Pagina.xaml

Tutto interno a griglia layout root.

Grid.RowDefinitions → definisco come sono le righe della griglia

Grid.ColumnDefinitions → definisco come sono le colonne

es. 2 righe: 1)con Height=”5\*” 2)con Height=”1\*” → la prima occuperà 5/6 della pagina in verticale, la seconda 1/6

Tutto può essere costituito al suo interno da una griglia

<StackPanel>

<Grid>

,,,

</StackPanel>

Dopo aver definito la Grid al suo interno dobbiamo dichiarare come sono le righe/colonne e inserire gli elementi

<Grid x:Name="..." >

<Grid.RowDefinitions>

,,,

</Grid.RowDefinitions>

<StackPanel x:Name="..." Grid.Row="0">

<TextBlock x:Name="..." Text="..." Style="{StaticResource PhoneTextNormalStyle}"/>

</StackPanel>

Proprietà bottoni:

<Button Name="..."

BorderThickness="0" → bordo

Background="Transparent" → colore di sfondo

Content="Gioca Ora!" → testo contenuto

Grid.Row="0" → in che riga deve posizionarsi

Grid.Column="0" → in che colonna deve posizionarsi

Click="Inizia\_Click" → che funzione lancia al click (definita su pagina.xaml.cs)

FontFamily="Comic Sans MS"

FontSize="{StaticResource PhoneFontSizeLarge}"

Tag=”ciao”/> 🡪 campo utile per assegnare un testo ad un bottone che non si vuole far vedere

Immagine di sfondo ad elementi (es. <StackPanel>,<Button>,<Grid>...):

<Elemento>

<Elemento.Background>

<Imagebrush ImageSource=”Images/img1.jpg” ></Imagebrush>

</Elemento.Background>

<Elemento>

**C#**

Campi con getter e setter di una classe per evitare accesso diretto a quelli privati

Public class Persona{

Private nome;

public Persona(){…}

Public string Nome{

Get{

Return nome;

}

Set{

If(value!=””)

nome=value;

}

}

Per avere il campo -> Persona p=new Persona(…); string nome=p.Nome; (GET)

p.Nome=”pippo”; (SET)

Liste:

List<Persona> persone=new List<Persona>();

persona.Add(new Persona(…”));

Persona p=persone[0];

…

Conversione:

string n=”1”;

int i=Convert.toInt32(n);

string s=i.toString();

MessageBox:

MessageBox.Show(“ciao”);

MessageBoxResult m = MessageBox.Show("Sei sicuro?", "Esci dal gioco", MessageBoxButton.OKCancel);

if (m == MessageBoxResult.OK)

{

…

}

else …

Navigazione da una pagina all’altra:

NavigationService.Navigate(new Uri("nuova\_pagina.xaml", UriKind.Relative));

Passaggio parametri da una pagina ad un’altra:

es. clicco bottone e passo a pagina successiva parametri

pagina1.xaml.cs

private void GoToGame(object sender, RoutedEventArgs e)

{

int liv = (int)((Button)sender).Tag;

string uri = "/Game.xaml?id=" + liv.ToString();

NavigationService.Navigate(new Uri(uri, UriKind.Relative));

}

Pagina2.xaml.cs

protected override void OnNavigatedTo(NavigationEventArgs e)

{

base.OnNavigatedTo(e);

string liv = string.Empty;

/// IF: riesco a prendere il livello sul quale sto navigando

if (NavigationContext.QueryString.TryGetValue("id", out liv))

{

/// parametro messo in liv

}

}

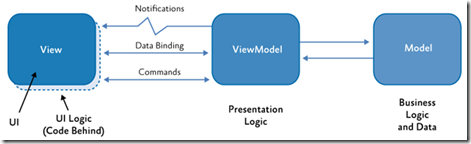
Data Binding:

**Pattern Model View ViewModel:**

Model = classi che eseguono il vero funzionamento

View = grafica (pagina.xaml+pagina.xaml.cs)

ViewModel = classi che collegano la view al model



Esempio:

Ho una classe Persona con un nome e un età (Model)

Questa persona deve potercomunicare i suoi cambiamenti di stato deve quindi implementare l’interfaccia INotifyPropertyChanged)

Creo una cartella Model e all’interno una nuova classe -> Person.cs

I campi di cui mi interessa il cambiamento devono avere setter e getter; nel setter una volta cambiato il campo privato, devo comunicare che il valore di quel campo è cambiato (ReisePropertyChanged(“nome\_campo”))

MODEL

public class Person : INotifyPropertyChanged

{

    private string name;

    private int age;

    public string Name

    {

        get

        {

            return name;

        }

        set

        {

            if (this.name != value)

            {

                this.name = value;

                this.RaisePropertyChanged("Name");

            }

        }

    }

    public int Age

    {

        get

        {

            return this.age;

        }

        set

        {

            if (this.age != value)

            {

                this.age = value;

                this.RaisePropertyChanged("Age");

            }

        }

    }

    public event PropertyChangedEventHandler PropertyChanged;

    private void RaisePropertyChanged(string propertyName)

    {

        PropertyChangedEventHandler handler = this.PropertyChanged;

        if (handler != null)

        {

            handler(this, new PropertyChangedEventArgs(propertyName));

        }

    }

}

VIEWMODEL

Creo una cartella ViewModel e all’interno una nuova classe PersonViewModel.cs

Anche questa classe deve poter notificare i cambiamenti di stato (all’interfaccia grafica) quindi deve implementare l’interfaccia INotifyPropertyChanged.

Se ho una lista di oggetti da osservare e presentare (non uno solo)devo avere tra i campi un ObservableCollection<Oggetto> lista ,la quale sarà passata come ItemSource ad una lista poi nella View tramite Lista(campo con setter e getter di lista)

public class PersonViewModel : INotifyPropertyChanged

{

    private string name;

    private int age;

    private ObservableCollection<Person> personDataSource;

    private ICommand loadDataCommand;

    private ICommand saveChangesCommand;

    public PersonViewModel()

    {

        this.loadDataCommand = new DelegateCommand(this.LoadDataAction);

        this.saveChangesCommand = new DelegateCommand(this.SaveChangesAction);

    }

    private void LoadDataAction(object p)

    {

        this.DataSource.Add(new Person() { Name = "John", Age = 32 });

        this.DataSource.Add(new Person() { Name = "Kate", Age = 27 });

        this.DataSource.Add(new Person() { Name = "Sam", Age = 30 });

    }

    private void SaveChangesAction(object p)

    {

        if (this.SelectedPerson != null)

        {

            this.SelectedPerson.Name = this.name;

            this.SelectedPerson.Age = this.age;

        }

    }

    public ICommand LoadDataCommand

    {

        get

        {

            return this.loadDataCommand;

        }

    }

    public ICommand SaveChangesCommand

    {

        get

        {

            return this.saveChangesCommand;

        }

    }

    public ObservableCollection<Person> DataSource

    {

        get

        {

            if (this.personDataSource == null)

            {

                this.personDataSource = new ObservableCollection<Person>();

            }

            return this.personDataSource;

        }

    }

    public string SelectedName

    {

        get

        {

            if (this.SelectedPerson != null)

            {

                return this.SelectedPerson.Name;

            }

            return string.Empty;

        }

        set

        {

            this.name = value;

        }

    }

    public int SelectedAge

    {

        get

        {

            if (this.SelectedPerson != null)

            {

                return this.SelectedPerson.Age;

            }

            return 0;

        }

        set

        {

            this.age = value;

        }

    }

    private Person selectedPerson;

    public Person SelectedPerson

    {

        get

        {

            return this.selectedPerson;

        }

        set

        {

            if (this.selectedPerson != value)

            {

                this.selectedPerson = value;

                if (this.selectedPerson != null)

                {

                    this.name = this.selectedPerson.Name;

                    this.age = this.selectedPerson.Age;

                }

                this.RaisePropertyChanged("SelectedName");

                this.RaisePropertyChanged("SelectedAge");

            }

        }

    }

    public event PropertyChangedEventHandler PropertyChanged;

    private void RaisePropertyChanged(string propertyName)

    {

        PropertyChangedEventHandler handler = this.PropertyChanged;

        if (handler != null)

        {

            handler(this, new PropertyChangedEventArgs(propertyName));

        }

    }

}

VIEW

Page.xaml.cs

Fisso il datacontex della pagina corrente ad un oggetto viewmodel (dico che questa view farà riferimento alla VM per reperire dati tramite Binding.

public Page()

{

    InitializeComponent();

    this.DataContext = new PersonViewModel();

}

Page.xaml

Costruisco una listbox e fisso l’ItemSource al campo con setter e getter DataSource (Lista)

Ogni elemento della lista seguirà un itemtemplate/datatemplate per essere costruito, verranno costruiti tanti elementi di lista quanti sono gli oggetti contenuti nella lista (DataSource)

Ogni elemento di lista costruito sarà quindi associato ad un oggetto della lista da qui tramite il {Binding Proprietà} andrà a popolare i pripri elementi, dove Proprietà è un campo con setter e getter dell’oggetto a cui è associato

SelectedItem=”{Binding SelectedPerson, Mode=TwoWay”} dice che nel momento in cui verrà selezionato un elemento della lista, verrà comunicato alla classeVM e viene applicato il set sul campo SelectedPerson (con getter e setter) in cui come value viene passato l’oggetto associato a quell’elemento di lista

<StackPanel x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

    <Button Content="LoadData" Command="{Binding LoadDataCommand}" />

    <ListBox ItemsSource="{Binding DataSource}" SelectedItem="{Binding SelectedPerson, Mode=TwoWay}" Height="100">

        <ListBox.ItemTemplate>

            <DataTemplate>

                <StackPanel Orientation="Horizontal">

                    <TextBlock Text="Name:"/>

                    <TextBlock Text="{Binding Name}" />

                    <TextBlock Text="Age:" Margin="10,0,0,0"/>

                    <TextBlock Text="{Binding Age}" />

                </StackPanel>

            </DataTemplate>

        </ListBox.ItemTemplate>

    </ListBox>

    <TextBlock Text="Name:"/>

    <TextBox Text="{Binding SelectedName, Mode=TwoWay}" />

    <TextBlock Text="Age:"/>

    <TextBox Text="{Binding SelectedAge, Mode=TwoWay}" />

    <Button Content="Save Changes" Command="{Binding SaveChangesCommand}" />

</StackPanel>

Come gestire la pressione del BackButton

Page.xaml.cs

Faccio override del metodo OnBackKeyPress, nell’esempio chiedo all’utente se è sicuro (MessageBox), se no devo cancellare il CancelEventArgs in modo da bloccare l’effetto delle pressione del backbutton, se si navigo verso una pagina che dico io (NavigationService.Navigate) e applico il refresh alla nuova pagina (nell’uri inserisco Refresh=True). Quest’ultima cosa può essere utile nel caso in cui torno alla pagina precedente a quella corrente e questa presenta dei dati dinamici che potrebbero essere cambiati, solitamente se si torna indietro viene riproposta una copia della pagina visitata in precedenza. Nel caso in cui

protected override void OnBackKeyPress(System.ComponentModel.CancelEventArgs e)

{

MessageBoxResult m=MessageBox.Show("Sei sicuro?", "Esci dal livello", MessageBoxButton.OKCancel);

if(m==MessageBoxResult.OK)

{

NavigationService.Navigate(new Uri("/PagLivelli.xaml?Refresh=true", UriKind.Relative));

}

else e.Cancel = true;}