Module

MVVM Pattern

Table of contents

- Introduction
- *MVVM Pattern*
- *Model - Services*
- View
- ViewModel
- DataBinding
- Commands
- Sample
- *MVVM Light Toolkit*
- Appendix

Introduction

- One possibility
  - XAML

contenu   arrière fond

```xml
<Button Content ="Go" Foreground="Green" Background="YellowGreen" Click="Button_Click" Margin="127,183,0,400" />
```

  - Code-behind

à mettre dans le fichier .cs

```csharp
private void Button_Click(object sender, RoutedEventArgs e)
{
    //TODO
}
```

Introduction

- Disadvantages

    - Very closed collaboration between the developer and the designer

        → evolution during updates

    - Not easy to write tests (simulation inputs from the UI)

    - Code that can become large

    - No abstraction between the graphical representation in XAML and the data in the associated code and the business layer

MVVM Pattern

- Architecture Pattern
  - M(odel)
  - V(iew)
  - V(iew)M(odel)

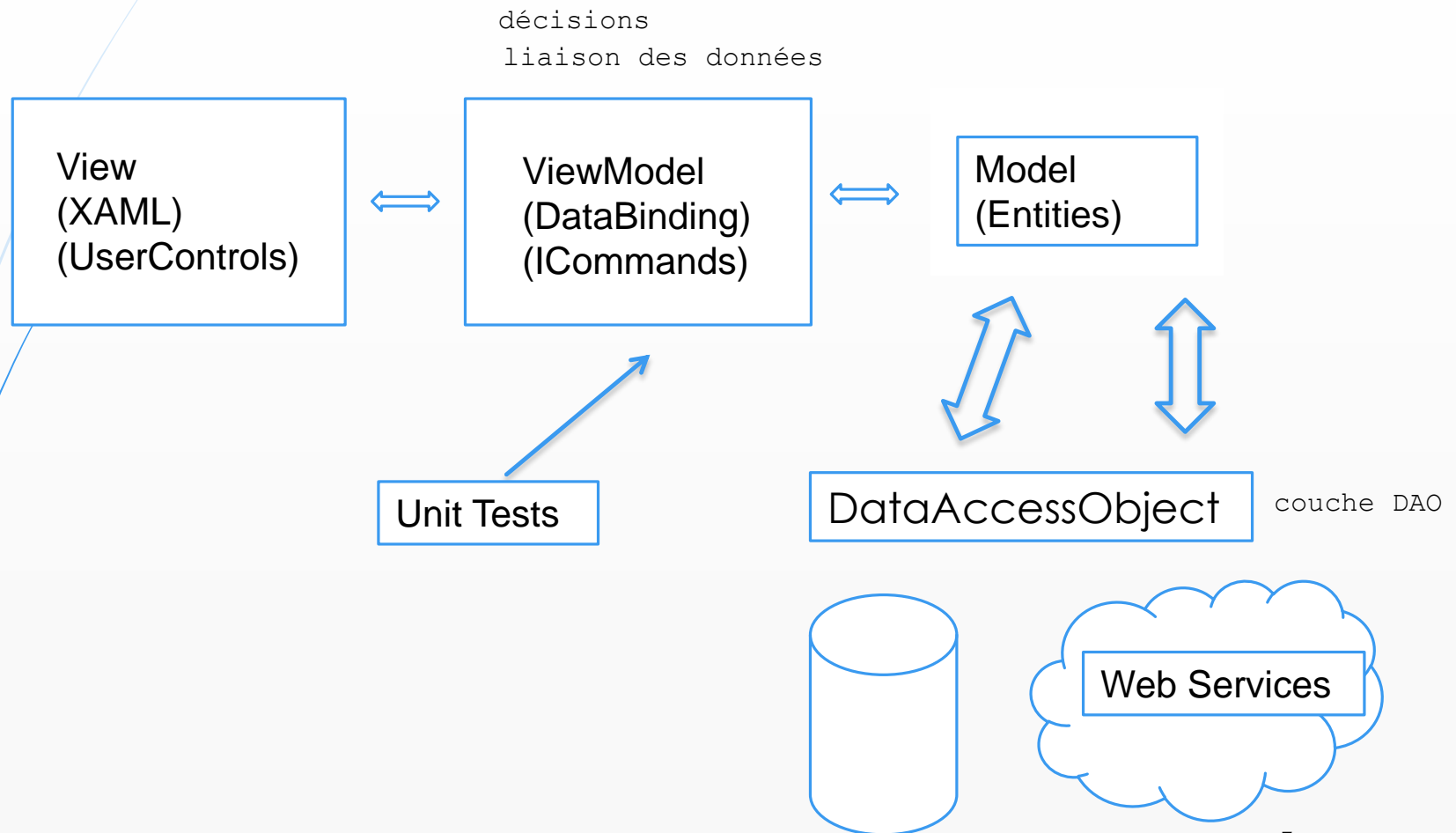   To isolate the domain logic from the user interface logic
- Benefits
  - Maintainability
  - Scalability

MVVM Pattern

- Mentioned for the first time in 2005 by J. Gossman, Microsoft developer architect
  - Engine binding (DataBinding)
  - Control system (Commands)
- Pros
  - Long-term project
  - Unit Testing
  - Segregation between the designer and the developer
- Cons
  - Creation of models or prototypes
  - Demo application

MVVM Pattern

```
décisions
liaison des données
```

View
(XAML)
(UserControls)
⟺
ViewModel
(DataBinding)
(ICommands)
⟺
Model
(Entities)

Unit Tests

DataAccessObject   couche DAO

Web Services

MVVM Pattern

- Two approaches
  - **View-first**
    - Far simpler to implement page navigations
  - Viewmodel-first
    - The viewmodel creates the view
    - Potentially offers complete independance from the UI allowing an app to be executed without a user interface

Model - Services

- Specifications
  - Entities of the application domain and their functionalities
    - Classes and / or libraries
  - Know everything about themselves and nothing about the other layers
- Implementation – Data Access Object (DAO) `pas d'interfaçage graphiq` `ex : console.log, etc`
  - Set of objects and methods that allow the manipulation of those objects

Model - Services

- Responsibilities
  - Set the data format
  - Define their access mode (read, edit, delete, create)
  - Ensure business rules through services
  - Perhaps ensure data integrity (validation)
  - Notify the ViewModel of changes in data

View

- Code corresponding to graphical interfaces
- Interactions with peripherals (keyboard / mouse) managed by the View (UserControls)
- Display data
  - Data Binding

ViewModel

- ViewModel (VM)
  - Abstract representation of the view
  - Manipulation of the data model
  - Weak coupling with view

ViewModel

- ► Role concerning the data
  - ►No knowledge about the data display
    - ►More freedom for the view to draw and to decide how data must appear
      - ►Example: a Boolean in VM can become a checkbox in View, a picture, ...
  - ►Load of the data necessary for the view
  - ►Preparation of the data : sorting, grouping, filtering
    - ►Example: a list of people separated in two lists for the view (woman – man) `filtre, sort, c'est le MV qui s'en occupe`

ViewModel

- Role concerning the data (…)
  - Possible choice to call a service according to the state
    - Example: calling the data validation before saving

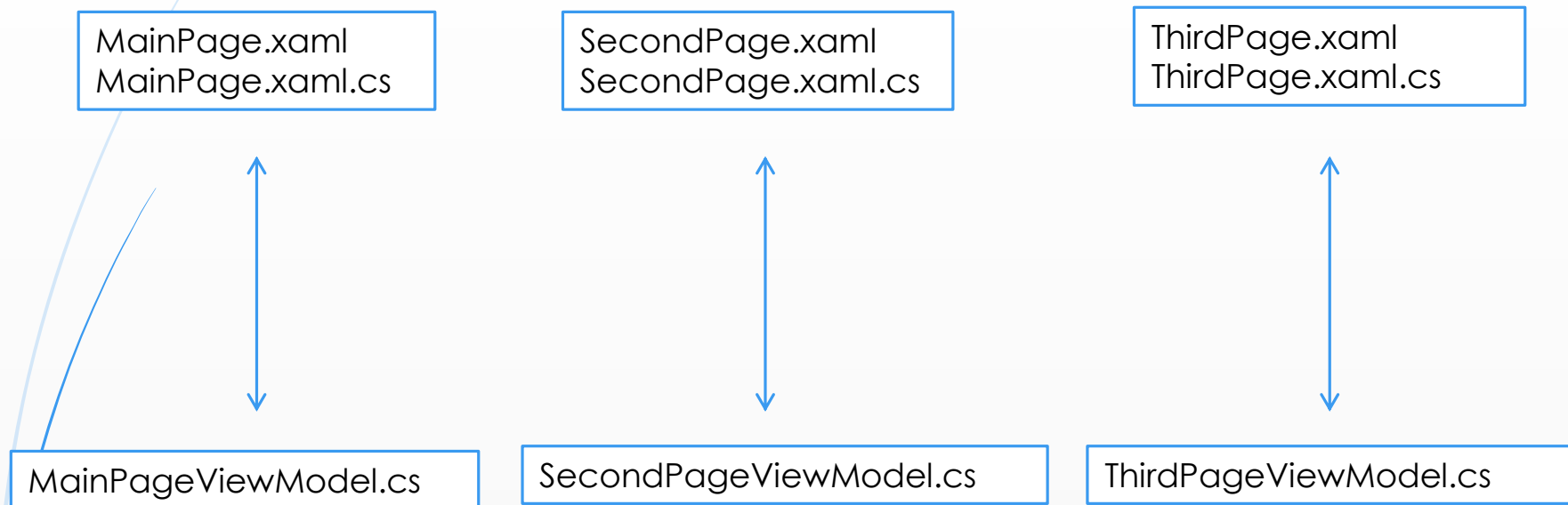 → Regulator between the View and the data

ViewModel

- ViewModel does not need the View
  - For maximum detachment, the programmer uses interfaces
  - The VM interface exposes itself the different data and the means to communicate with the layer
- So
  - Possibility of unit tests in view of the standardisation of the classes in VM
  - Possible TDD (Test Driven Development)
  - Reusable VM in several projects

ViewModel

un view model par page

- Architecture

| MainPage.xaml
MainPage.xaml.cs | SecondPage.xaml
SecondPage.xaml.cs | ThirdPage.xaml
ThirdPage.xaml.cs |

↕ ↕ ↕

| MainPageViewModel.cs | SecondPageViewModel.cs | ThirdPageViewModel.cs |

- ??? IViewModel.cs interface in the VM

Data Binding

- Way for applications to present and interact with data
    - One UI control can be bound to a property of a ViewModel object
- Basic concepts
    - Source
    - Target
    - Converter

sur une page, une liste de joueur, cette liste là doit être liée avec des données
d'une autre page (d'ailleurs), synchronisation des données (double ou simple sens)

Data Binding
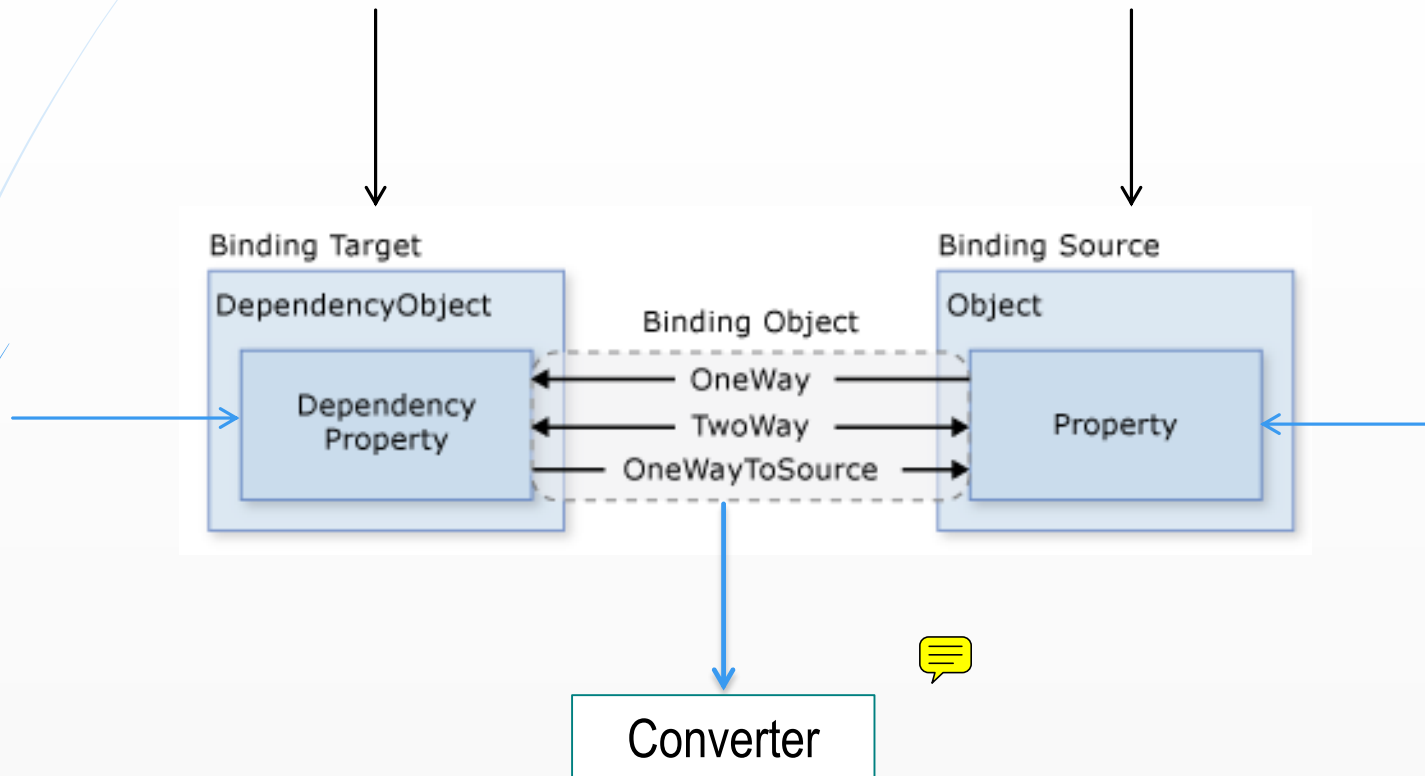
Data Binding

- Example   `<TextBox   Text = " {Binding Name} " …>`

  - Source = Student object
  - Property : **Name**
  - Target : TextBox
  - Dependancy property : Text
- Ability to specify
  - When
  - How
  - Why

  During the transfer from the source to the target

Data Binding

- Directions
  - OneTime
    - The value of the control is set once to the data value
      - Only when launching the application or when changing DataContext
    - Any subsequent changes are ignored
  - OneWayToSource
    - Source properties are updated whenever the control properties change

Data Binding

- Directions (…)
  - OneWay
    - Changes in the data object are synchronized to the control properties
    - Changes through the control are not synchronized back to the data source object
  - TwoWay
    - Changes in the data object are synchronized to the control property and vice-versa
  - Default (some controls have a default value)

Data Binding

- Property Change Notification
  - For OneWay or TwoWay binding, ViewModel classes must implement *INotifyPropertyChanged* interface
    - Interface that allows each property to notify the UI when its value changes
      - Allows a source object (e.g, from the viewmodel) to signal to a target FrameworkElement that a value needs updating in the UI
      - To generate an event when a property of the object has changed
      - The WPF binding engine knows that there is a change

Data Binding

- E.g, *myTextBox* bound to *myProperty* of *MyViewModelClass*
  - In *MyPage.xaml.cs* *(constructeur)*

```
public MyPage( )
{
        InitializeComponent();
        DataContext = new MyViewModelClass();

}
```

  - In *MyPage.xaml*

```
<TextBox  x:Name = "myTextBox"   Text = "{ Binding  MyProperty,  Mode = TwoWay }"   … />
```

Data Binding

- In *MyViewModelClass.cs,*

```csharp
public class MyViewModelClass : INotifyPropertyChanged
{
    private String _myProperty;

    public String MyProperty
    {   get { return _myProperty; }
        set
        { OnNotifyPropertyChanged("MyProperty"); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    private void OnNotifyPropertyChanged(string propertyName)
    {
            if( PropertyChanged != null)
                PropertyChanged((this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Event when the value of the attribute is changed

Data Binding

- In the MainViewModel.cs (another version)

```csharp
private Person person;
public Person Person...

private String name;
public String Name
{
    get { return name; }
    set { NotifyPropertyChanged(ref this.name, value); }
}

private bool NotifyPropertyChanged<T>(ref T variable, T valeur, [CallerMemberName] string nomPropriete = null)
{
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        RaisePropertyChanged(nomPropriete);
        return true;
}
```
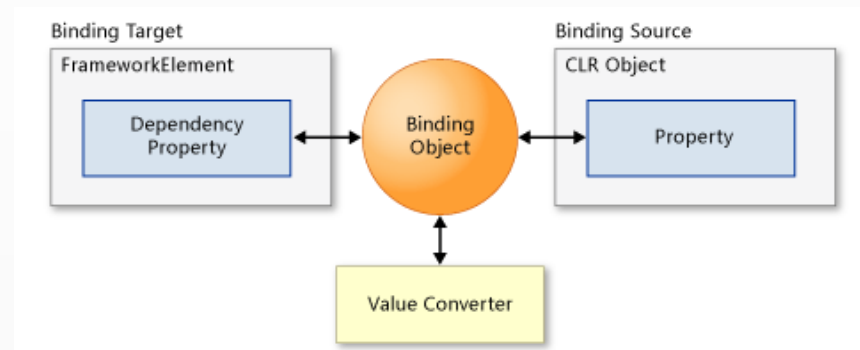
Data Binding

- A UI Collection can be bound to a collection of ViewModel objects

  - Set the ItemsSource property to a collection of data objects

- Two Conditions

  - The ViewModel class must implement *INotifyPropertyChanged* interface

    - To generate an event when

      - An item is added to the collection

      - An item is removed from the collection

      - A property of an item has changed

Data Binding

- The collection must be an object of *ObservableCollection*

  - Only for OneWay and TwoWay

  - Dynamic data collection that product notifications when items are added, deleted, …

    - During the instanciation of the class ViewModel, the list of objects managed by the model is browsed to fill the collection

    - Additional items or deleted items in the model are managed by the class VM during the last filling

Data Binding

- Connection between two different types of data
  - In the ViewModel layer or in another folder, create classes that implement one of the interfaces
    - IValueConverter
      - 2 méthods
        - Convert
        - ConvertBack
    - IMultiValueConverter
      - Objects Array as input
      - A converted object as output

Data Binding

- E.g.,

```
public class WeatherDescriptionToImageValueConverter:IValueConverter
  {
     public object Convert(object value, Type targetType, object parameter, string language)
     {
        var forecast = (string)value;
        if (forecast.Contains("nuageux"))
           return new BitmapImage(new Uri("ms-appx:/Images/cloudy.png"));
        else
           return new BitmapImage(new Uri("ms-appx:/Images/sunny.png"));
     }


     public object ConvertBack(object value, Type targetType, object parameter, string language)
     {
        throw new NotImplementedException();
     }
  }
```

Commands

▸ "Unlike a simple event handler, the (Command) commands separate the semantics and calling for action, logic. This allows different sources of the same call control logic and customizes it with respect to different targets"

▸ Controls

<Button … Command = "{ Binding …}" …>

Object, instance of a class that implements the ICommand interface which is placed in the ViewModel
Once the button is pressed by the user, the associated function is called

Commands

- Methods of the ICommand Interface
  - Execute
    - Defines the method to be called when the command is invoked
  - CanExecute
    - Defines the method that determines whether the command can execute in its current state
  - CanExecuteChanged
    - Event that occurs when changes occur that affect whether or not the command should execute

Commands

▶ E.g.,

```xml
<Button Content="LoadData" Command="{Binding LoadDataCommand}" />
<ListBox ItemsSource="{Binding DataSource}"<div class="wlWriterEdi
    <ListBox.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Name}" />
        </DataTemplate>
    </ListBox.ItemTemplate>
```

```csharp
public MainPage()
{
    InitializeComponent();

    // simple way to bind the view to the view model
    this.DataContext = new PersonViewModel();
}
```

Commands

- E.g.,

```
public class MainPageViewModel :
{
    …
    private Icommand _loadDataCommand;
    public Icommand LoadDataCommand
    {
        get {
            if ( _loadDataCommand == null )
            {
                        _loadDataCommand = new RelayCommand ( () => LoadData());
            }
            return _loadDataCommand ;
    }

    private void LoadData()
    {
        DataSource =  …;
    }
}
```

ViewModel Architecture

- One class for one class of the View
- ViewModelBase Class
  - Navigation support
  - Error validation
  - State preservation
  - Property change notification

Commands

```
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
        OnPropertyChanged(new PropertyChangedEventArgs(propertyName));
    }

    protected virtual void OnPropertyChanged(PropertyChangedEventArgs args)
    {
        var handler = PropertyChanged;
        if (handler != null)
            handler(this, args);
    }
}
```
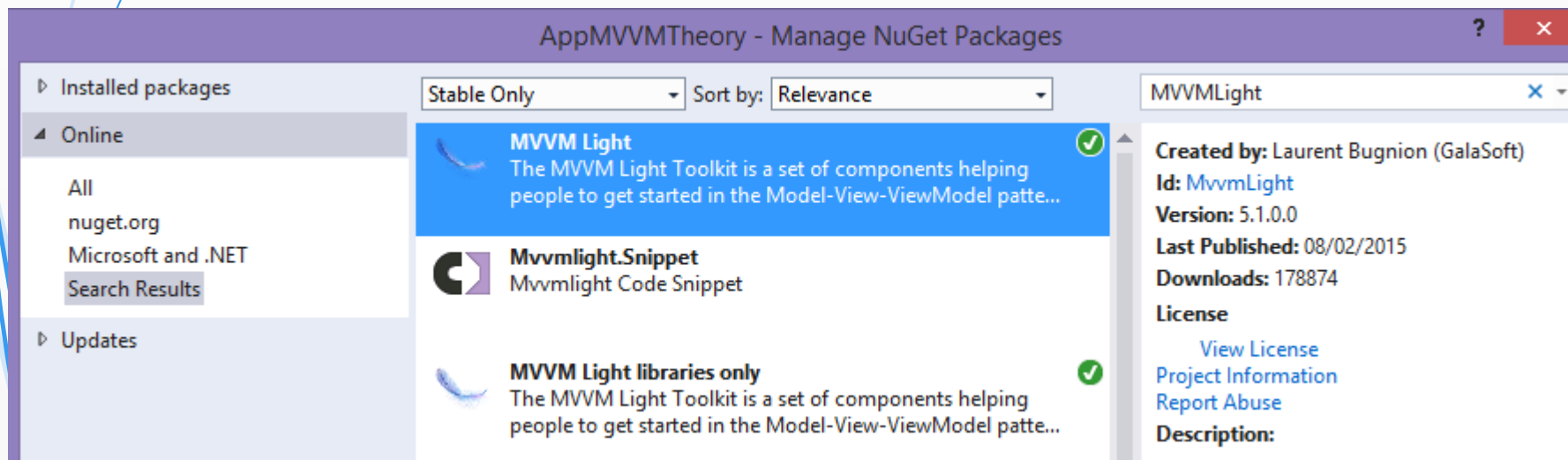
MVVM Light toolkit

- Toolkit
  - That accelerate the creation and development of MVVM applications in WPF, Windows Phone, Universal app
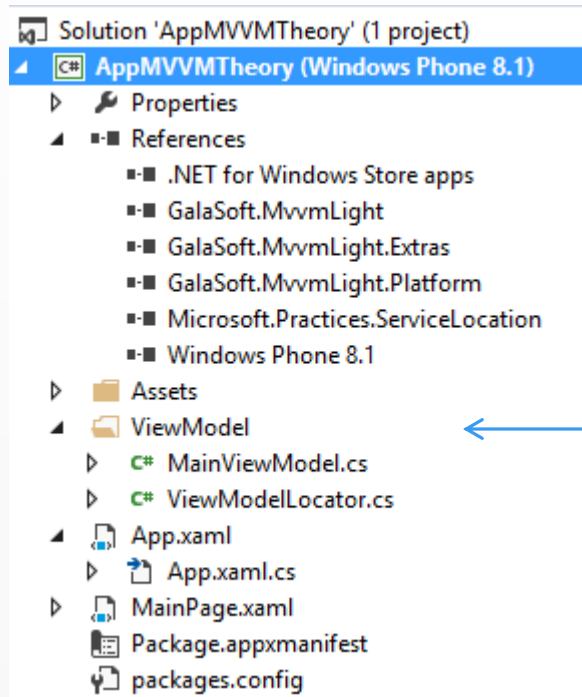 - Solution, one project, multiple folders

MVVM Light Toolkit

- To create a project whith MVVM Light Toolkit
  - Create the project
  - Manage NuGet Packages
  - Online

MVVM Light Toolkit

➡ In the project :

MVVM Light Toolkit

- In the App.xaml,

```xml
<Application x:Class="AppMVVM.App" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
  <Application.Resources>
    <vm:ViewModelLocator x:Key="Locator" xmlns:vm="using:AppMVVM.ViewModel" />
  </Application.Resources>
</Application>
```

MVVM Light Toolkit

In the ViewModelLocator.cs,

```csharp
public class ViewModelLocator
{
    /// <summary>
    /// Initializes a new instance of the ViewModelLocator class.
    /// </summary>
    0 references
    public ViewModelLocator()
    {
        ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);
        SimpleIoc.Default.Register<MainViewModel>();
    }

    0 references
    public MainViewModel Main
    {
        get
        {
            return ServiceLocator.Current.GetInstance<MainViewModel>();
        }
    }
}
```

MVVM Light Toolkit

```xml
<Page
    x:Class="AppMVVMTheory.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AppMVVMTheory"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    DataContext="{Binding Source={StaticResource Locator}, Path=Main}">
```

➡ In MainViewModel,

```csharp
public class MainViewModel : ViewModelBase...
```

MVVM Light Toolkit

- E.g, a UI Collection can be bound to a collection of ViewModel objects

  - Set the ItemsSource property to a collection of data objects

```
<Page
   x:Class="FlickClient.MainPage"
   ….
   DataContext="{Binding Source={StaticResource Locator}, Path=Main}"
   Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

   <Grid>
     <ListView ItemsSource="{Binding Forecast}" ItemTemplate="{StaticResource WeatherTemplate1}">
     </ListView>
   </Grid>
</Page>
```

MVVM Light Toolkit

```
public const string ForecastPropertyName = "Forecast";
private ObservableCollection<WeatherForecast> _forecast = null;

public ObservableCollection<WeatherForecast> Forecast
{
      get
      {
         return _forecast;
      }

      set
      {
         if (_forecast == value)
         {
            return;
         }
         _forecast = value;
         RaisePropertyChanged(ForecastPropertyName);
      }
}
```

Appendix

- Others frameworks
- Organization
  - MVVM Light : One solution, one project, many folders
  - Other : One solution, many projects