طريقك إلى MVVM

بسم الله الرحمن الرحيم

سنقوم هنا بعرض طريق مختصر وصغير لتقنية MVVM الرائعة، لذلك لن نطيل في الشرح النظري لكي لا نفقد متعة الجزء العملي والتطبيقي، بالتالي ستكون المقدمة مختصرة قدر الإمكان، وسنترك التوضيح للجزء العملي. وبعد أن نسلك هذا الطريق الصغير، ستكون لدينا القدرة للخوض في الطرقات الأخرى المعقدة والممتعة بنفس الوقت.

ما هي تقنية MVVM؟

تقنية من ميكروسوفت تستخدم لبناء التطبيقات ذات الواجهات (Presentation Application)، تعتمد وبشكل كبير على WPF, Silverlight، حيث أنها تستهلك معظم خواص تلك التقنيات مثل (Data Binding, Command, Styles, Templates…). اشتق الإسم من (Model-View-ViewModel)، بالتالي فهي مكونة من ثلاث أقسام رئيسية، سنوجزها بإختصار.

Model:

في عالم البرمجة مصطح مودل مقابل لمصطلح بيانات (Data)، بالتالي ضمن تطبيق MVVM فإن Model يمثل البيانات التي سنتعامل معها، طبعاً ليس كل أنواع البيانات. مثلاً لو كان لدينا تطبيق قواعد بيانات، فإن الكلاسات (Classes) التي تعكس حقول الجداول، هي ما يطلق عليها Models. والأهم من هذا أن تكون هذه المودلز مستقلة عن التطبيق ولا تأخذ أي اعتبار للأجزاء الأخرى، بمعنى لا يجب أن ترتبط أو تؤشر على جزء أخر من التطبيق، على العكس الأجزاء الأخرى ترتبط معها وتؤشر عليها.

View:

أيضاً مصطلح View مقابل لمصطلح الواجهات User Interfaces، لكن ضمن MVVM ليست هي تماما الواجهات إذا يمكن أن تمثل View وحدة، لقطعة صغيرة من واجهة معينة، بالتالي ربما تحوي الواجهة الواحدة على View أو أكثر. الجوهو الأساسي في بناءها هو الـ Data Binding، فضمن MVVM لا يوجد كود للواجهة(Code Behind). بالتالي ايضاً لا فهي مستقلة عن أجزاء التطبيق الأخرى.

ViewModel:

ألية الربط بين الـ Model والـ View، وهي ما تجعل كل منها مستقل عن التطبيق، حيث تقوم الـ ViewModelبتنظيم خواص (Properties) الخاصة بمودل معين، وتظهرها ل View بشكل تستطيع عرض تلك البيانات بشكل User Controls. كما توفر ألية لتنفيذ العمليات والأحداث على تلك البيانات، أقرب مثال هو تنفيذ حدث زر معين. حيث تقوم الـ ViewModel بعرض خاصية من نوع ICommand مسؤولة عن تنفيذ حدث معين، بينما يقصر عمل الـ View على ربط ذلك الزر بهذه الخاصية، عن طريق DataBinding كما تحدثنا.

متطلبات العمل مع MVVM؟

بالطبع لا يمكن البدء بتعلم هذه التقنية الرائعة دون المرور والتعرف على مجموعة من الميزات الرائعة أيضاً ضمن بيئة العمل، بما أن بداية هذا الطريق ستكون سهلة، لذلك التعرف على هذه الميزات سيكون أسهل، حيث سنقدم مبادئ بسيطة عن كل ميزة، خلال التطبيق، وفي مراحل التطوير سنتعرف على الخواص الأخرى والطرق الأخرى في استخدام تلك الميزات، والتي أهمها:

1. Data Binding
2. Commands
3. Styles
4. Templates
5. Validation
6. Notifications
7. Converters

الآن دعونا نبدأ بالجزء العملي، وسنلاحظ خلال التطبيق فائدة تلك الميزات، حيث سنقوم بعرضها بأبسط طريقة كما تحدثنا، وعند الوصول لنقطة معينة، سنقوم بعرض أفضل طريقة لاستخدام ميزة معينة.

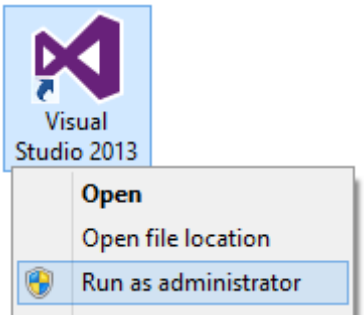
للحصول على فهم أفضل، قم بمتابعة وتطبيق الخطوات التالية خطوة بخطوة، بالطبع يوجد تطبيق هذا الدرس في المرفقات.

بناء نظام بيانات مصغر

الهدف من هذا المشروع الصغير، هو بناء تطبيق صغير يقوم بعرض مجموعة من الأشخاص People حيث كل شخص يمثل بالمودل Person، وبإمكاننا إضافة شخص جديد، تعديل أو حذف شخص سابق، بالطبع لن نعتمد على قواعد البيانات، لكي لا نضيف درجة تعقيد إضافية للمشروع (حيث أن استخدام قواعد البايانات هي هدفنا من هذه الدروس لاحقاً بإذن الله تعالى).

سيتم تنفيذ المشروع اعتماد على مفهوم التطوير السريع (Agile Methodology)، اي سنقوم ببناء تطبيق صغير، ثم سنقوم بتوسعته شيئاً فشيئاً، ليتسنى لنا فهم كل جزء بشكل أفضل، لذلك في النهاية سيكون لدينا عدة إصدارات من هذا العمل.

1. قم بتشغيل الـ Visual Studio بشكل مسؤول، وذلك بالضغط على أيقونته بالزر الأيمن واختيار خيار(Run As Administrator).



ثم قم بإنشاء مشروع جديد من نوع (WPF Application)، أعطيه إسم (Mvvm.Session01)، ولا تنسى أيضاً تسمية المشروع (Solution) بإسم مناسب مثل (MvvmWay).

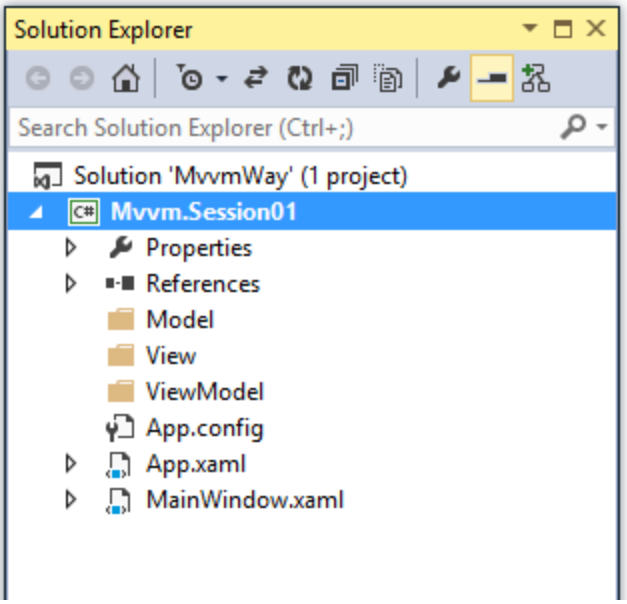
2. قم بإضافة ثلاث مجلدات إلى مشروعك الجديد بالإسماء التالية.

(1 Model

(2 View

(3 ViewModel

الآن يجب أن يكون لديك شكل المشروع بالشكل التالي:



3. الأن قم بإضافة كلاس جديد (New Class) إلى مجلد Model، سميه (Person.cs)، وقم بكتابته بالشكل التالي:

```
namespace Mvvm.Session01.Model
{
    public class Person
    {
        private int _id;
        public int Id
        {
            get { return _id; }
            set { _id = value; }
        }

        private string firstName;
        public string FirstName
        {
            get { return firstName; }
            set { firstName = value; }
        }

        private string lastName;
        public string LastName
        {
            get { return lastName; }
            set { lastName = value; }
        }

        public string FullName
        {
            get
            {
```

```
                return string.Format("{0} {1}", FirstName, LastName);
            }
        }
    }
}
```

4. قم بإضافة كلاس جديد تحت إسم (PersonViewModel.cs) ضمن مجلد ViewModel، قم بكتابته بالشكل التالي:

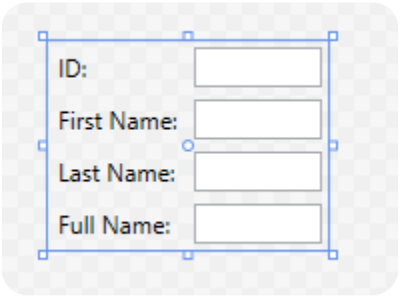
```
namespace Mvvm.Session01.ViewModel
{
    using Mvvm.Session01.Model;
    public class PersonViewModel
    {
        private Person model;

        public int Id
        {
            get { return model.Id; }
            set
            {
                model.Id = value;
            }
        }

        public string FirstName
        {
            get { return model.FirstName; }
            set
            {
                model.FirstName = value;
            }
        }

        public string LastName
        {
            get { return model.LastName; }
            set
            {
                model.LastName = value;
            }
        }
        public string FullName
        {
            get { return model.FullName; }
        }
        public PersonViewModel(Person person)
        {
            model = person;
        }

        public PersonViewModel()
            :this(new Person())
        { }
```

```
    }
}
```

5. الأن قم بإضافة عنصر من نوع User Control إلى مجلد View، وأعطه إسم Person.xaml، كما تحدثنا سنستخدم هذا الجزء لعرض البيانات للمستخدم، بالتالي يجب علينا الإهتمام بتنظيم وتركيب هذا الجزء، ففي النهاية المستخدم لن يرى سواه.

الآن قم بتحديث كود XAML ليصبح كالتالي:

```
<UserControl x:Class="Mvvm.Session01.View.Person"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <UniformGrid Columns="2">
        <Label Content="ID:"/>
        <TextBox Text="{Binding Id}"/>

        <Label Content="First Name:"/>
        <TextBox Text="{Binding FirstName}"/>

        <Label Content="Last Name:"/>
        <TextBox Text="{Binding LastName}"/>

        <Label Content="Full Name:"/>
        <TextBox Text="{Binding FullName, Mode=OneWay}"/>
    </UniformGrid>
</UserControl>
```



الآن قد يخطر لك تساؤل كيف سيتم ربط الأجزاء الثلاثة ببعضها البعض؟

كما لاحظنا فإن الـ ViewModel قامت بتعريف مؤشر على اوبجكت من نوع Person وهو ما ستستخدمه كـ Model لتنظيم البيانات، وأيضاً قامت بعمل تغليف لخواص ذلك المودل، حيث قمنا بتعريف Id, FirstName, LastName, FullName مرة أخرى عن طريق model. بالتالي يبقى لدينا أن نربط الـ

View بـِViewModel ، وعندها تتمكن الـ View من الدخول للخواص السابقة مباشرة (طبعاً هذا المفهوم هو أحد المفاهيم التي سنتوسع بشرحها لاحقاً، فهل من المعقول إعادة كتابة جميع الـ Properties ضمن الـ ViewModel؟ وسنعطي الحلول المتوفرة ووجهات النظر حولها، لكن الآن يكفينا النظرة الحالية).

أما عن كيفية ربط الـ View بـِ ViewModel فيتم عن طريق الخاصية DataContext الموجود ضمن FrameworkElement، وبما أن الـ View من نوع User Control والذي بدوره حفيد ل FrameworkElement ، بالتالي إسناد هذه الخاصة لأوبجكت جديد من نوع PersonViewModel، كفيل بتحقيق الهدف.

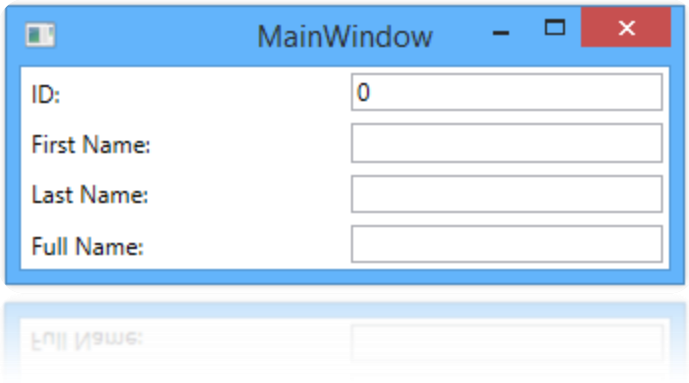
6. الآن ضمن الكود الخاص بـ View (Code Behind) نقوم بكتابة السطر التالي، بالطبع هذه الطريقة خاطئة نوعاً ما، فهناك عدة طريق لتحقيق هذا الهدف، ولكننا نكتفي بالأبسط والأوضح هنا:

```
namespace Mvvm.Session01.View
{
    using System.Windows.Controls;
    using Mvvm.Session01.ViewModel;
    public partial class Person : UserControl
    {
        public Person()
        {
            InitializeComponent();
            DataContext = new PersonViewModel();
        }
    }
}
```

7. الآن لجعل التطبيق قابل للعمل، يجب استضافة الـ View ضمن نافذة معينة، لذلك سنستخدم النافذة الأساسية MainWindow.xaml، حيث نقوم بتعريف فضاء أسماء جديد ضمنها للإشارة إلى مجلد الـ View، ضمن نضيف عنصر من نوع Person.xaml إلى الواجهة بالشكل التالي:

```
<Window x:Class="Mvvm.Session01.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:Views="clr-namespace:Mvvm.Session01.View"
        Title="MainWindow" Height="140" Width="340">
    <Grid>
        <Views:Person/>
    </Grid>
</Window>
```

8. الآن نقوم بتشغيل البرنامج بالضغط على F5، لتظهر لدينا الواجهة كالتالي:



ربما تتساءل ما فائدة هكذا برنامج لا يسمن ولا يغني من جوع؟

في الواقع وكما تحدثت أن عملية التطوير ستكون بشكل إصدرات، وهذا الإصدار سيكون طريقنا للتعرف على الميزات الأخرى والتي ستضفي ميزات كبيرة على تطبيقنا، لذلك سنبدأ بمناقشة بعض المشاكل، والتي سينتج عن حلها التعرف على ميزات جديدة.

Notification System

كان من المفترض عند إدخال قيم ل FirstName, LastName أن يتم تحديث قيمة الـ FullName، وقد لاحظنا أن الخاصية FullName ضمن المودل Person عبارة عن الإسم الأول متبوع بمسافة بيضاء ثم الإسم الأخير، لكن شيء من هذا لم يحدث!

الطريقة التقلدية كانت بإعادة إسناد خاصية الإسم الكامل للحقل FullNameTextBox، وهذا صحيح، لكننا الآن ضمن نظام منفصل الطبقات ويجب فصل عملية العرض عن عملية الـ Loigc. لذلك يجب وجود آلية مهتمها إبلاغ FullName TextBox بضرورة تحديث قيمتها، بسبب حدوث تغير بإحدى مكوناتها (FirstNam or LastName). هذه الآلية هي Notification ويتم تطبيقها ضمن ViewModel أو Model باعتبارهم مصدر البيانات، عن طريق تطبيق INotifyPropertyChanged Interface، ومن خلاله يمكننا تنبيه نظام DataBinding بضرورة تحديث البيانات اللازمة.

تطبيق عملية التنبيه ضمن Model أو ViewModel تختلف حسب طبيعة المشروع وتركيبته، نحن الآن نملك الخيار، سنضعه ضمن ViewModel

لذلك سنجعل الكلاس PersonViewModel يرث من INotifyPropertyChanged والموجود ضمن فضاء الأسماء System.ComponentModel، ستعطينا هذه العملية حدث وحيد يسمى PropertyChanged من نوع PropertyChangedEventHander، لذلك عند حدوث تعديل معين في البيانات، نقوم بتشغيل هذا الحدث.

عملية تشغيل الحدث تكون كالتالي:

```
if (PropertyChanged != null)
     PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
```

ومكانها سيكون في مرحلة إسناد البيانات ل FirstName, LastName أي ضمن الـ set كالتالي:

```
public string FirstName
{
     get { return model.FirstName; }
     set
     {
         model.FirstName = value;
         if (PropertyChanged != null)
             PropertyChanged(this, new PropertyChangedEventArgs("FirstName"));
       }
}
```

لكن ولكي لا نعيد كتابة السطر الممل السابق، سنقوم بوضعه ضمن ميثود خاصة، نسميها RaisePropertyChanged، ونعطيها إسم الخاصية التي حدث عندها التغيير، فتكون لدينا بالشكل التالي:

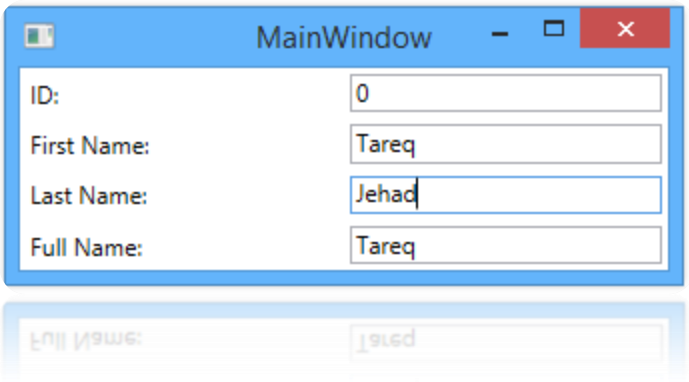
```
private void RaisePropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
```

بعد هذه المجموعة من التعديلات يصبح لدينا شكل الـ PersonViewModel.cs بالكامل كالتالي:

```
namespace Mvvm.Session01.ViewModel
{
    using Mvvm.Session01.Model;
    using System.ComponentModel;
    public class PersonViewModel : INotifyPropertyChanged
    {
        private Person model;

        public int Id
        {
```

```
            get { return model.Id; }
            set
            {
                model.Id = value;
            }
        }

        public string FirstName
        {
            get { return model.FirstName; }
            set
            {
                model.FirstName = value;
                RaisePropertyChanged("FullName");
            }
        }

        public string LastName
        {
            get { return model.LastName; }
            set
            {
                model.LastName = value;
                RaisePropertyChanged("FullName");
            }
        }
        public string FullName
        {
            get { return model.FullName; }
        }
        public PersonViewModel(Person person)
        {
            model = person;
        }

        public PersonViewModel()
            :this(new Person())
        { }

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;

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

الآن نقوم بتشغيل التطبيق F5، ونقوم بإدخال الإسم الأول والإسم الأخير، ولكن نلاحظ ملاحظة بسيطة:



الملاحظة أنه لا يتم تحديث الإسم الكامل إلا بعد فقد التركيز من الحقل، مثلاً كما نلاحظ في الصورة، تم تحديث الإسم الكامل إلى الإسم الأول فقط! لأننا مازلنا داخل حقل الإسم الأخير، وبمجرد تغيير التركيز من حقل الإسم الأخير سنحصل على الإسم الكامل كما يفترض.

هنا ينطرح سؤال؟ لماذا هذا السلوك، وهل يمكن تغييره؟

سبب هذا السلوك أنه في بعض الأحيان ربما يكون تحديث القيم، يعتمد على عملية طويلة نسبياً، كالبحث في قاعدة البيانات، فليس من الضروري تحديث قيمة معينة عند كتابة كل حرف، مما سيؤخر عمليه التنفيذ، بل عند كتابة القيمة كاملة، نقوم بالتحديث لمرة واحدة، وهذا سنلاحظة في بعض الأجزاء لاحقاً، أما الآن وبما أننا بحاجة لتحديث الإسم الكامل عند الكتابة مباشرة، نقوم بالتحديث على عملية DataBinding حيث نطلب منها القيام بالتحديث فور الكتابة، لذلك نذهب على Person.xaml ونقوم بتحديثها بالشكل التالي:

```
<UserControl x:Class="Mvvm.Session01.View.Person"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <UniformGrid Columns="2">
        <Label Content="ID:"/>
        <TextBox Text="{Binding Id}"/>

        <Label Content="First Name:"/>
        <TextBox Text="{Binding FirstName, UpdateSourceTrigger=PropertyChanged}"/>

        <Label Content="Last Name:"/>
        <TextBox Text="{Binding LastName, UpdateSourceTrigger=PropertyChanged}"/>

        <Label Content="Full Name:"/>
        <TextBox Text="{Binding FullName, Mode=OneWay}"/>
    </UniformGrid>
</UserControl>
```

الآن قم بتشغيل البرنامج ولاحظ السلوك المطلوب.

سأترك لك مجموعة من التساؤلات، قبل الإنتقال لإكمال هذا الجزء وعرض مجموعة من Person، وتنفيذ مجموعة عمليات عليها.

By: Tareq Jehad
Email: tareqjehad@yahoo.com