Notes sur le module : *WPF.*

Armel Pitelet

22 janvier 2022

Table des matières

1 Bases

Mot clef : **https ://fr.wikipedia.org/wiki/Windows_Presentation_Foundation**.

Permet de créer des application en mode fenêtre.Pour utiliser WPF, soit on passe par du .Net Framework (ancienne version) soit on passe sur du .Net Core (nouvelle).

Remarque : *XAML.*

Language balise associé au WPF. C'est via XAML que l'on définit le design. Voir les différents layout ci dessous

Remarque : *Stack Panel.*

Layout qui permet de stacker plusieurs éléments les uns à la suite des autres.

Remarque : *Wrap panel.*

Identique au stack panel mais passe à la ligne suivante lorsque la ligne en cours est pleine (dynamiquement en fonction de la taille de l'écran contrairement au stack panel).

Remarque : *Dock panel.*

Permet de fixer un élément à un endroit donné avec une taille donnée.

Remarque : *Grid.*

Permet de créer des lignes et des colonnes avec des tailles possiblement relative les unes aux autres.

Remarque : *Canvas.*

Permet un placement libre des éléments.

Note :

Pour une intro à wpf : https://wpf-tutorial.com/.

Remarque : *Textbox.*
Permet de créer une boite de texte éditatble.

Remarque : *Textblock.*
Même chose que textblox mais le texte n'est pas éditable.

Remarque : *Combobox.*
Permet de faire une liste déroulante à possibilité de choix fixes.

Remarque : *Listbox.*
Même chose que combobox mais sous forme de liste et pas d'un menu déroulant.

Remarque : *Menu.*
Permet de faire des menus, avec potentiellement des sous menu inclus.

Note :
Attention par défaut les images ne sont pas embarquées dans l'application WPF. On les voits dans le designer mais pas dans l'application une fois celle ci lancée. Il faut changer le statut de l'image en "ressource de l'application" (dans les paramètres Propriétés/Avancé). Cela permet d'embarquer l'image avec le dll. On peut aussi la mettre en "Contenue". A ce moment l'image sera copiée (à la compilation) avec la même arborescence que l'application lors de la création.

Note :
Pour faire des gitignore facilement https://www.toptal.com/developers/gitignore

Note :
Pour de la génération de classe automatique https://github.com/AutoFixture et https://github.com/bchavez/Bogus.

Remarque : *Utiliser les ToString.*
Mauvaise pratique pour faire de l'affichage en WPF

2 Event et Binding

Remarque : *Sur les events.*
Les events sont en fait des implémentation du pattern observateur qui permettent de déclencher les évènement sur un changement d'état. POur ce sur quoi c'est basé Nico pour son cour voir : https://docs.microsoft.com/fr-fr/dotnet/standard/events/.

Remarque : *Sur le remplacant de WCF.*
Il s'agit de gRPC : google remote protocole col (service gRPC ASP.Net Core). S'appuie sur http 2 (sérialisation binaire). Très proche sur du wcf (basé sur SOAP : sérialisation xml) mais avec de bien meilleur performance.

Remarque : *Sur le binding.*
Il est possible d'utiliser le Binding pour relier n'importe quel élément. Attention on ne bind que les propriétés, pas les attributs.

Remarque : *DataContext.*

Définit la source des bindings. (peut etre un objet fenêtre ou un objet de type manager, ou autre...).

Remarque : *ObservableCollection.*

Permet de définir des collections observables : permet de s'abonner par la suite pour suivre les changements.

Remarque : *Sur IList et IEnumerable.*

Si l'on veut faire la modif autant passer par le IList mais si l'on veut just eliste autant avoir un IEnumerable pour garder la généricité la plus haute possible.

Remarque : *Sur le mapping.*

Il ne fuat pas faire trop de profile de mapping, mais les grouper par grand groupes d'objets (et pas par objets).

Remarque : *Sur la notion de Model.*

Un ObjectModel (dédié à la vue) sert pour l'affichage : c'est encore différent d'un Dto(transfert sérialisation), d'une Entity (classe de bdd) ou d'un objet métier. Il peut contenir de la validation de données supplémentaire.

Remarque : *Sur les Entity.*

Les Entities peuvent contenir des méthodes.

Remarque : *Element Name.*

après un binding = qqchose (comme SelectedItem), ElementName=UnAutre élément () permet de binder à autre chose que le context courant (sur un autre élément graphique). On peut également changer le `mode` de binding (voir one way, two way, ...). `UpdateSourceTrigger` permet de changer l'élément déclencheur (PropertyChanged,...)

Remarque : *Pour les changement sur les property soient appliqué sans passer par un SelectedItem.*

On peut utiliser INotifyPropertyChanged dans nos modèles. Cela impose de définir un évènement handler PropertyChangedEventHandler. L'implémentation de la méthode associée impose de changer les propriétés classiques en attribut + propriété set qui track le changement (voir le UserModel dans le code de nico.). En fait SelectedItem a en interne ces propriété qui font que les changement dessus sont suivis.

Remarque : *L'annotation [CallerMemberName].*

devant un argument dans une méthode permet de récupérer le nom de la propriété qui appel cette méthode.

Remarque : *ILSPie.*

permet de récupérer du code décompiler à partir d'un .dll. Le fichier pdb permet de faire le lien entre le .dll et le code .cs

Remarque : *Lors d'un Binding : Trois questions à se poser.*

- 1 - Quel est le DataContext ? Si il n'y en a pas on remonte à l'élément du dessus. (dans le xaml, jusqu'au niveau de la fenêtre.). Ici on a set le DataContext au niveau de la main window.
- 2 - Quelle est la propriété source (itemsource ou à droite du binding), propriété cible (selecteditem ou à gauche du binding),

— 3 - Quel mode de binding utilisé (dans un sens, dans l'autre, dans les deux sens, property Mode=... dans un binding).

Remarque : *Exemple implicite du binding.*
Le textBlock Text = "{Binding Name}" est un binding implicite dans le context définies par item-source.

Remarque : *Pour un équivalent du apt get-install.*
Voir ChocolaTey (présent dans node.js) pour un équivalent sous windows.

Note :
Voir Bogus pour un autre genre de AutoFixture.

Remarque : *Sur les converter.*
Il faut hériter de IValueConverter et décorer avec ValueConversion(source, destination). Il faut ensuite implémenter Convert (du modèle à la vue) et ConvertBack(de la vue vers le modèle). Dans les namespace (xlmns en début de fichier xaml), puis les ressources (Env Windows ressource) il faut ensuite inclure les converter. On peut ensuite utiliser le converter. Ces converters peuvent servir pour les images par exemple (convertion d'image) ou bien faire de la vérifiction de droit.

Remarque : *Sur la vue.*
On appel vue la partie xaml du wpf. Les donnéès provenant de la vue proviennent de la fenêtre. Voir la v2 du code de nico.

3 Sur lien wpf et api

Remarque : *Si l'on oublie les chaines deconnection.*
Il existe un site connection string qui répertorie tt les synatxes possiblent en fonction du type de bdd : https://www.connectionstrings.com/.

Remarque : *Sur les migrations avec CLI.*
`dotnet ef migrations add InitialCreate --projet chemin_vers_le.csproj` pour initialiser la première migration (il faut lui indiquer le chemin du projet ou est le DbContext). On doit aussi pouvoir faire -- output projetcible.csproj pour déplacer la migration ailleur. Pour faire la migration il est nécessaire d'avoir un constructeur qui prend un DbContextOptions.

Remarque : *Pour utiliser la chaine de connexion du appsettings.json pour la migration.*
Parait compliqué. On peut au mieux lui passer la chaine via le programme de l'API).

Remarque : *Sur l'update de la bdd.*
Soit on passe par les ligne de commande (via update) ou bien le configurer dans le program.cs de l'api. (voir la partie `app.Services.CreateScope()`)

Remarque : *Sur le `using` en inline.*
Inline `using` (IDisposable) des truc permet de créer un scope locale dans lequel les `trucs` seront détruit à la fin du scope. IDisposable est une interface qui implémente la méthode Dispose (qui permet de *refermer* des ressources (les rendre inaccessible jusqua ce que le Garbage collector les désaloues)). Ex : un DbContext est un disposable et pourrait être utilisé de cette manière.

Remarque : *Sur le Set du DbContext.*

Lorsque l'on fait un Context.Set<Type> on crée une représentation de la bdd. Si l'on fait un Where dessus on récupère juste un IQueryable sur la représentation. La requête vers la bdd n'est faite qu'à partir du moment ou l'on fait un ToList() ou un AsEnumerable() dessus.

Remarque : *Sur les retry de requête.*

Il faut utiliser des bibliothèques externes comme Polly qui permet d'implémenter un retry avec condition et nombre d'essaie avec ou sans attente.

Remarque : *Dans le cli.*

on peut faire dans un terminal `dotnet watch run` pour faire de la compilaytion automatique à chaque sauvegarde. Il faut être dans le dossier contenant le proejt (ou la solution → à tester.)

Remarque : *Pour styliser un peu les applications.*

Il faut passer par des framework externe : **matérial-design**, ou bien **mahapps-metro**.