TP WPF

Le MVVM, la 3D, les ressources

Lemettre Arnaud

Version 1.0

13 Pages

15/08/2010

EPITA-MTI2014-WPF-TP-3D-MVVM-RESOURCES

```
<Button Content="Prof-MTI">
    <Button.Triggers>
        <EventTrigger RoutedEvent="Button.Click">
            <EventTrigger.Actions>
                <BeginStoryboard Storyboard="{StaticResource StudentLearning}"/>
```

Multimédia
Technologies de l'Information

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Propriétés du document

Auteur	**Lemettre Arnaud**
Version	1.0
Nombre de pages	13
Références	EPITA-MTI2014-WPF-TP-3D-MVVM-RESOURCES

Historique du document

Date de vision	Version	Auteur	Changements
06/07/2010	0.1	Lemettre	création

Site de référence

Description	url
Site MTI	
Blog MTI	

Sommaire

Introduction

Le but de ce TP est de se familiariser avec la technologie WPF de Microsoft.

Durant ce TP les notions suivantes seront abordées :

- La construction d'un projet WPF autour du design pattern MVVM
- Réalisation d'une interface utilisant les fonctionnalités de 3D
- Utilisation des ressources de WPF

Ce travail est à faire individuellement, tout code similaire sur deux personnes sera considéré comme un travail non rendu et non négociable, il est recommandé de lire le sujet jusqu'à la fin.

Toute mention de login_l fait référence à votre login EPITA.

Ce TP ne demande pas plus de 4h de travail.

Bonne chance ;)

Partie 1 : Utilisation du MVVM

Renseignements

Dans cette partie, vous allez devoir réaliser une interface implémentant le design pattern MVVM.

Le but de l'application étant de pouvoir créer une fenêtre de logging et doit rediriger vers la fenêtre suivante. Notre projet se composera donc d'une fenêtre permettant la saisie de login / mot de passe, ainsi qu'une autre fenêtre qui s'affichera uniquement dans le cas de validation de l'authentification. Cette dernière action déclenchera la fermeture de la fenêtre de login également.

Un projet implémentant le MVVM est composé comme suit :

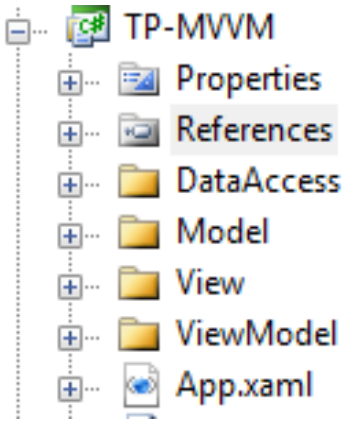

Figure 1 : Composition de la solution Visual Studio

Vous devez ajouter à la racine de votre solution les deux classes suivantes :

- RelayCommand
- WindowCloseBehaviour

La première nous servira pour gérer les commandes. Cette classe pourra être recopiée de projet en projet. En toute logique, elle ne doit pas changer (sauf utilisation particulière dans un projet).

La deuxième classe permet de pouvoir gérer la fermeture de fenêtre au travers de binding.

Les vues

Vous devez maintenant ajouter dans le répertoire des Views, les deux vues permettant d'afficher d'une part les informations de login et d'autre part le succès du login.

Login

La 1ère vue sera donc composée :

- De 2 labels
- De 2 textbox (login / password)
- D'un bouton permettant de valider login / password

Dans cette vue n'oubliez pas de placer un dataTrigger permettant de fermer la fenêtre lors du succès de la méthode de login.

Liste des personnes

Celle-ci sera composée :

- D'une listbox (permettant d'afficher une liste de personnes)

Les ViewModels

Login

A ces vues, il faut créer les viewModels associés. Dans celui en liaison avec le login, il faut une méthode permettant de récupérer l'ensemble des users au travers d'une fonction de la dataAccess (les users peuvent être contenus dans un fichier XML).

Exemple de fichier XML :

```
<?xml version="1.0" encoding="utf-8" ?>
<users>
      <user>
            <name >Admin</name>
            <pwd >Admin</pwd>
      </user>
</users>
```

Pour charger cette liste vous pouvez utiliser du linq to XML.

La commande permettant de déclencher la fonction de login et par voie de conséquence la fonction de login permettant de savoir si une personne est accréditée ou non.
Il ne faudra pas oublier de binder vos propriétés à la vue.

Liste des personnes

Pour le second viewModel associé à la liste de personne, il vous faudra également une fonction qui chargera l'ensemble des personnes dans une liste qui sera bindée à votre vue.

Exemple de fichier XML :

```
<?xml version="1.0" encoding="utf-8" ?>
<list>
  <user>
    <name>thomson</name>
    <age>24</age>
    <firstname>david</firstname>
  </user>
  <user>
    <name>dupont</name>
    <age>42</age>
    <firstname>thomas</firstname>
  </user>
</list>
```

Pour charger cette liste vous pouvez utiliser du linq to XML.

Travail à Faire

Vous devez créer une solution visual studio, avec un projet de type WPF/C#. Vous pouvez créer cette solution avec le framework 4.5.

Le nom de la solution devra se nommer login_lTP2

Le nom du projet devra être WPF_MVVM

Partie 2 : Faire de la 3D

Renseignements

Fini de s'amuser … Maintenant qu'on à les bases de la 3D en WPF, le but ici n'est pas de refaire un jeu vidéo mais de faire une interface.

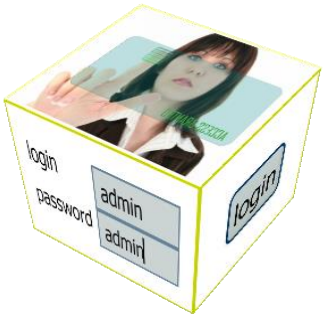

Figure 2 : Interface à réaliser

Pour cela nous allons repartir de l'interface simple du précédent exercice consistant à faire une interface d'authentification. Pour cela on va créer un cube avec une face pour les saisies et une autre face pour le bouton.

Ajout de ressources

Afin de simplifier le développement nous allons nous servir de fichiers ressources. Ce fichier ressource va contenir les faces de notre cube.

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <MeshGeometry3D
            x:Key="squareMeshFrontLeft"
            Positions="-1,-1,1 1,-1,1 1,1,1 -1,1,1"
            TriangleIndices="0 1 2 0 2 3"
            TextureCoordinates="0,1 1,1 1,0 0,0"/>
    <MeshGeometry3D
            x:Key="squareMeshFrontRight"
            Positions="1,-1,1 1,-1,-1 1,1,-1 1,1,1"
            TriangleIndices="0 1 2 0 2 3"
            TextureCoordinates="0,1 1,1 1,0 0,0"/>
    <MeshGeometry3D
            x:Key="squareMeshTop"
            Positions="-1,1,1 1,1,1 1,1,-1 -1,1,-1"
            TriangleIndices="0 1 2 0 2 3"
            TextureCoordinates="0,1 1,1 1,0 0,0"/>
</ResourceDictionary>
```

Ce fichier cependant ne contient pas toutes les ressources qui nous serons nécessaires pour cette application. Il faut notamment y ajouter la couleur pour la lumière :

```
<DiffuseMaterial
x:Key="visualHostMaterial"
Brush="White"
Viewport2DVisual3D.IsVisualHostMaterial="True"/>
```

La vue

Pour cet exemple,notre élément de base sera une grille dans laquelle on va venir positionner notre Viewport3D.

La caméra

Pour la caméra, vous utiliserez une PerspectiveCamera. Le lookDirection et la position sont à votre discrétion.

Lumière

Afin d'éclairer la scène, placer une lumière directionnelle de couleur blanche par exemple. Si vous souhaitez avoir un rendu différent, vous pouvez ajouter d'autres lumières.

Les interfaces

Ajouter les différentes faces de cube pour construire l'interface en 3D. Sur la face du haut vous mettrez une image symbolisant la connexion. Sur la face de gauche les 2 textbox et sur la face de droite le bouton de login. Le bouton login ne fera qu'afficher le login et mot de passe qui ont été saisis. Pour organiser les composants à l'intérieur des faces, vous pouvez également utiliser une grille.

Afin de pouvoir afficher à l'écran les différents composants, vous devrez sans doute ajuster la vue en appliquant une transformation de type rotation sur les axes sur lesquels vous avez défini votre interface.

Travail à faire

Vous devez créer un nouveau projet de type WPF/C#. Vous devez créer ce projet avec le framework 4.5 dans la solution créée précédemment.

Le projet devra être nommé : WPF_3D

Partie 3 : Bonus

Renseignements

Cette partie n'est pas obligatoire mais vous permettra de mettre un concept qui permettra de traiter des opérations longues tout en ne figeant pas votre interface.

Travail à faire

Créer un projet WPF dans la solution réalisée dans la 1ère partie. Cette interface sera composée de deux boutons.

Le projet devra se nommer : WPF_bonus

Ajouter une fonction qui simule un traitement long (utiliser la classe Sleep par exemple). Sur le 1er bouton vous appellerez directement cette fonction.

Sur le deuxième bouton vous implémenterez un background worker. Ce background worker appellera également cette fonction.

Si vous avez bien implémenté vos traitements lors de l'appui sur le premier bouton, l'interface deviendra blanche, lors de l'appui sur le second bouton l'interface doit rester réactive.

Modalité de rendu

Les fichiers seront à rendre dans une tarball ayant pour nom :

login_l.zip

Cette tarball devra comprendre à la racine:

- Un dossier contenant la solution Visual Studio qui devra compiler :
  Nom : login_lTP2
  Une fois décompressé, nous devrions avoir :
  /Login_lTP2
  *.sln
  /WPF_MVVM
  /WPF_3D
  /WPF_bonus

Le tout à envoyer sur l'adresse mti.rendu.dotnet@gmail.com avec les balises suivantes :

[MTI2014][WPF][login_l][TP2]

Annexe

RelayCommand

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Input;
using System.Diagnostics;

namespace TP_MVVM
{
    public class RelayCommand : ICommand
    {

        #region variables
        /// <summary>
        /// action à executer
        /// </summary>
        readonly Action<object> _execute;
        /// <summary>
        /// test pour savoir si on peut executer la commande
        /// </summary>
        readonly Predicate<object> _canExecute;
        #endregion

        #region Constructeur

        /// <summary>
        /// Crée une nouvelle commande que l'on peut toujours executer
        /// </summary>
        /// <param name="execute">Le code à executer</param>
        public RelayCommand(Action<object> execute)
            : this(execute, null)
        {
        }

        /// <summary>
        /// Crée une nouvelle commande
        /// </summary>
        /// <param name="execute">le code à executer</param>
        /// <param name="canExecute">Si on peut executer ou pas</param>
        public RelayCommand(Action<object> execute, Predicate<object> canExecute)
        {
            if (execute == null)
                throw new ArgumentNullException("execute");

            _execute = execute;
            _canExecute = canExecute;
        }

        #endregion

        #region ICommand Members

        [DebuggerStepThrough]
        public bool CanExecute(object parameter)
        {
            return _canExecute == null ? true : _canExecute(parameter);
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
```

```
            remove { CommandManager.RequerySuggested -= value; }
        }

        public void Execute(object parameter)
        {
            _execute(parameter);
        }

        #endregion // ICommand Members
    }
}
```

WindowCloseBehaviour

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;

namespace TP_MVVM
{
    /// <summary>
    /// permet de fermer une fenêtre par binding MVVM
    /// </summary>
    public static class WindowCloseBehaviour
    {
        public static void SetClose(DependencyObject target, bool value)
        {
            target.SetValue(CloseProperty, value);
        }

        public static readonly DependencyProperty CloseProperty =
            DependencyProperty.RegisterAttached(
            "Close",
            typeof(bool),
            typeof(WindowCloseBehaviour),
            new UIPropertyMetadata(false, OnClose));

        private static void OnClose(DependencyObject sender,
DependencyPropertyChangedEventArgs e)
        {
            if (e.NewValue is bool && ((bool)e.NewValue))
            {
                Window window = GetWindow(sender);
                if (window != null)
                    window.Close();
            }
        }

        private static Window GetWindow(DependencyObject sender)
        {
            Window window = null;

            if (sender is Window)
                window = (Window)sender;

            if (window == null)
                window = Window.GetWindow(sender);

            return window;
        }
    }
}
```