



Implantation IESN

Environnement de développement de logiciels

IG3 — Labo 4 — 2016-2017



Objectifs

- Application Windows Phone ou Universal App qui respecte le MVVM

Notre projet va comprendre deux pages : la première affichera une liste d'étudiants, la seconde les coordonnées d'un étudiant sélectionné sur la première page.

Votre solution comprendra au moins :

- Soit un projet dans lequel vous aurez 3 folders : View, ViewModel et Model
- Soit plusieurs projets (un pour PC/tablette, un pour smartphone et un « shared ») ; dans l'espace partagé, vous y placez au moins ViewModel et Model. (Les vues étant dans chacune de leur partie (projet)).

Notre couche View comprendra deux classes MainPage et SecondPage.

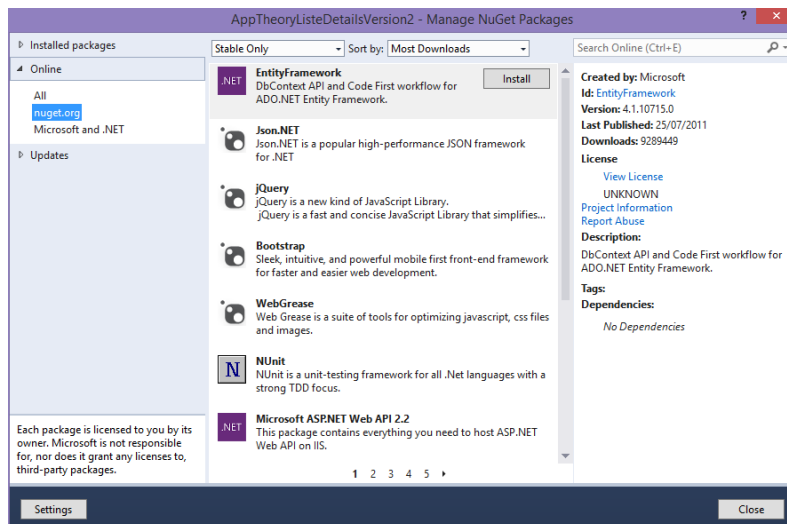
La couche ViewModel comprendra aussi deux classes MainPageViewModel et SecondPageViewModel.

Etape 1 : liaison de données

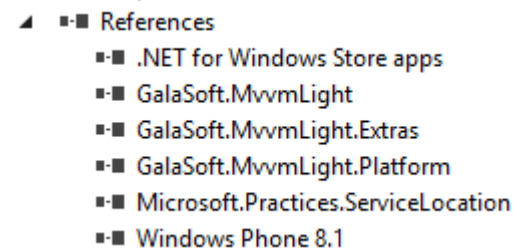
1. Créez une nouvelle solution.
2. Ajoutez un dossier Model dans lequel nous créons deux classes :
 - a. Student.cs qui comprend comme propriétés Name et Age ainsi qu'un constructeur général.
 - b. AllStudents.cs qui crée des données pour les tests.

```
public static class AllStudents
{
    1 reference
    public static IEnumerable<Student> Students { get; set; }

    2 references
    public static IEnumerable<Student> GetAllStudents()
    {
        return Students = new List<Student> {
            new Student("Student 1", 20),
            new Student("Student 2", 19),
            new Student("Student 3", 21)
        };
    }
}
```



3. Nous souhaitons un dossier ViewModel pour la découpe MVVM ; cette découpe peut



se faire de manière automatique via un outil spécifique. Ces outils

spécifiques sont nombreux. Nous prenons NuGet package MvvmLight qui nous aidera dans l'implémentation de cette couche VM. Ce package n'est plus si automatique mais offre encore des mécanismes aisés.

Placez-vous sur le projet, un clic droit ; sélectionnez *Manage NuGet Package* pour télécharger MvvmLight. Faites chercher MvvmLight et téléchargez (cf. page suivante).

Vérifiez que les références ont bien été incorporées.

Soit un dossier ViewModel a été ajouté automatiquement dans votre projet. Il contient deux classes *ViewModelLocator.cs* et *MainViewModel.cs* qui correspondra à *MainPage.xaml*.

Soit vous devez les ajouter manuellement mais procédons par étape.

Dans la couche ViewModel, une première classe ***ViewModelLocator*** joue le rôle de centralisateur.

Avant de coder cette classe, que faire au niveau central de l'application, donc dans *App.xaml* ?

Dans le fichier App.xaml, créons une ressource nommée Locator ; elle sera créée par l'instanciation de la classe *ViewModelLocator*.

Pour ce faire,

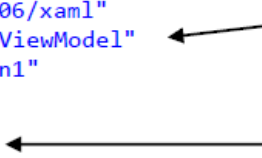
- Créez tout d'abord un alias pour *MyNomApplication.ViewModel* nommé vm
- Ensuite, créez la ressource statique Locator qui nous servira par la suite.

Ce qui donne dans App.xaml :

Une clé est ainsi associée au ViewModel ; elle porte le nom **Locator**.

Plus important : une instance de ViewModelLocator va être créée une seule fois au lancement de l'application.

```
<Application
  x:Class="MVVMSimple.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:vm="using:AppTheoryListeDetailsVersion1.ViewModel"
  xmlns:local="using:AppTheoryListeDetailsVersion1"
  RequestedTheme="Light">
  <Application.Resources>
    <vm:ViewModelLocator x:Key="Locator"/>
  </Application.Resources>
</Application>
```



Que placer dans la classe ViewModelLocator.cs ?

- La création du container IoC pour l'injection de dépendances¹
- L'enregistrement des classes ViewModel
- La création du service de navigation
- L'enregistrement du service dans le conteneur IoC
- Cela permettra aux classes du ViewModel de recevoir une instance du système de navigation dans chacun de leur(s) constructeur(s)
- Une centralisation des propriétés qui permettent l'accès à des instances de chacune des classes du ViewModel dans chacune des classes de la View

Procédons par étape. Soit vous avez déjà les deux classe ViewModelLocator et MainViewModel soit vous les ajoutez au dossier ViewModel.

Dans la classe ViewModelLocator, vous y lisez le résultat ci-dessous soit vous l'encodez.

On crée le container IoC, on enregistre le premier VM (MainViewModel) et on ajoute une propriété Main qui renvoie une instance de MainViewModel.

¹ A vous de rechercher ce que signifie cette notion en programmation OO

```

1 reference
public ViewModelLocator()
{
    ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

    SimpleIoc.Default.Register<MainViewModel>();
}

1 reference
public MainViewModel Main
{
    get
    {
        return ServiceLocator.Current.GetInstance<MainViewModel>();
    }
}

```

4. Repartons côté View. Soit MainPage.xaml.

Nous souhaitons y faire apparaître une liste d'étudiants qui vient de la couche Model mais en passant par le ViewModel. Il y aura du *DataBinding*. Dans la View, nous aurons une listView liée à une propriété Students du VM ; cette propriété permettra de garnir les données provenant de AllStudents (liste des étudiants) qui est dans la couche « données ».

Pour ce faire, dans la vue:

- a. Plaçons un lien entre la classe de la View et la classe correspondante dans le ViewModel via le **DataContext** en nous servant de la ressource **Locator**.

```

<Page
    x:Class="AppTheoryListeDetailsVersion2.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:AppTheoryListeDetailsVersion1"
    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}">

```

Le DataContext lie la ressource Locator définie au niveau central (App.xaml) et la propriété placée dans le ViewModelLocator qui renvoie l'instance du ViewModel correspondant (Main). Ainsi, la classe de la vue et la classe du VM sont liées.

5. Dans la vue, créez le lien de la liste de la vue et celle qui viendra du ViewModel via un Binding.

```

<ListView Grid.Row="1" Width="355" HorizontalAlignment="Left" ItemsSource="{Binding Students,Mode=TwoWay}"

```

Où Students est une propriété de la classe MainViewModel.

6. Dans la classe `MainViewModel`, encodez l'en-tête :

```
public class MainViewModel : ViewModelBase, INotifyPropertyChanged
```

Elle implémente *ViewModelBase* (dû à l'emploi de MVVMLight) et *INotifyPropertyChanged* (cf. théorie).

Créez une propriété *Students*.

Elle doit être de type *ObservableCollection* (cf. théorie). La classe doit implémenter l'interface *INotifyPropertyChanged* et donc, la propriété a une allure différente.

```
public class MainViewModel : ViewModelBase, INotifyPropertyChanged
{
    private ObservableCollection<Student> _students;

    4 references
    public ObservableCollection<Student> Students
    {
        get { return _students; }
        set {
            _students = value;
            RaisePropertyChanged("Students");
        }
    }
}
```

Dans la propriété, nous devons y signaler de faire attention aux changements. De nouveau, MVVMLight nous facilite la tâche par *RaisePropertyChanged*. L'argument de cette méthode est le nom de la propriété (attention !).


Dans le constructeur de la classe, nous lions la propriété *Students* du VM à la liste provenant du Model. Ainsi, les données de la couche « données » garnissent *Students* dans le VM qui, à son tour, garnit la *listView* de la View.

```
public MainViewModel()
{
    Students = new ObservableCollection<Student>(AllStudents.GetAllStudents());
}
- . -
```

Etape 2 : sélection d'un élément affiché sur une nouvelle page

1. Modifiez la vue dans *MainPage* :

```
<ListView Grid.Row="1"
    Width="355" HorizontalAlignment="Left"
    ItemsSource="{Binding Students,Mode=TwoWay}"
    SelectedItem="{Binding SelectedStudent, Mode=TwoWay}"
    Margin="0,35.167,0,-35">
```



2. Modifiez MainViewModel :

```
private Student _selectedStudent;

5 references
public Student SelectedStudent
{
    get { return _selectedStudent; }
    set
    {
        _selectedStudent = value;
        if (_selectedStudent != null)
        {
            RaisePropertyChanged("SelectedStudent");
        }
    }
}
```

3. Ajoutez une page de type Basic Page : SecondPage.xaml.
Dans la couche ViewModel, ajoutez une classe SecondViewModel.cs.
4. Dans la classe ViewModelLocator.cs, nous enregistrons SecondViewModel au container IoC et nous ajoutons une propriété qui donnera accès à une instance unique de SecondViewModel.

```
0 references
public ViewModelLocator()
{
    ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

    SimpleIoc.Default.Register<MainViewModel>();
    SimpleIoc.Default.Register<SecondViewModel>();
}

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

5. Ajustez SecondPage.xaml.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:vm="clr-namespace:ViewModel;assembly=ViewModel"
      DataContext="{Binding Source={StaticResource Locator}, Path=Second}">
```

6. Nous devons naviguer entre les pages ; dès lors, nous nous abonnons au navigationService via le ViewModelLocator.
Pour cela, on crée une instance du navigationService, appelée navigationPages, on l'enregistre au container et on crée deux abonnements correspondant aux deux pages de notre app.

```

public ViewModelLocator()
{
    ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

    SimpleIoc.Default.Register<MainViewModel>();
    SimpleIoc.Default.Register<SecondViewModel>();
    NavigationService navigationPages = new NavigationService();
    SimpleIoc.Default.Register<INavigationService>(() => navigationPages);
    navigationPages.Configure("MainPage", typeof(MainPage));
    navigationPages.Configure("SecondPage", typeof(SecondPage));
}

```

Les deux dernières instructions associent deux libellés MainPage et SecondPage aux types de page.

7. Dans MainViewModel,

- On ajoute au moins une variable d'instance pour la navigation : `_navigationService`
- On ajoute un constructeur (avec la marque « préféré ») avec un paramètre pour la navigation.

```

private INavigationService _navigationService;
[PreferredConstructor]
public MainViewModel(INavigationService navigationService)
{
    _navigationService = navigationService;
    // initialisation de la liste
    Students = new ObservableCollection<Student>(AllStudents.GetAllStudents());
}

```

- Certains ajoutent une valeur par défaut au paramètre : `INavigationService navigationService = null`

8. Dans main.xaml, ajoutez une barre d'outils :

```

<Grid x:Name="CommandBarCourse" Grid.Row="2">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"></ColumnDefinition>
        <ColumnDefinition Width="Auto"></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <AppBarButton x:Name="EditStudent" Grid.Column="1"
        Icon="Edit"
        Command="{Binding EditStudentCommand}"></AppBarButton>
</Grid>

```

Nous y apercevons une Command en lieu et place du Button_Click qui nécessiterait du code-behind. La propriété associée doit être décrite dans la classe MainViewModel (cf. théorie).

9. MVVMLight simplifie le système de commandes vu en théorie grâce notamment à une méthode RelayCommand.

Déclarez l'attribut privé `_editStudentCommand` de type `ICommand` et sa propriété. Dans la propriété, on n'écrit pas de `set` (parfaitement inutile...) mais la méthode d'accès (`get`).

Si la valeur de l'attribut est `null`, il faut l'instancier par la méthode `RelayCommand` dont l'argument est une expression `lambda` qui remplace une fonction (cf. délégué).

```
private ICommand _editStudentCommand;
0 references
public ICommand EditStudentCommand
{
    get
    {
        if (this._editStudentCommand == null)
        {
            this._editStudentCommand = new RelayCommand(() => EditStudent());
        }
        return this._editStudentCommand;
    }
}

1 reference
private void EditStudent()
{
    // élément sélectionné ?
    if (CanExecute() == true)
    {
        _navigationService.NavigateTo("SecondPage", SelectedStudent);
    }
}

1 reference
public bool CanExecute()
{
    return (SelectedStudent == null) ? false : true;
}
```

La navigation se fait via le *navigationService* où nous accédons à la seconde page via la méthode *NavigateTo* qui a deux arguments : le libellé associé à la seconde page placé dans le *ViewModelLocator* et l'élément à transférer, à savoir l'étudiant sélectionné.

10. Préparons SecondViewModel.cs.

Placez une propriété correspondant à l'élément sélectionné de la page principale et la navigation.

Ainsi que l'attribut `_navigationService` et le constructeur (cf. ci-avant).

```
3 references
public Student SelectedStudent
{
    get;
    set;
}
```



```
private INavigationService _navigationService;
[PreferredConstructor]
0 references
public SecondViewModel(INavigationService navigationService = null)
{
    _navigationService = navigationService;
}
```

Il faut maintenant programmer ce qui se passe quand on arrive sur la deuxième page, c'est-à-dire programmer la méthode `OnNavigateTo`.

Plutôt que de coder tout dans le code-behind (`SecondPage.xaml.cs`) associé à la vue où se trouve la méthode `OnNavigateTo`, on programme en deux étapes.

a. Dans le code-behind, on y place :

```
public sealed partial class SecondPage : Page
{
    public SecondPage()...

    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        ((SecondPageViewModel)DataContext).OnNavigatedTo(e);
    }

    protected override void OnNavigatedFrom(NavigationEventArgs e)...
```

b. Dans le VM, on y place :

```
1 reference
public void OnNavigateTo(NavigationEventArgs e)
{
    SelectedStudent = (Student)e.Parameter;
}
```

Tout est prêt. On ajuste la vue.

```
<StackPanel Orientation="Vertical" Margin="0,51.167,0,-51">
  <StackPanel Orientation="Horizontal" Height="Auto">
    <TextBlock Text="Nom : " FontSize="20" Width="100"/>
    <TextBlock Text="{Binding SelectedStudent.Name}" FontSize="20" />
  </StackPanel>
  <StackPanel Orientation="Horizontal" Height="Auto">
    <TextBlock Text="Age : " FontSize="20" Width="100"/>
    <TextBlock Text="{Binding SelectedStudent.Age}" FontSize="20"/>
  </StackPanel>
</StackPanel>
```