altra possibilitat consistiria a disposar d'una interfície principal amb uns **botons que ens porten a les diferents opcions del programa**. Aquest tipus d'interfícies són útils en cas d'utilitzar **pantalles tàctils** perquè faciliten l'interacció amb l'usuari, no obstant això són una mica enutjoses per a interfícies sense dispositiu tàctil perquè perds la perspectiva del conjunt de l'aplicació. Sobretot si existeixen multitud d'opcions.

Una altra opció que s'aplica molt en aplicacions mòbils és la implementació d'un ***Hamburguer Menu*** que disposa d'una barra d'opcions que mostra simplement els botons principals, però que després podem desplegar per a veure el total de les opcions de les quals disposa el programa.



Per a complementar les anteriors sempre sol utilitzar-se una distribució del contingut de l'aplicació mitjançant pestanyes, encara que, com ja he dit per a complementar , ja que **no és gens recomanable** realitzar una interfície solament amb pestanyes, ja que en cas de tindre moltes opcions el programa es torna immanejable.

6.1. **Barra de menú.**

Aquest control s'utilitza per a l'organització de les opcions de les quals disposa un programa, se situa en la part superior i divideix les funcionalitats en seccions, de les quals pengen les subfuncionalitats.

Aquest tipus de components estructuren de manera jeràrquica les diferents opcions de les quals disposa una aplicació.

Amb aquest control se solen utilitzar propietats generals per a la seua configuració com per exemple: **Background, Effect, Height, Width o ToolTip**.

Hi ha altres propietats que solen utilitzar-se amb aquest control i que són més particulars d'aquest. Com per exemple:

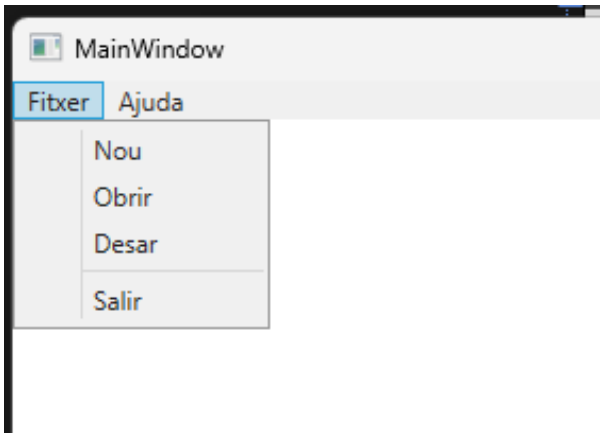
- **Items**: col·lecció on es guarden els ítems del menú.
- **IsMainMenu**: determina si el menú és el principal o no.

Aquest és el control principal que actua com a contenidor de les opcions que pengen del menú.

MenuItem

La barra de menú es mostra buida ja que es tracta d'un contenidor simplement. Per a afegir opcions hem d'utilitzar la classe **MenuItem**, mijançant la qual podrem especificar les opcions de manera jeràrquica. Per exemple, si definim un MenuItem i afegim diverses opcions dins d'ell s'estructurarà de manera jeràrquica i aquests passaran a ser sub ítems del principal.

```
<Menu x:Name="menuPrueba" HorizontalAlignment="Left" Height="Auto"
VerticalAlignment="Top" Width="518" IsMainMenu="True">
            <MenuItem Header="Fitxer">
                <MenuItem Header="Nou" />
                <MenuItem Header="Obrir" />
                <MenuItem Header="Desar" />
                <Separator />
                <MenuItem Header="Salir" />
            </MenuItem>
            <MenuItem Header="Ajuda">
                <MenuItem Header="Sobre l'aplicació"/>
            </MenuItem>
</Menu>
```



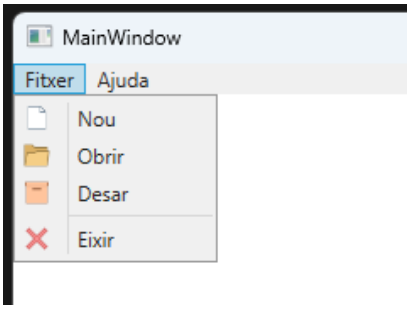
Fixeu-vos que ens queden buits a l'esquerra dels noms dels ítems, aquests es deixen per a afegir altres components com a icones o caselles de selecció.

També s'utilitza el control **Separator** per a organitzar les diferents opcions.

Per a afegir icones podem fer-ho de la següent forma:

```
<Menu x:Name="menuPrueba" HorizontalAlignment="Left" Height="Auto"
VerticalAlignment="Top" Width="518" IsMainMenu="True">
            <MenuItem Header="Fitxer">
                <MenuItem Header="Nou">
                    <MenuItem.Icon>
                        <Image Source="/nou.png"/>
                    </MenuItem.Icon>
                </MenuItem>
                <MenuItem Header="Obrir">
                    <MenuItem.Icon>
                        <Image Source="/obrir.png"/>
                    </MenuItem.Icon>
                </MenuItem>
                <MenuItem Header="Desar">
                    <MenuItem.Icon>
                        <Image Source="/desar.png"/>
                    </MenuItem.Icon>
                </MenuItem>
                <Separator />
                <MenuItem Header="Eixir">
                    <MenuItem.Icon>
                        <Image Source="/eixir.png"/>
                    </MenuItem.Icon>
                </MenuItem>
            </MenuItem>
            <MenuItem Header="Ajuda">
                <MenuItem Header="Sobre l'aplicació"/>
            </MenuItem>
</Menu>
```

En la figura següent es pot apreciar el resultat

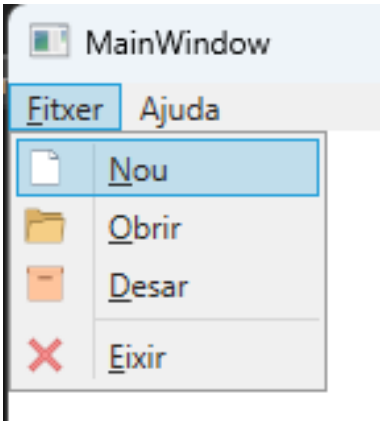


Per a activar les opcions d'un menú podem utilitzar el ratolí o bé el teclat. Per a subratllar la tecla que utilitzarem per a accedir a l'opció del menú mitjançant el teclat col·locarem el guió baix abans de la lletra que volem que siga la seleccionada, per exemple **"_Obrir".**

Per exemple, modifiquem l'exemple anterior de la següent manera:

```
<Menu x:Name="menuPrueba" HorizontalAlignment="Left" Height="Auto"
VerticalAlignment="Top" Width="518" IsMainMenu="True">
            <MenuItem Header="_Fitxer">
                <MenuItem Header="_Nou">
                    <MenuItem.Icon>
                        <Image Source="/nou.png"/>
                    </MenuItem.Icon>
                </MenuItem>
                <MenuItem Header="_Obrir">
                    <MenuItem.Icon>
                        <Image Source="/obrir.png"/>
                    </MenuItem.Icon>
                </MenuItem>
                <MenuItem Header="_Desar">
                    <MenuItem.Icon>
                        <Image Source="/desar.png"/>
                    </MenuItem.Icon>
                </MenuItem>
                <Separator />
                <MenuItem Header="_Eixir">
                    <MenuItem.Icon>
                        <Image Source="/eixir.png"/>
                    </MenuItem.Icon>
                </MenuItem>
            </MenuItem>
            <MenuItem Header="Ajuda">
                <MenuItem Header="Sobre l'aplicació"/>
            </MenuItem>
</Menu>
```

En la figura següent s'observa el resultat:



D'aquesta forma al polsar **Alt+F** s'obriria el menú **Fitxer** i amb **Alt+O** el menú **Obrir.**

6.2. **Barra d'eines.**

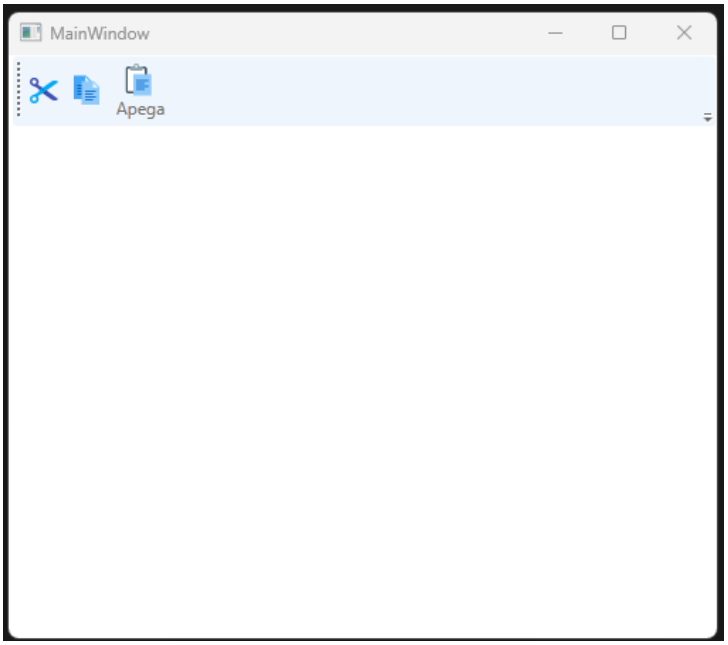
Aquest control s'assembla a un panell amb botons, encara que té algunes funcionalitats que el distingeixen, com per exemple la possibilitat d'afegir separadors de manera senzilla, reubicar les diferents barres d'eines, o fins i tot mostrar de forma solapada aquells components que no es visualitzen en la barra d'eines per tindre una amplària massa xicoteta.

Aquest control està associat a **ToolBarTray** que ens serveix de contenidor en cas de disposar de diverses barres d'eines. En cas d'utilitzar una solament podem deixar-la en solitari sense el control **ToolBarTray**.

Es defineixen les etiquetes de **ToolBar** i després s'afegeixen els botons i separadors.

```
<DockPanel>
        <ToolBar DockPanel.Dock="Top">
            <Button Command="Cut" ToolTip="Tallar">
                <Image Source="/icons8-cut-48.png" Height="24" />
            </Button>
            <Button Command="Copy" ToolTip="Copiar">
                <Image Source="/icons8-copy-48.png" Height="24"/>
            </Button>
            <Button Command="Paste" ToolTip="Apegar">
                <StackPanel >
                    <Image Source="/icons8-paste-48.png" Height="24" />
                    <TextBlock Margin="3,0,0,0">Paste</TextBlock>
                </StackPanel>
            </Button>
        </ToolBar>
        <Grid>

        </Grid>
</DockPanel>
```

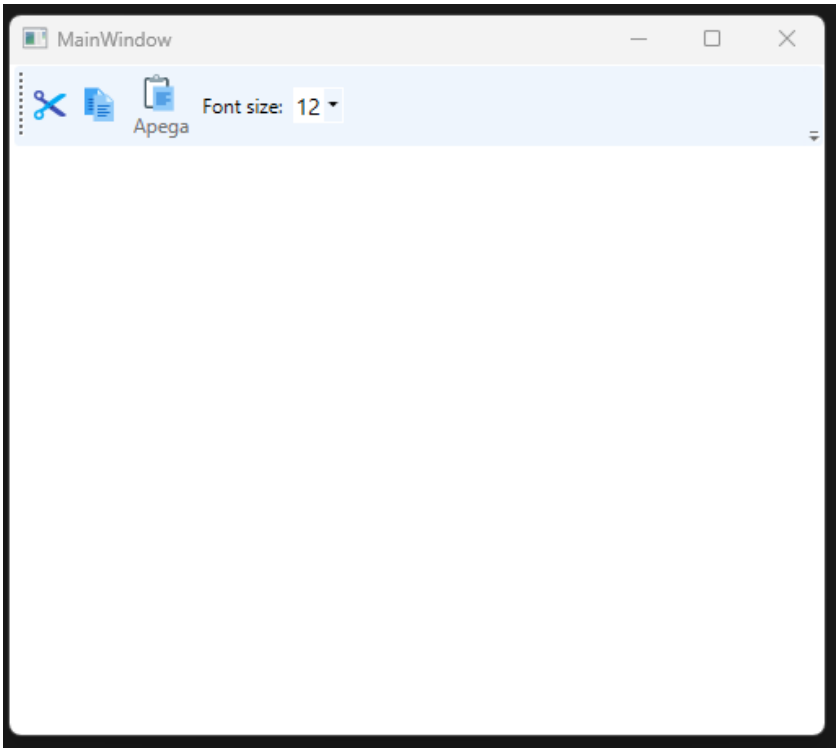
Aquest és el resultat que obtindríem:



També podem afegir components diferents, com per exemple un combobox

```
<ToolBar>
        <Label>Font size:</Label>
        <ComboBox>
              <ComboBoxItem>10</ComboBoxItem>
              <ComboBoxItem IsSelected="True">12</ComboBoxItem>
              <ComboBoxItem>14</ComboBoxItem>
              <ComboBoxItem>16</ComboBoxItem>
        </ComboBox>
</ToolBar>
```

Cada botó tindrà associat un control d'esdeveniments que realitzarà les accions oportunes.



6.3. **Ribbon.**

És un component de recent incorporació perquè va sorgir amb el programa Office 2007. Es tracta d'un panell superior on s'estructuren les diferents seccions per pestanyes. Dins de cada pestanya es tenen moltes opcions, les més habituals de la secció.

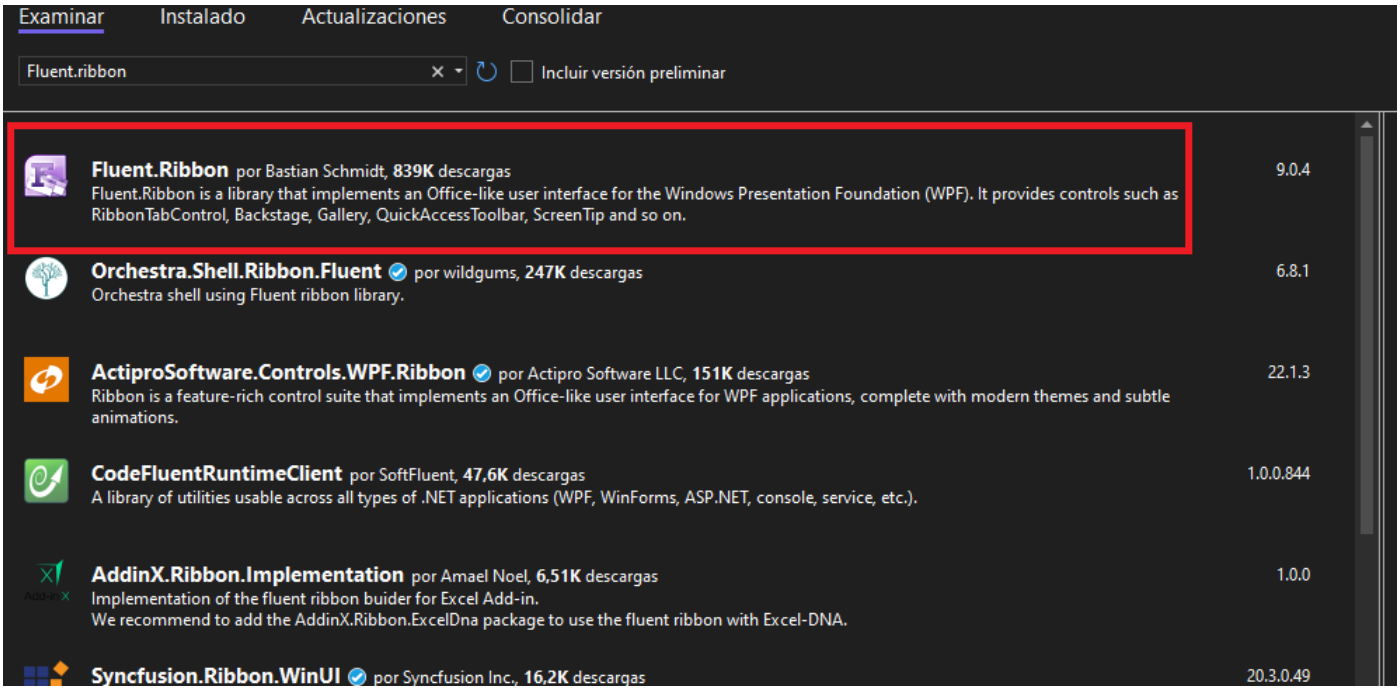
Podem agrupar les opcions per cada secció en grups de components. També tenim la possibilitat de desplegar més opcions de les mostrades.

Aquest tipus de components persegueixen mostrar de manera ràpida les principals opcions utilitzades d'un programa, mentre que les menys usades romanen ocultes llevat que vulguem utilitzar-les.

Podem utilitzar el control que proporciona Microsoft, o bé un més modern, anomenat **Fluent Ribbon**. El control proporcionat per Microsoft no ve inclòs per defecte al VSC, per tant ens centrarem en un control extern, el riibon de Fluent.

Fluent Ribbon

Utilitzarem el control proporcionat per Fluent . És pràcticament igual al de Microsoft, però amb un aspecte més modern. Es tracta d'un control que es troba en el **repositori de nuget** , per la qual cosa podem instal·lar-lo d'una manera senzilla.

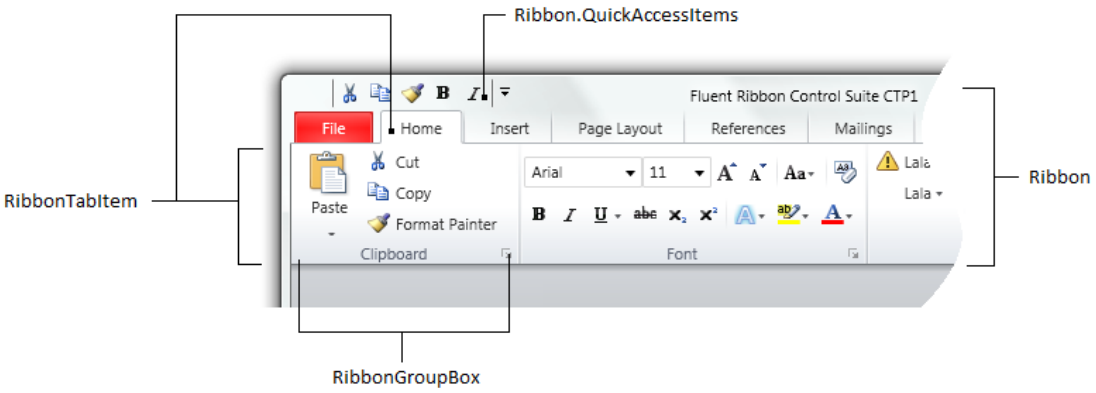


Després d'instal·lar-ho, hem d'afegir els estils en el nostre fitxer de configuració App.xaml:

```
<!-- Estils per a Fluent  Ribbon -->
<ResourceDictionary
 Source="pack://application:,,,/Fluent;Component/Themes/Generic.xaml" />
```

Finalment, per a incorporar-ho a la nostra finestra definim el conjunt d'etiquetes:

```
xmlns:Fluent="urn:fluent-ribbon"
```



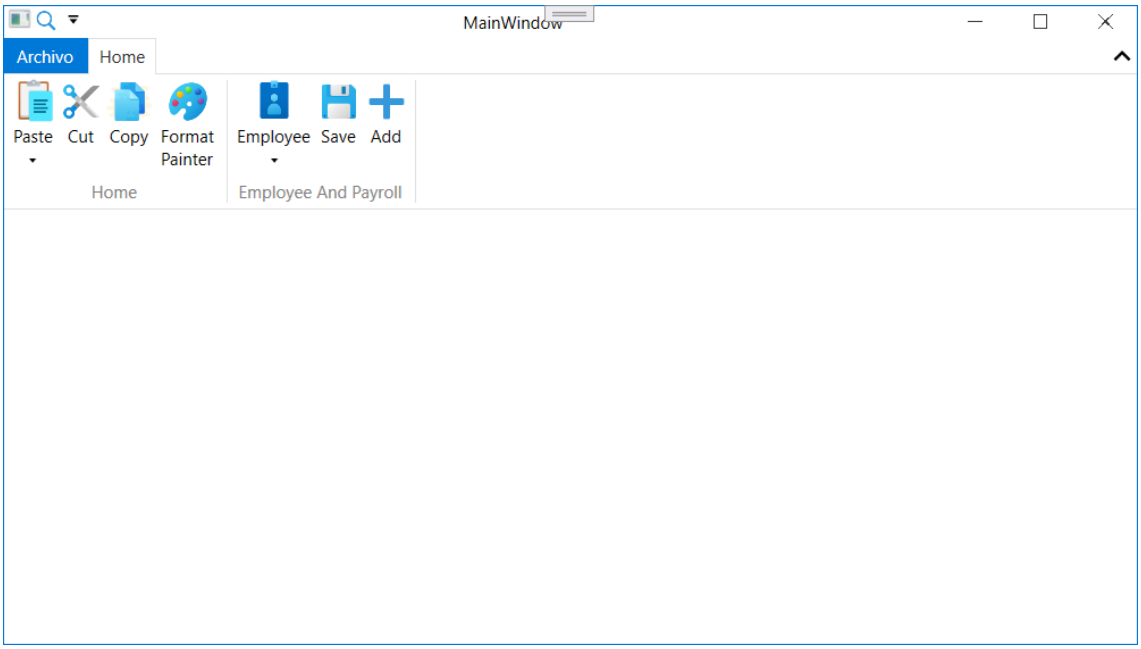
L'aspecte és més modern, però encara que no té les mateixes parts el control de Windows, si disposa de les principals.

No disposa del botó d'ajuda, però sí de les opcions d'accés ràpid, encara que per a poder veure-les hem de canviar el tipus de finestra a Fluent **:RibbonWindow**, en cas contrari no les podrem veure.

La part que es refereix a Application Menu es diu **Menu**. A continuació posem un exemple:

```
<Fluent:RibbonWindow x:Class="di.ejercicios.prueba.fluent.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:Fluent="urn:fluent-ribbon"
        xmlns:local="clr-namespace:di.ejercicios.prueba.fluent"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Fluent:Ribbon>
            <Fluent:Ribbon.QuickAccessItems>
                <!--Use Content or Target Property to set QAT item-->
                <Fluent:QuickAccessMenuItem IsChecked="true">
                    <Fluent:Button Header="Pink" Icon="Iconos/buscar.png" />
                </Fluent:QuickAccessMenuItem>
            </Fluent:Ribbon.QuickAccessItems>
            <Fluent:Ribbon.Menu>
                <Fluent:Backstage>
                    <Fluent:BackstageTabControl>
                        <Fluent:BackstageTabItem Header="Opciones" />
                        <Fluent:Button Header="Cerrar"
                                LargeIcon="Iconos/quit.png"  />
                    </Fluent:BackstageTabControl>
                </Fluent:Backstage>
            </Fluent:Ribbon.Menu>
            <Fluent:RibbonTabItem Header="Home"  >
                <!-- Home  group-->
                <Fluent:RibbonGroupBox x:Name="ClipboardGroup" Header="Home">
                    <Fluent:SplitButton LargeIcon="Iconos/paste.png"
                                Header="Paste" >
                        <Fluent:MenuItem Icon="Iconos/paste.png" Header="Keep
                                Text Only" />
                        <Fluent:MenuItem Icon="Iconos/paste.png" Header="Paste
                                Special..." KeyTip="S"/>
                    </Fluent:SplitButton>
                    <Fluent:Button LargeIcon="Iconos/cut.png" Header="Cut"/>
                    <Fluent:Button LargeIcon="Iconos/copy.png" Header="Copy"/>
                    <Fluent:Button LargeIcon="Iconos/format_painter.png"
                                Header="Format Painter"/>
                </Fluent:RibbonGroupBox>
```

```
            <!-- Employee And Payroll group-->
            <Fluent:RibbonGroupBox x:Name="Employee" Header="Employee And
                    Payroll">
                <Fluent:SplitButton LargeIcon="Iconos/personal.png"
                            Header="Employee">
                    <Fluent:MenuItem Icon="Iconos/paste.png" Header="Keep
                            Text Only" />
                    <Fluent:MenuItem Icon="Iconos/paste.png" Header="Paste
                            Special..." />
                </Fluent:SplitButton>
                <Fluent:Button LargeIcon="Iconos/save.png" Header="Save"/>
                <Fluent:Button LargeIcon="Iconos/add.png" Header="Add" />
            </Fluent:RibbonGroupBox>
        </Fluent:RibbonTabItem>
    </Fluent:Ribbon>
</Grid>
</Fluent:RibbonWindow>
```



Nota: Al moment de confeccionar aquest material el component Fluent Ribbon presentava problemes per a indicar les icones amb rutes relatives. Per aquest motiu els exemples que acompanyen aquesta unitat estan escrits amb rutes absolutes.

6.4. **Pestanyes.**

Les pestanyes ens permeten separar diferents continguts dins d'una mateixa funcionalitat. Normalment s'utilitzen com a suport per a poder diferenciar la informació mostrada en compartiments classificats segons la funcionalitat o el tipus d'informació que mostren.

No resulta eficaç realitzar una interfície únicament amb pestanyes, com ja he dit s'utilitza com a suport de les altres formes d'organització de l'aplicació. Realitzar una aplicació solament amb pestanyes implica que l'aplicació resulte poc escalable.

El component per a WPF és **TabControl** que és el contenidor de les pestanyes. Aquest control ens permet disposar les pestanyes de quatre formes diferents:

- En la part superior en horitzontal. Aquesta és l'opció **per defecte**.
- En la part inferior en horitzontal
- En la part esquerra en vertical
- En la part dreta en vertical

Per a canviar l'aspecte per defecte de la orientació de les pestanyes utilitzem la propietat **TabStripPlacement**.

Com ja hem dit aquest component és un contenidor de controls **TabItem** que representen una pestanya.

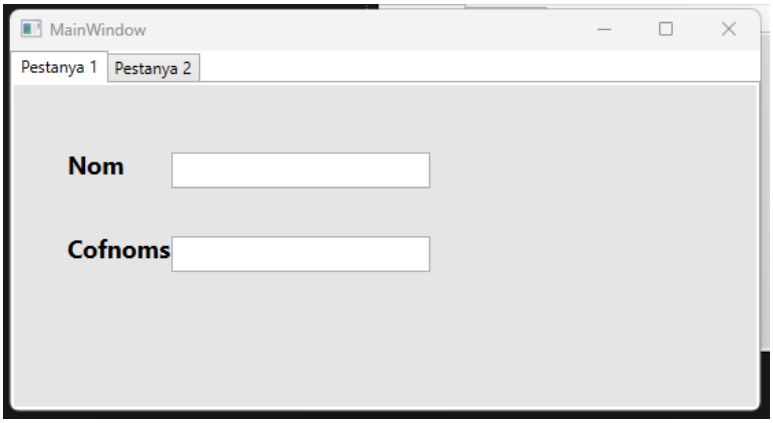
En un component TabControl col·locarem tants TabItems com a pestanyes tinguem. La propietat **Header** ens proporciona el títol de la pestanya. Fins i tot podem col·locar panells dins de la capçalera per a afegir text i imatges.

Podem recórrer les pestanyes que té el contenidor **TabControl** amb la propietat *Items*.

Aquest control té múltiples esdeveniments associats però el que més sol utilitzar-se és el que es dispara en canviar de pestanya. Aquest esdeveniment és **SelectionChanged**.

Vegem un exemple:

```
        <TabControl>
            <TabItem Header="Pestanya 1">
                <Grid Background="#FFE5E5E5">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="83*"/>
                        <ColumnDefinition Width="711*"/>
                    </Grid.ColumnDefinitions>
                    <Label Content="Nom" HorizontalAlignment="Left" Margin="34,41,0,0" VerticalAlignment="Top"
FontSize="18" FontWeight="Bold" Grid.ColumnSpan="2"/>
                    <Label Content="Cofnoms" HorizontalAlignment="Left" Margin="34,102,0,0"
VerticalAlignment="Top" RenderTransformOrigin="0.129,0.393" FontSize="18" FontWeight="Bold" Grid.ColumnSpan="2"/>
                    <TextBox HorizontalAlignment="Left" Margin="58,49,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="189" Height="26" FontSize="18" Grid.Column="1"/>
                    <TextBox HorizontalAlignment="Left" Margin="58,110,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="189" Height="26" FontSize="18" Grid.Column="1"/>
                </Grid>
            </TabItem>
            <TabItem Header="Pestanya 2">
                <Grid Background="#FFE5E5E5">
                    <Label Content="Altres dades" HorizontalAlignment="Left" Margin="34,41,0,0"
VerticalAlignment="Top" FontSize="18" FontWeight="Bold"/>
                    <TextBox HorizontalAlignment="Left" Margin="169,49,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="334" Height="154"/>
                </Grid>
            </TabItem>

        </TabControl>
```



6.5. **Hamburger Menú.**

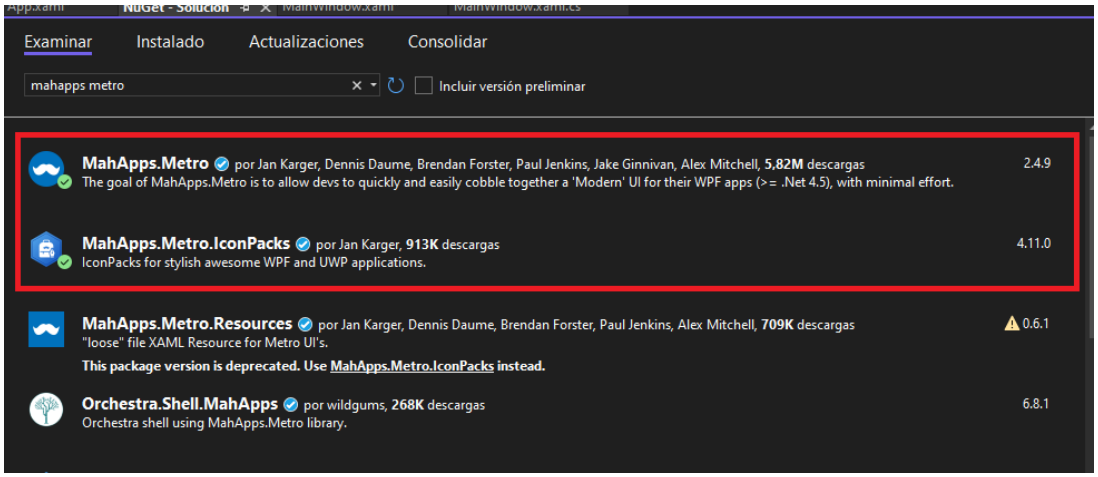
Es tracta d'un menú ocultable que es mostra en pitjar el botó de les tres ratlles, per aquesta icona es diu així a aquest component.

Es troba normalment a la cantonada esquerra superior i s'expandeix d'esquerra a dreta. Es compon d'una sèrie d'opcions principals que al polsar ens mostren un panell específic en la part central.

Per defecte, en WPF, no es disposa d'aquest control i s'ha d'afegir mitjançant la llibreria auxiliar de components i estils **MahApps**.

Recordem com es fa:

- Instal·lar des de l'administrador de Nugets:



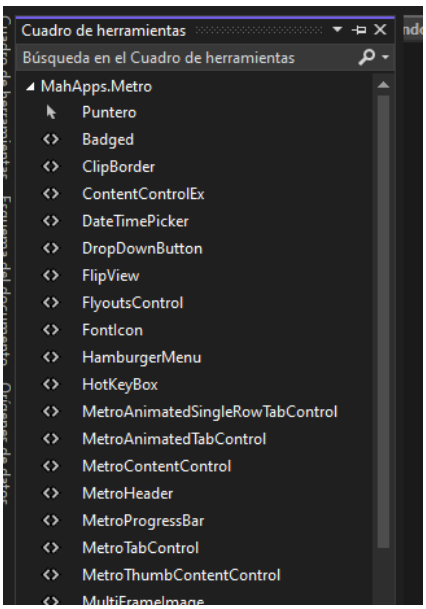
- Afegir al firxer App.xaml els recursos:

```
<ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <!-- MahApps.Metro resource dictionaries. Make sure that all file
names are Case Sensitive! -->
                <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Controls.xaml" />
                <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Fonts.xaml" />
                <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Colors.xaml" />
            </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

- Afegir a la finestra la llibreria d'etiquetes:

```
xmlns:mah="http://metro.mahapps.com/winfx/xaml/controls"
```

I ja podrem vore els controls al quadre d'eines:



Anem amb un exemple. Anirem afegint els elements que apareixeran en el nostre menú desplegable. Però per a això utilitzarem en primer lloc un *DataTemplate* per a maquetar els elements del menú (podeu vore més exemples a la pàgina de MahApps Metro). Igual que succeeix en les pàgines web amb CSS podem tindre un fitxer que modifique la forma de veure els components, per a això utilitzarem els Template o plantilles de WPF, que ens permeten definir la forma en la qual es veuran els nostres controls.

En aquest cas utilitzem unes plantilles per a veure els elements del HamburguerMenu. Per a això utilitzarem uns d'estils personalitzats per a implementar aquest tipus d'interfície. Al principi de la finestra trobem el codi que usa un dels tipus d'items del control (**HamburguerMenuGlyphItem)** aplicat en dues plantilles diferents, el menú `MenuItemTemplate` i les opcions de baix `OptionsMenuItemTemplate.`

Observem aquesta part del codi:

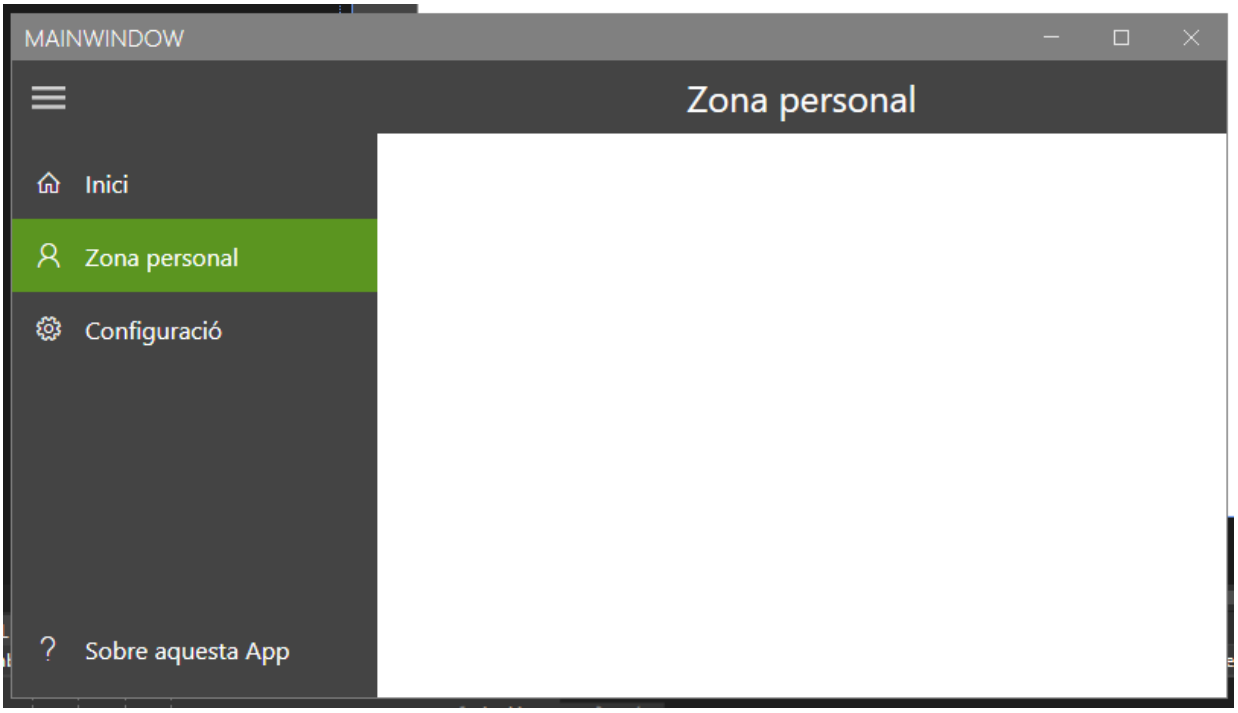
```
<Grid.Resources>
            <DataTemplate x:Key="MenuItemTemplate" DataType="{x:Type
mah:HamburgerMenuGlyphItem}">
                <Grid Height="48">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="48" />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Grid.Column="0"
                               FontSize="16"
                               HorizontalAlignment="Center"
                               VerticalAlignment="Center"
                               FontFamily="Segoe MDL2 Assets"
                               Text="{Binding Glyph}" />
                    <TextBlock Grid.Column="1"
                               VerticalAlignment="Center"
                               FontSize="16"
                               Text="{Binding Label}" />
                </Grid>
            </DataTemplate>

            <DataTemplate x:Key="OptionsMenuItemTemplate" DataType="{x:Type
mah:HamburgerMenuGlyphItem}">
                <Grid Height="48">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="48" />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <TextBlock Grid.Column="0"
                               FontSize="16"
                               HorizontalAlignment="Center"
                               VerticalAlignment="Center"
                               FontFamily="Segoe MDL2 Assets"
                               Text="{Binding Glyph}" />
                    <TextBlock Grid.Column="1"
                               VerticalAlignment="Center"
                               FontSize="16"
                               Text="{Binding Label}" />
                </Grid>
            </DataTemplate>


        </Grid.Resources>
```

Observa la definició del Grid de dues columnes, el nom de la plantilla (x:Key) i l'estil aplicat.

A continuació incorporem el control i definim els items del menú. Durant la definició del control utilitzem les plantilles creades i les apliquem a cadascuna d'aquestes seccions.

```
<mah:HamburgerMenu x:Name="HamburgerMenuControl"
                           IsPaneOpen="False"
                           ItemTemplate="{StaticResource MenuItemTemplate}"
                           OptionsItemTemplate="{StaticResource OptionsMenuItemTemplate}"
                           ItemClick="HamburgerMenuControl_OnItemClick"
                           OptionsItemClick="HamburgerMenuControl_OnItemClick"
                           SelectedIndex="1"
                           DisplayMode="CompactInline">

            <!--  Items  -->
            <mah:HamburgerMenu.ItemsSource>
                <mah:HamburgerMenuItemCollection>
                    <mah:HamburgerMenuGlyphItem Glyph="" Label="Inici"/>
                    <mah:HamburgerMenuGlyphItem Glyph="" Label="Zona personal"/>
                    <mah:HamburgerMenuGlyphItem Glyph="" Label="Configuració"/>
                </mah:HamburgerMenuItemCollection>
            </mah:HamburgerMenu.ItemsSource>

            <!--  Options  -->
            <mah:HamburgerMenu.OptionsItemsSource>
                <mah:HamburgerMenuItemCollection>
                    <mah:HamburgerMenuGlyphItem Glyph="" Label="Sobre aquesta App"/>
                </mah:HamburgerMenuItemCollection>
            </mah:HamburgerMenu.OptionsItemsSource>

            <!--  Content  -->
            <mah:HamburgerMenu.ContentTemplate>
                <DataTemplate DataType="{x:Type mah:HamburgerMenuItem}">
                    <Grid x:Name="TheContentGrid">
                        <Grid.RowDefinitions>
                            <RowDefinition Height="48" />
                            <RowDefinition />
                        </Grid.RowDefinitions>
                        <Border Grid.Row="0"
                                Background="#FF444444">
                            <TextBlock x:Name="Header"
                                       HorizontalAlignment="Center"
                                       VerticalAlignment="Center"
                                       FontSize="24"
                                       Foreground="White"
                                       Text="{Binding Label}" />
                        </Border>
                        <ContentControl x:Name="TheContent"
                                        Grid.Row="1"
                                        Focusable="False"
                                        Content="{Binding Tag}" />
                    </Grid>
                </DataTemplate>
            </mah:HamburgerMenu.ContentTemplate>

        </mah:HamburgerMenu>
```

I el resultat seria aquest:



7. Templates (plantilles)

Per a canviar l'aspecte dels components tenim els estils i també disposem dels *Templates*.

Encara que al final tenen la mateixa fi tenen les seues diferències que es detallen en la taula següent:

Estils	Templates
S'utilitzen per a modificar els valors per defecte de la propietat d'un component.	Permet modificar l'estructura d'un control al qual la plantilla és aplicada. Per exemple podem modificar la forma d'un botó i fer-lo redó.
Solament podem utilitzar propietats existents en el control.	També podem modificar les propietats d'un control però que no podran ser modificades per ho valores del control, solament es podrà fer en cas de realitzar ***template-binding*** amb les propietats del control. Les principals parts d'una plantilla són: • **Content Presenter**: s'utilitza per a canviar la manera de visualitzar el contingut del component. • **Header**: s'utilitza per a modificar la presentació de les capçaleres d'aquells components que les tenen com per exemple un **Tab Control**.

- **ItemsHost**: modifica la visualització dels objectes fill d'un component
- **ItemContainerTemplate**: aplica una plantilla al contenidor d'ítems.

Podem modificar el comportament dels nous elements mitjançant *Triggers*.

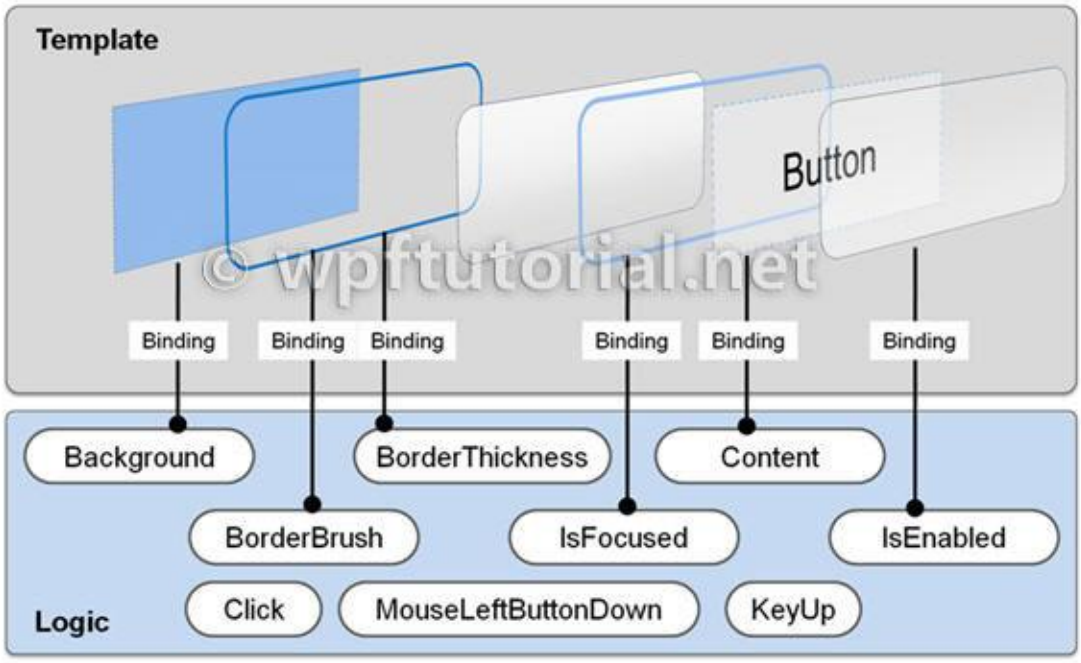
És important destacar que les propietats que volem que puguen ser modificades per l'usuari **han d'estar especificades en la plantilla com *Template-Binding*** en cas contrari la propietat no podrà canviar-se respecte a la plantilla original.

7.1. **Control Template.**

Els controls separen la lògica (estats, esdeveniments i propietats) de l'aparença visual del control (template).

Els controls tenen una plantilla per defecte que determina l'aspecte visual d'aquest. La plantilla per defecte s'adjunta juntament amb el control.

La plantilla és definida per una propietat anomenada **Template**. Canviant el valor d'aquesta propietat per una plantilla nova podem canviar l'aspecte d'un botó.



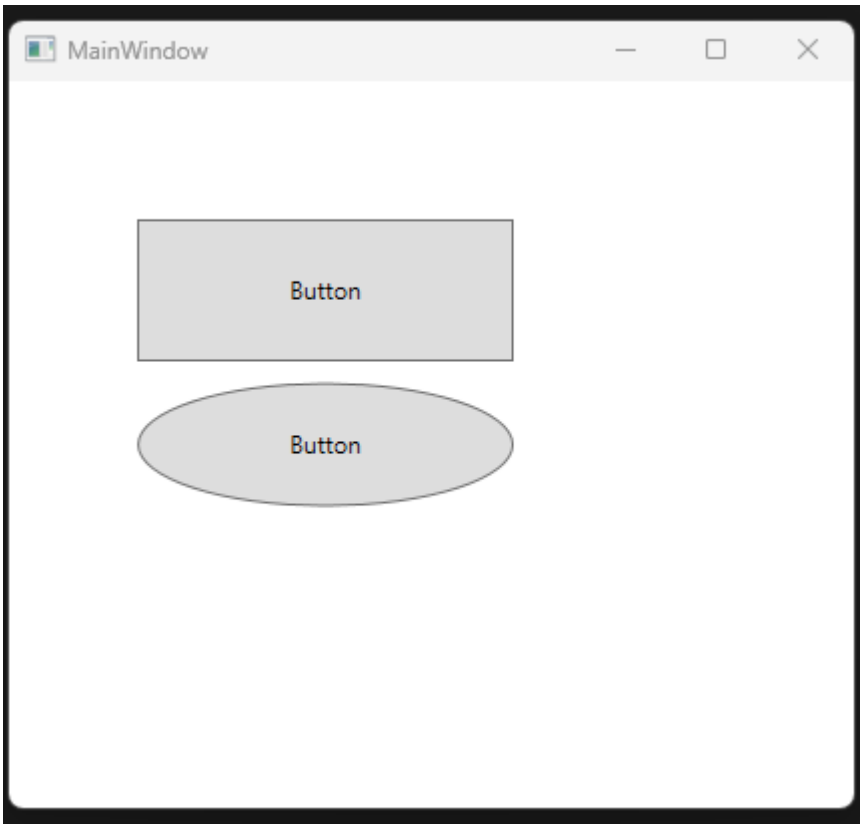
Les plantilles es defineixen dins de la secció **Style** del control. A continuació es mostra un exemple en el qual es canvia la forma per defecte d'un botó perquè siga el·líptica.

```
<Style x:Key="DialogButtonStyle" TargetType="Button">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type Button}">
        <Grid>
          <Ellipse Fill="{TemplateBinding Background}"
            Stroke="{TemplateBinding BorderBrush}"/>
              <ContentPresenter                HorizontalAlignment="Center"
              VerticalAlignment="Center"/>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

I aplicaríem la plantilla al botó així:

```
<Button Content="Button" HorizontalAlignment="Left" Margin="64,0,0,0"
VerticalAlignment="Center" Height="62" Width="188" Style="{StaticResource
DialogButtonStyle}" />
```

El resultat és el següent (botó normal i botó amb template):



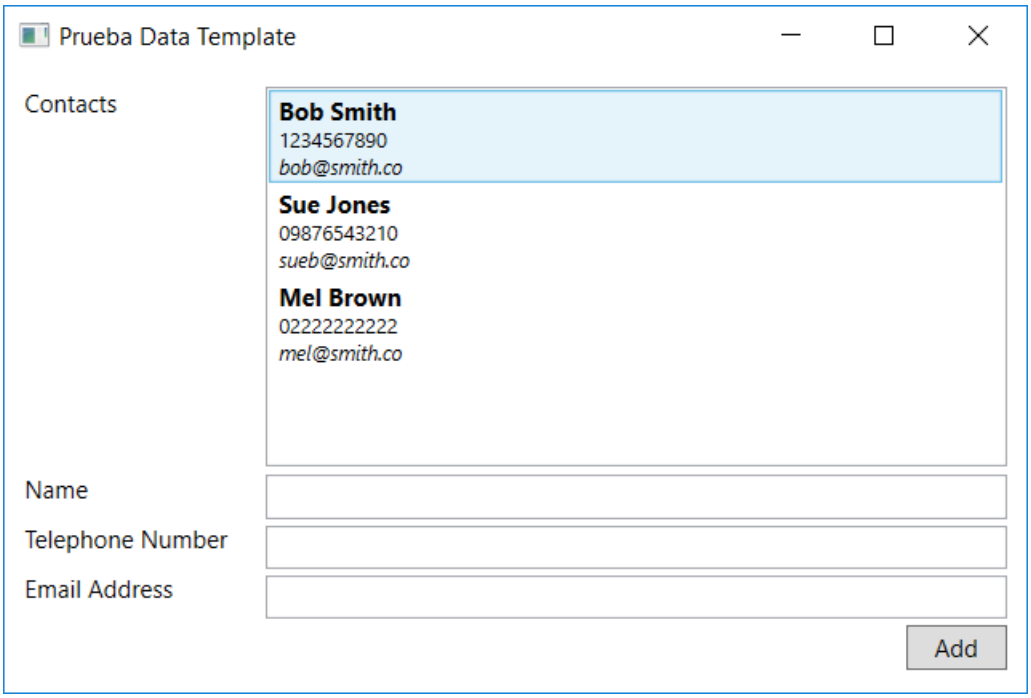
Un altre exemple seria el canviar el fons d'un botó en passar el ratolí per damunt:

```
<Style TargetType="{x:Type Button}" x:Key="sbtnNormal">
    <Setter Property="Background" Value="LightGray"/>
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="{x:Type Button}">
          <Border Background="{TemplateBinding Background}">
            <ContentPresenter HorizontalAlignment="Center”      VerticalAlignment="Center"/>
          </Border>
        </ControlTemplate>
      </Setter.Value>
      </Setter>
      <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter Property="Background" Value="DarkGray"/>
        </Trigger>
      </Style.Triggers>
</Style>
```

7.2. **Data Template.**

Es tracta d'un concepte similar a l'anterior, no obstant això està més orientat a la manera de visualitzar els ítems de dades de controls tipus **ListBox, ComboBox o ListView**.

Vegem a continuació un exemple per a veure una millor presentació dels elements d'una llista.



Volem mostrar les dades dels nostres contactes de l'anterior forma, per a això haurem de crear una plantilla de dades com la següent:

```
<ListBox x:Name="listaLibros">
          <ListBox.ItemTemplate>
              <DataTemplate>
                 <StackPanel>
                   <TextBlock FontWeight="Bold" FontSize="12" Text="{Binding Titol}"/>
                   <TextBlock FontSize="10" Text="{Binding Autor}"/>
                   <TextBlock FontStyle="Italic" FontSize="10" Text="{Binding Editorial}"/>
                 </StackPanel>
              </DataTemplate>
          </ListBox.ItemTemplate>

</ListBox>
```

Tenim una classe definida així:

```
public class libro
    {
        public string Titol { get; set; }
        public string Autor { get; set; }
        public string Editorial { get; set; }

    }
```

I carreguem la llista amb objectes de la classe:

```
InitializeComponent();

listaLibros.ItemsSource = new[] {
                new libro { Titol = "Les meravelles de WPF", Autor = "Pere",
Editorial="Anaya" },
                new libro { Titol = "Programant amb WPF", Autor = "Paco",
Editorial="Paraninfo" },
                new libro { Titol = "Visual Studio 2022", Autor = "Douglas Adams",
Editorial="McGraw Hill" }
            };
```

També podem introduir **Triggers** dins de les plantilles de dades.

```
<ListBox x:Name="listaLibros">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <Border x:Name="bdr" BorderBrush="Blue" BorderThickness="1">
                        <StackPanel>
                            <TextBlock x:Name="txt" FontWeight="Bold" FontSize="12"
Text="{Binding Titol}"/>
                            <TextBlock FontSize="10" Text="{Binding Autor}"/>
                            <TextBlock FontStyle="Italic" FontSize="10" Text="{Binding
Editorial}"/>
                        </StackPanel>
                    </Border>
                    <DataTemplate.Triggers>
                        <Trigger SourceName="bdr" Property="IsMouseOver" Value="True">
                            <Setter TargetName="bdr" Property="Background"
Value="LightGray"/>
                            <Setter TargetName="txt" Property="Foreground" Value="Red"/>
                        </Trigger>
                    </DataTemplate.Triggers>
                </DataTemplate>
            </ListBox.ItemTemplate>

        </ListBox>
```

En passar el ratolí per damunt el fons es torna gris i el nom es posa de color roig.

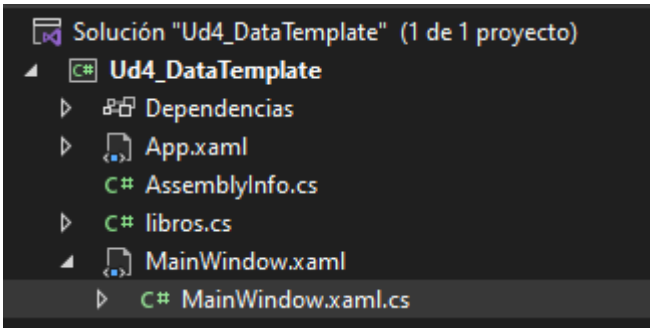
Altra opció seria crear una col·lecció d'objectes de tipus llibre i associar aquesta col·lecció al ListBox ....

ANNEX I: MODEL MVC EN LA PRÀCTICA

Ja portem uns quants projectes i sense adornar-nos ja hem treballat amb el model MVC als projectes. Recordes que el MVC (Model View Controller) separa l'aplicació en 3 parts diferenciades:

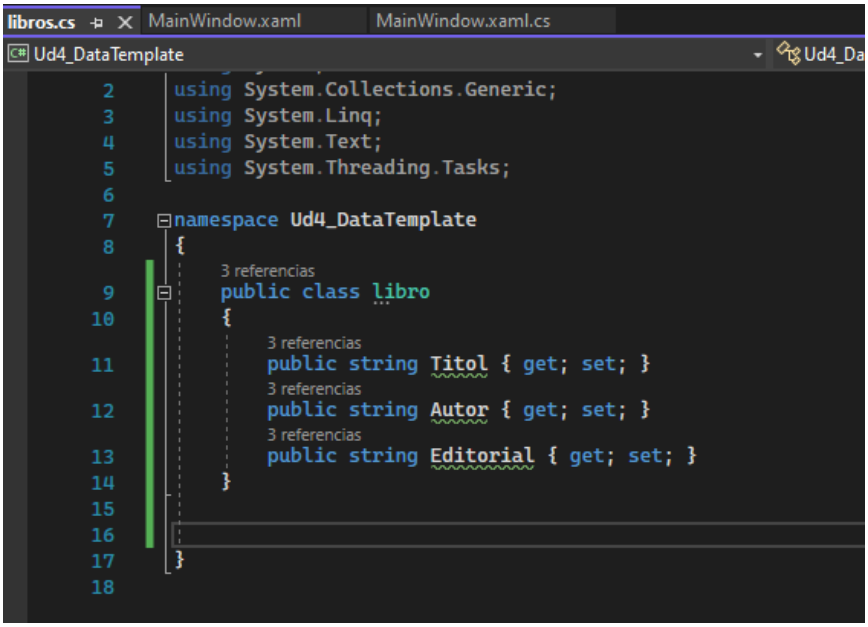
- **Model**: Es l'estructura de dades. Encara no hem accedit a dades, però si creem una classe per a crear objecte i mostrar-los, aquesta classe seria el model de dades.
- **Vista**: Son les finestres, és el frontend de l'aplicació
- **Controlador**: És la part en C# on controlem la lògica de l'aplicació, per exemple els esdeveniments.

Un exemple de projecte que segueix aquest model seria l'últim exemple d'aquesta unitat. Observem l'explorador de solucions del projecte:



I si observem el contingut de cada fitxer podem identificar que cada fitxer correspon a cada part del model:

MODEL (libros.cs)



VISTA (MainWindow.xaml)

```
<Window x:Class="Ud4_DataTemplate.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Ud4_DataTemplate"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="419">
    <Grid>
        <ListBox x:Name="listaLibros">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <Border x:Name="bdr" BorderBrush="Blue" BorderThickness="1">
                        <StackPanel>
                            <TextBlock x:Name="txt" FontWeight="Bold" FontSize="12" Text="{Binding Titol}"/>
                            <TextBlock FontSize="10" Text="{Binding Autor}"/>
                            <TextBlock FontStyle="Italic" FontSize="10" Text="{Binding Editorial}"/>
                        </StackPanel>
                    </Border>
                    <DataTemplate.Triggers>
                        <Trigger SourceName="bdr" Property="IsMouseOver" Value="True">
                            <Setter TargetName="bdr" Property="Background" Value="LightGray"/>
                            <Setter TargetName="txt" Property="Foreground" Value="Red"/>
                        </Trigger>
                    </DataTemplate.Triggers>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>

    </Grid>
</Window>
```

CONTROLADOR (MainWindow.cs)

MainWindow.xaml | MainWindow.xaml.cs

nplate | Ud4_DataTemplate.MainWindow

```
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Ud4_DataTemplate
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 referencias
    public partial class MainWindow : Window
    {
        0 referencias
        public MainWindow()
        {
            InitializeComponent();

            listaLibros.ItemsSource = new[] {
                new libro { Titol = "Les meravelles de WPF", Autor = "Pere", Editorial="Anaya" },
                new libro { Titol = "Programant amb WPF", Autor = "Paco", Editorial="Paraninfo" },
                new libro { Titol = "Visual Studio 2022", Autor = "Douglas Adams", Editorial="McGraw Hill" }
            };

        }
    }
}
```