

MVVM

De la découverte à la maîtrise

WPF, Silverlight, WP7 — un pattern pour les gouverner tous



Focus

De Jonathan Antoine et
Thomas Lebrun

MVVM

*De la découverte
à la maîtrise*

Jonathan Antoine et Thomas Lebrun

MVVM

*De la découverte
à la maîtrise*

Focus

Digit Books

Éditeur de livres numériques

Brest

infos@digitbooks.fr

http://www.digitbooks.fr

DIGIT BOOKS

© Digit Books, 2011

ISBN : 978-2-8150-0209-7

Prix PDF : 15 €

Couverture : Yves Buraud

Illustration des auteurs

<http://www digitbooks.fr/catalogue/mvwm-antoine-lebrun.html>

Les programmes figurant dans ce livre ont pour but d'illustrer les sujets traités. Il n'est donné aucune garantie quant à leur fonctionnement une fois compilés, assemblés ou interprétés dans le cadre d'une utilisation professionnelle ou commerciale.

Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de l'auteur, de ses ayants droit, ou ayants cause, est illicite (loi du 11 mars 1957, alinéa 1er de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. La loi du 11 mars 1957 autorise uniquement, aux termes des alinéas 2 et 3 de l'article 41, les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective d'une part et, d'autre part, les analyses et les courtes citations dans un but d'exemple et d'illustration.

Table des matières

Préface de David Catuhe	1
Préface des auteurs	3
Pourquoi ce livre	3
Prérequis et public	4
Organisation de l'ouvrage	4
Les conventions utilisées	5
À propos des auteurs	6
Jonathan Antoine	6
Thomas Lebrun	6
À propos des relecteurs	6
Arnaud Auroux	6
David Catuhe	7
Julien Corioland	7
Léonard Labat	8
Remerciements	8
1 Présentation du pattern MVVM	9
1. 1. Le pattern MVVM : qu'est-ce que c'est ?	10
1. 2. Historique et objectifs	11
1. 2. 1. Historique	11
1. 2. 1. 1. Le pattern MVC (Model-View-Controller)	11
1. 2. 1. 2. Le pattern MVP (Model-View-Presenter)	13
1. 2. 2. Les objectifs	14
1. 3. Les frameworks MVVM existants	16
1. 4. Pourquoi et quand préférer le pattern MVVM ?	18
1. 5. À retenir	21

2

Les différents éléments et leurs rôles	22
2. 1. Le modèle de données	24
2. 1. 1. Le modèle global et le modèle de présentation	24
2. 1. 2. Utilisation du modèle par le ViewModel	26
2. 1. 3. Utilisation du modèle par la vue	26
2. 1. 4. Quels sont les différents acteurs ?	27
2. 2. La vue : une représentation concrète des données et des actions possibles	28
2. 2. 1. Une vue peut utiliser plusieurs ViewModel	31
2. 2. 2. Quels sont les différents acteurs ?	32
2. 2. 3. Quel type de contrôles choisir ?	35
2. 3. Le ViewModel : une représentation abstraite de la vue	36
2. 3. 1. Le ViewModel, c'est de la tarte	37
2. 3. 2. Ce n'est pas le contrôleur qu'il y avait dans MVC	39
2. 3. 3. Quels sont les différents acteurs ?	40
2. 3. 4. Le ViewModel n'a pas besoin d'une vue	40
2. 4. À retenir	41

3 Les différentes philosophies 42

3. 1. View First	43
3. 1. 1. Définition	43
3. 1. 2. Implémentation	45
3. 1. 2. 1. Le ViewModelLocator	46
3. 1. 2. 2. Autre possibilité	47
3. 1. 3. Avantages	48
3. 1. 4. Inconvénients	49
3. 1. 5. Autres remarques	50
3. 1. 6. À retenir du View First	50
3. 2. ViewModel First	51
3. 2. 1. Définition	51
3. 2. 2. Implémentation	52
3. 2. 3. Avantages	56
3. 2. 4. Inconvénients	56
3. 2. 5. Autres remarques	58
3. 2. 6. À retenir du ViewModel First	58
3. 3. Model First	59
3. 3. 1. Définition	59
3. 3. 2. Implémentation	59
3. 3. 3. Avantages	61

3. 3. 4. Inconvénients	61
3. 3. 5. À retenir du Model First	62
3. 4. Quelle philosophie adopter	62
3. 5. À retenir	63
4 Construire la partie modèle	64
4. 1. Architecture du modèle de données	65
4. 2. Comment définir le modèle global	68
4. 3. Comment créer le modèle client	72
4. 3. 1. Classe de base du modèle client	72
4. 3. 2. Définition du modèle client	75
4. 3. 2. 1. Mapping	76
4. 3. 2. 2. Wrapping	80
4. 4. Comment concevoir les services d'accès aux données	83
4. 4. 1. Abstraction des services de données	83
4. 4. 2. Implémentation concrète des services	86
4. 4. 3. Le pattern Unit Of Work	92
4. 4. 4. Appels asynchrones	95
4. 5. Validation des données	100
4. 5. 1. Validation du format	101
4. 5. 2. Validation fonctionnelle	110
4. 6. Édition des données : suivi des valeurs du modèle client	113
4. 7. À retenir	119
5 Construire un ViewModel	120
5. 1. Architecture et relations entre les ViewModels	121
5. 1. 1. Classe de base des ViewModels	121
5. 1. 1. 1. Un objet .Net classique	121
5. 1. 1. 2. Pourquoi ne pas utiliser un DependencyObject comme classe de base ?	122
5. 1. 2. Héritage de ViewModels	124
5. 1. 3. Composition	127
5. 2. Un ViewModel pour les gouverner tous	129
5. 3. Injection de dépendances	132
5. 4. Comment exposer les différentes entités ?	136
5. 4. 1. Sous quelle forme exposer le modèle ?	137
5. 4. 2. Comment utiliser les services d'accès aux données ?	140
5. 4. 3. Trier, filtrer et grouper les données exposées et suivre l'élément courant	142
5. 4. 3. 1. Filtrer les données	144
5. 4. 3. 2. Grouper les données	145

6. 4. 2. 3. Amélioration de cette implémentation à l'aide des WeakEvents	215
6. 4. 3. Binding avancé sur les éléments sélectionnés	217
6. 5. Utilisation du modèle	222
6. 5. 1. Afficher les informations de validation	222
6. 5. 2. Utiliser les types particuliers	225
6. 5. 2. 1. Les valeurs énumérées	225
6. 5. 2. 2. Utilisation d'indexer ou de dictionnaires	230
6. 5. 3. Mettre en forme les collections	231
6. 6. Mettre en place des interfaces « vivantes » et réactives	232
6. 6. 1. Transitions simples sans animation	232
6. 6. 2. Transitions avec animations	233
6. 7. Concevoir les vues sans ViewModel concret	238
6. 7. 1. Instances de données	239
6. 7. 2. Données de tests	241
6. 8. À retenir	243
7 Édition de contrôles personnalisés	244
7. 1. Création de contrôles personnalisés	245
7. 1. 1. Bien choisir la classe de base	245
7. 1. 2. Exposer les informations importantes	247
7. 1. 3. Exposer des événements	248
7. 1. 3. 1. Évènements routés	248
7. 1. 3. 2. Évènements classiques	250
7. 1. 4. Donner le contrôle aux développeurs	251
7. 1. 4. 1. Commandes routées	251
7. 1. 4. 2. Commandes classiques	254
7. 1. 5. Définir le rendu visuel du contrôle	254
7. 2. Cas pratique : le contrôle de mise en attente	256
7. 2. 1. Description du besoin : cahier des charges	256
7. 2. 2. Choix du contrôle de base	258
7. 2. 3. Création de la représentation visuelle	259
7. 2. 4. Mise en place de la logique fonctionnelle	263
7. 2. 5. Annuler l'action	265
7. 2. 6. Utilisation concrète du contrôle	267
7. 2. 6. 1. Exemple de code au sein d'une vue (fichier XAML)	267
7. 2. 6. 2. Exemple de code au sein d'un ViewModel	267
7. 3. À retenir	269

8 MVVM et testabilité	271
8. 1. L'importance des tests	271
8. 1. 1. Les tests fonctionnels	272
8. 1. 2. Les tests unitaires	272
8. 1. 3. Les tests d'interfaces graphiques	273
8. 2. Tests du modèle	274
8. 3. Tests du ViewModel	275
8. 3. 1. Utilisation avec WPF	275
8. 3. 1. 1. Initialisation des tests unitaires	276
8. 3. 1. 2. Mise en place de « bouchons »	280
8. 3. 2. Le cas de Silverlight	281
8. 4. Tests des vues	285
8. 5. À retenir	288
9 Glossaire	289
Index	291

Préface de David Catuhe

Comment faire un bon développement ? En voilà une question qui aura et qui fera couler beaucoup d'encre. On peut même dire qu'il s'agit dans le domaine de l'informatique d'une vraie question philosophique. Les design patterns sont apparus pour tenter d'apporter une pierre à la l'édifice qui se propose de répondre à cette interrogation ; leur but étant de fournir des briques algorithmiques pour des situations précises. Ces briques ont tous les avantages d'une solution qui a été longuement réfléchié et couvrent donc un ensemble plus ou moins important de problèmes.

Parmi ces designs patterns, MVVM est sorti récemment du lot car il s'intègre de manière très efficace dans nos environnements de développement, tels que WPF ou Silverlight. Il adresse notamment le besoin impérieux de pouvoir tester simplement et de manière automatique. Il guide également le développeur sur le chemin tortueux de la séparation forte des couches, ce qui permet d'avoir des architectures modulaires et hautement adaptables.

Bien évidemment, MVVM reste un design pattern et ne fera pas le travail à votre place. Il donne juste un cadre, des outils et une manière d'architecturer nos projets. C'est un accélérateur et un moyen de sécuriser un projet. Vous pourriez très bien faire vos développements sans (ou avec mais sans le savoir). Toutefois l'idée est ici de réutiliser des réflexions de la même manière que l'on réutilise du code.

Et comme tous les sujets à la mode, de nombreux écrits sont apparus sur le sujet de MVVM. Étant amateur de philosophie à mes heures, je me suis donc intéressé à la vision que chaque auteur pouvait avoir de l'approche et surtout comment la relation avec le monde réel de l'entreprise et du développement « dans la vraie vie » était faite. Et ce que l'on peut dire c'est que le pragmatisme n'est pas souvent de mise.

C'est donc avec un plaisir non dissimulé que j'ai lu le livre que vous avez entre les mains car il aborde le sujet de manière claire avec des exemples concrets et pragmatiques. On ressent que les auteurs, que je félicite au passage, ne sont pas uniquement des théoriciens mais aussi des développeurs de terrain qui ont mis en pratique maintes fois ce qu'ils conseillent. Ils ne parlent donc pas uniquement des bonnes choses, mais également des écueils qu'il faut éviter.

Au final, vous ressortirez de la lecture de ce livre avec une vision claire de MVVM, mais également avec les poches pleines d'exemples de code et de réflexions à intégrer dans vos futurs développements.

Même si l'outil est important, c'est la main qui le tient qui fait la différence. Et votre main ne sera que plus efficace avec ce livre.

David CATUHE

Responsable relations techniques avec les développeurs – Microsoft

<http://blogs.msdn.com/b/eternalcoding/>

Préface des auteurs

Pourquoi ce livre

Consultants et formateurs en société de services, nous avons souvent été amenés à rencontrer des personnes désireuses de commencer un nouveau projet WPF, Silverlight ou Windows Phone, mais qui ne savaient pas trop par où commencer. Certes, l'évolution de la technologie a permis de mettre en avant des techniques et pratiques maintenant utilisées par bon nombre de développeurs. Mais une question persiste toujours : pour son développement, comment faire pour (bien) démarrer?

Souvent, si ce n'est toujours, ces technologies riment avec MVVM. Cet acronyme du terme anglais *Model View ViewModel* correspond à une architecture spécifique et à une série de bonnes pratiques de développement inexorablement liées à ces nouvelles technologies. C'est dans le cadre de nos expériences « sur le terrain » que nous avons pu nous familiariser avec ce pattern et maîtriser ses spécificités.

Nous avons donc souhaité faire profiter les lecteurs des connaissances que nous avons acquises, sur le terrain, dans le cadre de notre travail. L'objectif

n'est pas de vous transformer en grands gourous de MVVM, mais de vous permettre d'y voir plus clair dans les différentes notions, les différentes façons de procéder, etc. À terme, il vous sera plus facile de savoir par où démarrer votre projet, quelles techniques vous allez utiliser (ou recommander) et surtout la raison de vos choix !

Prérequis et public

Cet ouvrage s'adresse aux personnes soucieuses d'en savoir plus sur ce qui tend à devenir le pattern de référence, lorsque l'on parle de développement d'applications WPF, Silverlight ou Windows Phone. Que vous soyez développeur, architecte ou même chef de projet, ce livre vous permettra de savoir tout ce dont vous avez besoin !

Sa lecture vous présentera bien sûr les **informations techniques nécessaires à la mise en place** de MVVM dans un projet, mais il apporte aussi les **connaissances théoriques** sur ce pattern.

D'un point de vue technique, des notions en programmation orientée objet sont nécessaires pour bien assimiler les exemples. Enfin, si vous connaissez déjà quelques design pattern, c'est un grand plus !

Organisation de l'ouvrage

- × Le chapitre 1, *Présentation du pattern MVVM*, [page 9](#), a pour objectif de présenter, succinctement, le pattern MVVM en détaillant les différents éléments qui le compose.
- × Le chapitre 2, *Les différents éléments et leurs rôles*, [page 22](#), met l'accent sur ces fameux éléments de façon théorique, afin de déterminer clairement quels sont leur rôle, permettant ainsi au lecteur de bien comprendre leur importance.
- × Afin de bien comprendre les différentes philosophies de développement utilisables avec le pattern MVVM, le lecteur pourra se concentrer sur la lecture du chapitre 3, *Les différentes philosophies*, [page 42](#), qui lui

expliquera la différence entre le « View First », le « ViewModel First » et le « Model First ».

- × Les chapitres 4 à 6 présentent d'un point de vue technique, dans le détail et avec des exemples simples, mais issus de la vie réelle, la façon dont chacun des composants (à savoir le modèle, le ViewModel et la vue) doivent être construits et agencés pour pouvoir fonctionner ensemble.
- × Le chapitre 7, *Édition de contrôles personnalisés*, page 244, permettra au lecteur de connaître les techniques à utiliser lorsqu'il souhaite développer ses propres contrôles, tout en faisant en sorte que ceux-ci soient compatibles (et donc facilement utilisables) avec le pattern MVVM.
- × Enfin, parce qu'un développement n'est complet que si l'on s'est assuré qu'il n'y a pas (ou peu) de bugs et afin de limiter les possibles régressions au cours du développement, le chapitre 8, *MVVM et testabilité*, page 271, explique comment réaliser les tests sur les différents éléments, chacun ayant ses propres particularités.

Les conventions utilisées

Voici les conventions typographiques de cet ouvrage :

Italique

Met en exergue les termes nouveaux ou la signification des acronymes.

Police à chasse fixe

Met en valeur les éléments de code dans le texte.

Des notes éveillent votre attention sur des points précis.

Le texte ainsi signalé vous met en garde.

À propos des auteurs

Jonathan Antoine

Architecte/Consultant/Formateur, Jonathan est un passionné d'informatique en général et il s'intéresse plus particulièrement à toutes les problématiques d'interfaces homme-machine dites « naturelles ». Ses travaux professionnels et personnels sont basés sur le framework WPF, Silverlight et Windows Phone. Il est de même reconnu Microsoft MVP dans la catégorie *Client Application Development* (<https://mvp.support.microsoft.com/profile/Jonathan.ANTOINE>).

Depuis quelques années, il essaye de communiquer au mieux sa passion au travers de la création et gestion du site <http://wpf-france.fr/>, son blog en anglais www.jonathanantoine.com, ainsi que par son rôle de modérateur sur le site developpez.com.

Thomas Lebrun

Architecte/Développeur chez Infinite Square, Thomas est expert sur les technologies WPF, Silverlight et Windows Phone. Il s'intéresse plus particulièrement à l'enrichissement de l'expérience utilisateur, ainsi qu'à la mise en place d'architecture logicielle. Son implication dans la communauté, au travers de conférences, articles, livres, etc., lui vaut le titre de Microsoft MVP depuis cinq ans *dans la catégorie Client Application Development* (<https://mvp.support.microsoft.com/profile=FF4FF146-E963-4665-9785-B84B6045E86D>).

À propos des relecteurs

Arnaud Auroux

Architecte/Développeur .Net chez la société Infinite Square. Arnaud est spécialisé dans le développement d'applications riches (Silverlight, WPF et

Windows Phone). Il s'intéresse également aux technologies de géolocalisation et de synchronisation (Bing maps, Sync Framework).

Arnaud intervient sur la réalisation de projets autour de ces technologies. Il est chargé de la mise en place de l'architecture logicielle, ainsi que des bonnes pratiques de développement. Et parce que l'architecture est un sujet complexe à traiter, la vision et l'expérience de chacun apportant une analyse différente du sujet, il a accompagné Thomas et Jonathan dans la relecture de leur ouvrage afin de leur offrir un œil extérieur sur la mise en pratique du modèle MVVM.

David Catuhe

Responsable des relations techniques avec les développeurs chez Microsoft, David est expert sur les technologies en rapport avec l'interface utilisateur (WPF/Silverlight/Html5/Xna/DirectX). Il est passionné par le développement logiciel depuis plus de 15 ans et affectionne tout particulièrement les technologies en rapport avec la 3D (il a dans une vie passée développé un moteur 3D temps réel en .Net). Ce passionné, fier d'être développeur, se définit lui-même comme un geek qui a fait de sa passion son métier. Il a, tout au long de sa carrière, expérimenté et vu de nombreuses approches pour architecturer les développements notamment dans le cadre de MVVM.

Julien Corioland

Consultant et formateur .NET chez Infinite Square, Julien est expert sur les technologies WPF et Silverlight. Il se spécialise tout particulièrement sur la plateforme Windows Phone 7 à laquelle il a consacré un ouvrage. Il travaille au quotidien sur la réalisation de projets utilisant ces technologies et il a apporté au travers ses relectures un regard critique par sa vision de la mise en place de MVVM sur de nombreux projets concrets « dans la vraie vie ».

Léonard Labat

Stagiaire chez Infinite Square, Léonard est passionné par le développement, et particulièrement les technologies Silverlight et Windows Phone, autour desquelles il a coécrit un ouvrage aux éditions ENI. Il s'est, dès les premiers projets sur lesquels il a travaillé, attelé à la mise en place de bonnes pratiques et de patterns tels que MVVM.

Remerciements

Nous tenons à remercier toutes les personnes qui nous ont apportées leur soutien, que ce soit lors de l'écriture ou la relecture de cet ouvrage. Parmi ces personnes, nous remercions tout particulièrement David Catuhe, Julien Corioland, Arnaud Auroux et Léonard Labat ainsi que Simon Ferquel pour leurs relectures et avis d'experts.

Finalement nous (enfin surtout Jonathan) remercions tout particulièrement Noémie Antoine pour son soutien indéfectible et sa patience tout au long de ces mois. Avoir une femme aussi parfaite n'est pas donné à tout le monde.

1

Présentation du pattern

MVVM

Pattern de développement de référence pour le développement d'applications WPF, Silverlight et Windows Phone, le pattern MVVM (Model-View-ViewModel) est devenu un élément incontournable de la panoplie des développeurs. Cependant, tous ne savent pas comment en utiliser toutes les subtilités disponibles, de par sa « jeunesse » et son manque de spécifications détaillées.

Au cours de ce chapitre, nous aborderons précisément ce qu'est le pattern MVVM, et nous ferons un rapide comparatif des différents patterns existants qui ont abouti à sa création.

2

Les différents éléments et leurs rôles

MVVM correspond au triptyque *Model-View-ViewModel*. Ce sont les trois éléments sur lesquels est basé le pattern. Voici une description succincte de chacun :

- × *Le modèle de données* (ou *Model*) correspond aux différentes entités métier utilisées par l'application.
 - × *La vue* (ou *View*) correspond à la représentation qui est faite de ces données.
 - × *Le modèle de la vue* (ou *ViewModel*) correspond à une représentation abstraite de la vue. Il va aussi manipuler le modèle de données.
-

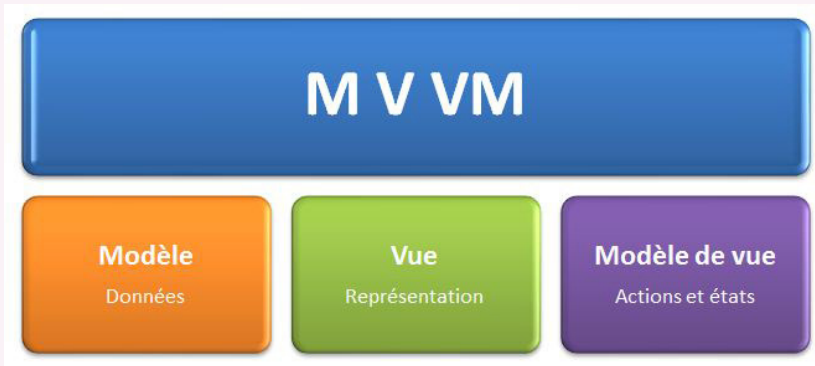


Figure 2-1 : Les différents éléments du pattern MVVM

Comme nous avons pu le voir dans le chapitre précédent, ce pattern a notamment pour objectif de faciliter au mieux le travail conjoint du développeur et du designer. Pour cela, il repose sur le concept de *couplage faible* entre la vue et le ViewModel. On parle de couplage fort lorsque deux éléments d'architecture logicielle sont liés et ne peuvent pas fonctionner indépendamment. Les briques de construction pour enfants sont des exemples concrets du couplage fort : une pièce de Lego ne peut s'imbriquer qu'avec une autre pièce de Lego et on ne peut pas en utiliser une d'une autre marque.

Une solution à ce problème est de créer des contrats fonctionnels : on ne définit pas les différents éléments par ce qu'ils sont, mais par ce qu'ils font ou exposent. Dans notre cas, le ViewModel et la vue ne seront pas liés fortement et ils pourront exister indépendamment l'un de l'autre. Ainsi, le développeur peut travailler sur le ViewModel sans avoir à connaître la vue et le designer peut travailler sur la vue sans avoir à connaître le fonctionnement intrinsèque du ViewModel. La même relation existe entre le modèle de données et la vue.

Dans ce chapitre, nous allons définir plus précisément ce que sont ces différents éléments du pattern ainsi que leurs rôles. Les différents moyens de communication entre eux et les possibilités d'implémentation technique de chacune de ces parties seront abordées dans les chapitres suivants.

Les différentes philosophies

Il existe plusieurs façons d'utiliser le pattern MVVM au sein de ces applications. Ces possibilités, que nous appellerons des « philosophies », sont au nombre de trois :

- × View First
- × ViewModel First
- × Model First

Chacune possède, bien sûr, ses avantages et ses inconvénients que nous allons tenter de détailler tout au long de ce chapitre.

Avant d'aller plus loin, il est important de noter qu'il n'y a pas une philosophie meilleure que les autres, il s'agit simplement de trouver celle qui correspond le mieux aux besoins de l'application en cours de développement. De plus, ne soyez pas surpris si, dans l'un de vos développements, vous utilisez deux philosophies différentes : ce genre de choses se produit assez souvent

et reste en rapport direct avec le premier point : trouver la philosophie qui correspond le mieux à ce que vous souhaitez mettre en place dans votre application.

Commençons par définir la notion de « philosophie » pour nous autres développeurs. Ce que nous appelons philosophie, c'est tout simplement la méthodologie utilisée pour construire une application, en intégrant, au bon moment, les différents acteurs, à savoir les développeurs, les designers, les ergonomes, etc. Chacun d'entre eux intervient à un moment spécifique lors du développement et, par conséquent, cela influence beaucoup le résultat de l'application finale (si le designer est passé trop tôt, il risque d'y avoir des écrans avec une charte graphique différente, si l'ergonome intervient trop tard, il risque de demander à refaire certains des écrans, etc.). L'objectif final est de tenter de gagner en productivité, en évitant de refaire les choses plusieurs fois.

3.1. View First

3.1.1. Définition

Dans une approche View First, c'est la vue, autrement dit le code XAML, qui contrôle le flux de travail de l'application, à savoir la façon dont les différents éléments vont faire référence les uns par rapport aux autres.

La littérature anglo-saxonne parle aussi du View First comme de l'approche *Top Down*, à savoir du haut (la vue) vers le bas (le code et le modèle). Il faut le voir comme une vision en couches par rapport à l'utilisateur final : la vue est ce qu'il voit directement car il interagit avec et il n'a pas connaissance du code qui est « sous le capot » (instanciation et choix des ViewModels, etc.).

4

Construire la partie modèle

Le modèle endosse plusieurs responsabilités et ce chapitre présente les différentes façons de réaliser son implémentation. Cette partie du pattern MVVM est peut être l'une des moins couvertes car au premier abord il est facile de penser qu'elle est identique à celle utilisée dans les précédents patterns (MVP, MVC, etc.). Il en est tout autrement en réalité : l'utilisation des technologies de liaisons révolutionne la façon dont sont conçus les objets métier.

Dans un premier temps ce chapitre présente l'architecture de la couche modèle, puis il explique comment définir le modèle global et créer le modèle client. Enfin, il aborde les techniques possibles pour mettre en place une validation fonctionnelle des données ou encore permettre leur édition d'une façon plus conviviale pour l'utilisateur.

4.1. Architecture du modèle de données

La partie modèle est basée sur trois éléments :

- ✕ *Le modèle global*, indépendant de la partie cliente et qui est utilisé dans le cadre inter-applicatifs.
- ✕ *Le modèle client* – également appelé modèle de présentation – entièrement lié à l'application cliente et qui répond à ses problématiques.
- ✕ *Les services d'accès aux données*, présents sur la partie cliente et faisant le lien entre ces deux types de modèle.

Seuls le modèle client et les services d'accès sont visibles par les ViewModels et les vues. Le modèle client est conçu spécifiquement pour répondre aux différents besoins de la partie cliente (validation, édition, gestion des états, etc.). Il est intimement lié aux ViewModels et aux vues : il y a un couplage fort entre celui-ci et ces deux derniers. Il est tout à fait possible de s'abstraire de ce lien au moyen d'interfaces, mais cela alourdit le développement et il est recommandé de ne le faire que si cela est strictement nécessaire.

La manière dont sont construits et instanciés les objets du modèle de présentation peut être complexe et il est important que cette complexité ne se retrouve pas au sein des ViewModels qui n'ont pas cette vocation. Aussi, ce travail est réalisé au sein des services d'accès aux données qui sont alors utilisés par les ViewModels. Les services sont les intermédiaires entre le monde « extérieur » (c'est-à-dire, là où sont stockées les données) et le monde « intérieur » (c'est-à-dire l'application cliente). Ce sont eux qui implémentent la logique de conversion du modèle global vers le modèle client. Les ViewModels n'ont ainsi aucun besoin de connaître le modèle global puisqu'ils ne doivent et ne vont pas l'utiliser. Cette façon de concevoir les services d'accès aux données est ce que l'on appelle le design pattern *Repository*.

Construire un ViewModel

Le ViewModel est la pierre angulaire du pattern MVVM et nous avons vu précédemment quelles étaient ses différentes fonctions. Dans ce chapitre, nous allons voir comment il peut remplir ces rôles de façon concrète. Dans la suite, nous allons vous présenter une manière bien précise de construire un ViewModel. S'il existe bien sûr d'autres façons de faire, le cas étudié est générique et applicable dans la majorité des situations rencontrées dans un contexte professionnel.

Comme vous pouvez vous en douter, il n'y a pas un seul ViewModel, mais plusieurs, qui sont utilisés conjointement ou indépendamment. Nous étudierons donc dans un premier temps l'architecture utilisée pour construire les ViewModels. Ensuite, nous répondrons aux différentes problématiques relatives à l'exposition des données du modèle et des différentes actions possibles sur le ViewModel. Finalement, nous étudierons comment sont effectuées les communications entre plusieurs ViewModels.

Dans ce chapitre nous considérons que le lecteur possède une connaissance suffisante des différents concepts introduits dans les technologies de construction d'interfaces riches telles que les liaisons de données (*binding*) ou encore le système de commandes.

5.1. Architecture et relations entre les ViewModels

5.1.1. Classe de base des ViewModels

5.1.1.1. Un objet .Net classique

Les ViewModels ne sont pas d'un type particulier, au contraire des objets .Net classiques. Cependant, ils vont tous être utilisés d'une façon bien précise au sein des vues : à l'aide des liaisons de données ou Binding. Celles-ci fonctionnent conjointement avec l'interface `INotifyPropertyChanged` déclarant uniquement l'événement `PropertyChanged`. Aussi, une classe de base nommée `ViewModelBase` va être mise en place et chaque `ViewModel` va en dériver afin de factoriser cette logique de notification des changements de valeurs des propriétés. Cette classe va tout simplement implémenter l'interface `INotifyPropertyChanged` afin de permettre les liaisons de données.

```

/// <summary>
/// Classe de base pour tous les ViewModel
/// </summary>
public class ViewModelBase : INotifyPropertyChanged
{

    #region INotifyPropertyChanged Members
    public event PropertyChangedEventHandler PropertyChanged;

    /// <summary>
    /// Déclenche l'événement PropertyChanged pour une
    /// propriété donnée.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="exp">L'expression permettant
    /// de retrouver la propriété.</param>
    protected void RaisePropertyChanged<T>(Expression<Func<T>> exp)
    {
        var memberExpression = exp.Body as
            MemberExpression;
    }
}

```

Construire les vues

Les deux précédents chapitres ont permis de décrire de façon approfondie les différentes manières d'implémenter les parties modèle et ViewModel. C'est donc sans surprise que ce chapitre va s'intéresser aux vues dans le cadre du pattern MVVM.

Les vues sont les parties émergées de l'iceberg qu'est une application riche. En tant que telle c'est aussi la partie qui va recevoir le plus d'attention des utilisateurs finaux. Le moindre défaut visuel va être remarqué immédiatement alors qu'un manque dans le modèle ou les ViewModel ne sera connu, pour ainsi dire, que des réalisateurs de l'application. Afin d'éviter de glisser sur cette patinoire, il est nécessaire de réaliser à la perfection cette couche du pattern. Ceci est valable à la fois d'un point de vue esthétique et ergonomique, mais aussi d'un point de vue technique pour permettre de faire évoluer facilement l'application. Dans la suite du chapitre, nous nous intéresserons plus au travail des développeurs tout en indiquant les moyens permettant de faciliter le travail d'un graphiste et ou d'un designer.

Dans un premier temps, une définition de ce que sont les vues sous un angle technique sera donnée. L'architecture générale des différents éléments

constituant les vues sera ensuite présentée. À partir de celle-ci nous étudierons comment les vues peuvent utiliser les `ViewModels` de façon simple et comment elles vont répondre à des besoins un peu plus avancés. Finalement, après avoir décrit les utilisations du modèle au sein des vues, des techniques permettant la création de vues sans `ViewModel` au moment de la conception seront décrites.

6.1. Une vue concrètement c'est quoi ?

Définir quels sont les éléments constitutifs d'une vue et surtout où placer le code correspondant est un terrain glissant. C'est en effet un sujet qui fait souvent débat lorsque l'on aborde le pattern MVVM.

Tout comme son ancêtre Delphi, le framework .NET utilise de façon massive des fichiers que l'on appelle le code-behind pour représenter le code relatif aux interfaces graphiques. Ceci est valable pour les technologies web (ASP, ASP.NET), comme pour les technologies de clients lourds et légers (WPF, Silverlight, Windows Phone 7). Le concept est qu'une interface graphique est subdivisée en deux parties : une relative aux éléments graphiques et leur disposition et une autre relative à la gestion des événements et l'interaction avec les autres composants techniques de l'application (les services d'accès aux données et les `ViewModels` en particulier). Le code-behind est cette deuxième partie : il est caché, à l'abri, derrière les éléments graphiques et c'est, en quelque sorte, l'intelligence de la vue. La partie graphique, quant à elle, est représentée dans notre cas dans le format XAML dans le fichier homonyme. Dans Visual Studio, ces deux fichiers vont de pair, le code-behind porte l'extension `.xaml.cs` et le fichier XAML porte l'extension `.XAML`.

L'illustration 6-1 présente une transposition dans le monde réel de ces deux fichiers.

Édition de contrôles personnalisés

Les chapitres précédents se sont attachés à décrire les trois parties du pattern MVVM : le modèle, le ViewModel et finalement la vue. Ce chapitre va quant à lui s'attarder plus en profondeur sur la création de contrôles personnalisés compatibles avec MVVM.

Ce sujet relève plus de la bonne connaissance du framework (WPF, Silverlight ou Windows Phone) qu'uniquement de la bonne compréhension de ce pattern. Cependant, les différentes règles principales à respecter pour faciliter l'utilisation de contrôles personnalisés dans le cadre de MVVM vont être présentées.

Dans un premier temps, les bonnes pratiques seront présentées de façon théorique sous la forme de rappels. Elles seront ensuite illustrées par un exemple pratique : la création d'un contrôle indiquant à l'utilisateur qu'un traitement est en cours. Cet exemple permettra de présenter un à un les différents choix à réaliser ainsi que les éléments nous ayant permis de les faire.

7.1. Création de contrôles personnalisés

Comme cela a été indiqué précédemment, cette partie se focalise sur les différentes techniques utiles lors de la création d'un contrôle personnalisé. Ce sont des techniques utilisables dans un projet n'implémentant pas le pattern MVVM, mais les connaître et les implémenter est une bonne chose car cela laisse une plus grande latitude aux utilisateurs des bibliothèques de contrôles développées. Ces derniers sont pour rappel utilisés par des développeurs, c'est-à-dire un public averti au fait des briques techniques mises en place.

7.1.1. Bien choisir la classe de base

La première chose à faire est de bien choisir la classe de base à utiliser pour le nouveau contrôle à réaliser. En effet, en fonction de celle-ci plusieurs comportements déjà présents dans le framework pourront être réutilisés sans développement supplémentaire. Voici les différents contrôles disponibles (tous héritent de `DependencyObject`) :

- ✕ `Control` : c'est la classe de base présentant le moins de fonctionnalités par défaut, mais c'est aussi celle donnant le plus de liberté. Les éléments suivants dérivent tous de cette classe. Il faut donc la choisir lorsqu'ils ne répondent pas aux besoins du contrôle développé.
- ✕ `ContentControl` : c'est la classe de base à utiliser lorsque l'on souhaite afficher un contenu. Elle contient une propriété `Content` supplémentaire permettant de définir ce contenu, celui-ci pouvant être un objet métier ou un contrôle graphique.
- ✕ `HeaderedContentControl` : cette classe diffère de la précédente de par la présence de la propriété `Header` définissant un titre pour le contenu affiché.

MVVM et testabilité

Ce chapitre a pour objectif de démontrer quelle est l'importance des tests dans l'application du pattern MVVM, ainsi que la manière de les mettre en pratique. Attention, étant donné qu'il ne s'agit pas là d'un livre sur la réalisation de tests, tous les points ne seront pas abordés, mais uniquement ceux qui font sens dans le cadre de l'utilisation de MVVM.

8.1. L'importance des tests

Les tests représentent l'un des points clef du développement d'applications. Cela est valable aussi bien pour les technologies WPF, Silverlight, Windows Phone, que pour les autres types de développements (Web, etc.). L'un des principaux objectifs (et intérêts) d'implémenter le pattern MVVM concerne la simplicité qu'il offre pour réaliser des tests au sein de vos applications.

Il faut savoir qu'il existe trois grandes catégories de tests que l'on peut utiliser (ou non, cela reste à déterminer) dans nos développements. Nous allons donc détailler un peu plus ces trois grandes familles, afin que vous puissiez

mieux vous rendre compte de quel type de tests nous parlons lorsque nous les aborderons dans le cadre de MVVM.

8.1.1. Les tests fonctionnels

Les tests fonctionnels sont exécutés par des personnes qui ne sont pas des techniciens et dont le rôle est de s'assurer que les fonctionnalités de l'application ont correctement été implémentées d'un point de vue métier. Ainsi, c'est le rôle des testeurs fonctionnels de vérifier que lorsque l'utilisateur va cliquer sur un bouton, une grille sera correctement remplie (avec les bonnes données) et qu'une boîte de dialogue sera bien proposée à l'utilisateur.

En cas d'échec d'un test fonctionnel, il n'est pas de leur responsabilité de savoir pourquoi les données remontées ou affichées ne sont pas correctes : ils se contentent de dérouler un scénario de tests, d'effectuer des tests de non-régression et d'indiquer si le test s'est correctement déroulé ou non.

Les tests de non-régression permettent, entre deux versions d'une application, de s'assurer que les fonctionnalités présentes dans la V1 sont toujours présentes dans la V2 et qu'elles fonctionnent sans problèmes.

Les tests fonctionnels sont donc très pratiques et permettent aux testeurs/développeurs de se mettre, l'espace d'un instant, à la place de l'utilisateur final, afin de pouvoir se rendre compte des points forts, des points faibles, etc., de l'application. C'est d'ailleurs bien souvent un utilisateur final qui effectue ces tests car c'est lui qui a bien souvent la meilleure connaissance des besoins de l'application.

8.1.2. Les tests unitaires

Les tests unitaires vont permettre aux développeurs de s'assurer de la cohérence des données présentes dans l'application. Alors que le testeur fonctionnel a pu remarquer que les données remontées ne sont pas correctes (pas le bon nombre d'enregistrements, etc.), c'est au développeur, avec ses tests unitaires, de vérifier et comprendre d'où provient cette inexactitude.

Glossaire

Binding

En français « liaison », un binding correspond à un élément fréquemment utilisé dans les frameworks WPF et Silverlight. Il permet de synchroniser la valeur des propriétés de deux objets : la source et la cible. Il définit un lien entre ces deux éléments sans nécessairement que ceux-ci ne se connaissent : on parle alors de *couplage faible*.

Contrôle

Un contrôle est un élément graphique réutilisable dans une interface graphique. Le framework .Net en fournit toute une panoplie, tels que les champs textes, les images, etc., mais il est tout à fait possible d'en créer de nouveaux. Une vue d'une application est ainsi composée de ces contrôles disposés les uns à côtés des autres.

Design Pattern

On appelle « Design Pattern » un patron de conception, c'est-à-dire une solution de génie logiciel permettant de résoudre un problème récurrent dans le cadre de la programmation. Un Design Pattern peut être considéré comme une recette de cuisine.

Injection de dépendances

L'injection de dépendances est un mécanisme permettant de mettre en place l'inversion de contrôle. Cela consiste à fournir les dépendances entre les différentes classes par rapport à un fichier de configuration ou suivant certaines règles de localisation. Les ressources sont fournies de l'extérieur vers le demandeur.

Localisateur de services

Le localisateur de services s'utilise conjointement avec l'injection de dépendances. C'est un mécanisme permettant d'obtenir une ressource. À la différence de l'injection de dépendances, c'est le demandeur qui réclame explicitement ce dont il a besoin.

MVVM

C'est un acronyme anglais signifiant *Model View Model*. En lisant ce livre, vous aurez une bonne idée de ce que c'est concrètement.

Ressources

On désigne par ressources tout objet disponible à un moment donné dans un programme. Dans les frameworks WPF et Silverlight ce sont la plupart du temps des images, des pinceaux (Brush) ou encore des classes spécialisées (Converter, etc.).

Test unitaire

C'est un procédé informatique permettant de s'assurer du bon fonctionnement d'un extrait de code. Il est appelé unitaire car il permet de vérifier une « unité de code » consistant en le plus petit élément de spécification à vérifier.

XAML

Signifiant *eXtensible Application Markup Language*, le XAML est un langage déclaratif basé sur XML permettant de définir une interface graphique WPF ou Silverlight.

Index

A

Actions 148. *Voir aussi Commandes*
exposer 149

Actions de l'utilisateur 201

Afficher
contenu 245
informations de validation 222
liste d'éléments 246
modèle au sein des vues 27

Annuler l'action de l'application 265

Appels asynchrones 95

Approches de MVVM 42

Architecture des vues 171

Assembly 83

Assert 284

AsyncReponse 98

Attached Behavior 211

AutoEdit 276

AutoMapper 77

B

BeginEdit 113

Behaviors Blend 154, 207

Bindings 73, 168, 289
avancé 217
lier une collection 197
sur SelectedItems 217
utilisation 193

Blendabilité 47

BooleanToVisibilityConverter 199

Bouchons (tests) 280

Bubbling 248

ButtonBase 202

C

Caliburn 10

CaliburnMicro 17

CallMethodAction 207

CancelEdit 113

CanExecute 149, 202

CanExecuteChanged 149

- Cast 126
 - CategoriesListViewModel 137
 - ChargerItems 277
 - Cider 238
 - Cinch 17
 - ClassCleanup 278
 - Classe de base du modèle client 72
 - ClassInitialize 277
 - Code-behind 15, 45, 167
 - utilité 169
 - Coded UI Test 285
 - Collections
 - mise en forme 231
 - synchronisation 221
 - CollectionViewSource 143
 - Command 94, 201, 202
 - Commandes 149. *Voir aussi Actions*
 - classiques 254
 - exemple concret 153
 - implémenter 150
 - routées 251
 - sur des contrôles graphiques classiques 202
 - système 148
 - améliorer 155
 - CommandParameter 202
 - CommandTarget 202
 - Communication entre ViewModels 156
 - exemple concret 162
 - Composition 127
 - Construire les vues 166–243
 - Construire un ViewModel 120
 - ContentControl 245
 - ContentPresenter 259
 - Contrats fonctionnels 23, 83
 - Control 245
 - Contrôles 289
 - choix 35
 - de mise en attente 256
 - Contrôles personnalisés 244–270.
 - Voir aussi Commandes*
 - classe de base 245
 - créer 245
 - événements 248
 - exemple 256
 - fichiers 246
 - implémenter 267
 - propriétés 247
 - rendu visuel 254
 - Controller
 - MVC 12
 - Convert 228
 - ConvertBack 228
 - Converters 193
 - conversion de données 200
 - formatage de données 201
 - utiliser 199
 - Couplage faible 23
 - Créer le modèle client 72
 - CRUD 60
 - CurrentItem 147
 - CustomControl 35
- ## D
- Data-Centric 60
 - DataContext 45, 180
 - DataTemplate 183
 - implicites 61
 - DataTriggers 232

- d:DataContext 238
- d:DesignData 239
- d:DesignInstance 239
- Déclencheur 207
- Définir le modèle
 - client 75
 - global 68
 - héritage 71
- Delegate 248
- DeliverEvent 215
- DependencyObject 122
- Design 45
- Design Pattern 289
- Design time 47, 238
- Développement 44
- Dictionnaires 230
- Données
 - de tests 241
 - exposer 136
 - filtrer 142, 144
 - grouper 145
 - trier 146
- DoubleClickTrigger 209
- DynamicResource 173
- E**
- Écrans
 - fonctionnels 181
 - prédéfinis d'interaction 186
- Édition des données 113
- Éléments du pattern 23
- Empreinte mémoire de l'application 177
- Encapsulation 138
- EndEdit 113
- EntityFramework 71, 86
- Enum 225
- EnumToDescriptionConverter 226
- Ergonomie 44
- Erreurs transverses 112
- Évènements
 - classiques 250
 - contrôles personnalisés 248
 - écoute 209
 - faibles 215
 - routés 248
- EventArgs 248
- EventManager 249
- EventTrigger 207, 209
- EventTriggerBase<T> 209
- Execute 149
- Exposer
 - le modèle 137
 - les différentes entités 136
 - une action 149
- Expression Blend 208
- F**
- Fenêtres de dialogue 186
- Filter 144
- FindName 255
- Fowler, Martin 10
- Frameworks 16
- G**
- Garbage collector 160
- Gossman, John 11
- GoToState 236

Gresh, Marlon 159

GroupDescription 145

H

HeaderedContentControl 245, 258

Héritage de ViewModels 124

Hyperlink 202

I

ICategorieService 277

ICollectionView 143

ICommand 149, 150

ICommandSource 201

IDataErrorInfo 101, 112

Identifiant unique de l'enfant
vers le parent 71

IEditableObject 113

IList<> 85

IMedia 71

IMessenger 160

IMultiValueConverter 199

Indexer 230

InitialisationDesServices 277

Injection de dépendances 52,
132, 134, 290

Injection par constructeur 53

INotifyCollectionChangedWeakE-
ventManager 218

INotifyPropertyChanged 73, 121

InputBinding 202

InputsBindings 203

Instances de données 239

Interaction avec l'utilisateur 186

confirmation 187

contrôle d'attente 256

message 187

valider, annuler 187

Interfaces

définir 83

graphiques

tests 273

réactives 232

internal 250

Inversion de Contrôle 124, 133

solutions 134

IoC 52

IsDesignTimeCreatable 239

IServiceBase<> 84

IsSynchronizedWithCurrentItem 197

ItemsControls 246

ItemsSyncher 218

ItemSyncher 220

IValueConverter 199

K

KeyDownManager 215

L

Liaisons de données. *Voir Bindings*

LocalCanExecuteEvent 155

Localisateur de services 66, 290
mise en place 134

Localisation de services 134

Logique fonctionnelle 263

M

MainViewModel 129, 180

Mapper 76

Mapping 76

avantages et inconvénients 77

MarkupExtension 238

Médiateur 158

Médiator 158

Mémoire 177

fuites 214

MenuItem 202

MergedDictionaries 174

MessageToActionsMap 161

Messenger 160

Microsoft Expression Blend 238

Mock 280

Model 22

Modèle

acteurs 27

client 25, 65

classe de base 72

créer 72

définir 75

valeurs 113

client/consommateur 194

construire 64–119

de données 22, 24, 65

rôle 24

de la vue 22

de présentation 25

exposer 137

global 65

définir 68

tests 274

utilisation par

la vue 26

le ViewModel 26

utiliser 222

Model First 59

avantages 61

définition 59

implémenter 59

inconvénients 61

type d'applications 60

Modélisation de la vue 36

Model-View-Controller 11

Model-View-Presenter 13

Model-View-ViewModel 9

MoveCurrentTo 147

MoveCurrentToFirst 147

MoveCurrentToNext 147

MoveCurrentToPrevious 147

Multi-threading 134

MVC 11

MVP 13

MVVM 290

appliquer 42

approches 42

choisir 62

avantages et particularités 38

code-behind 15

contrôles, choix 35

éléments 23

fonctionnalités 18

frameworks 16

objectifs du projet 14

philosophies 42

choisir 62

prérequis 19

présentation 9–21

MVVM Light Toolkit 17

N

Namespace XML 267

NInject 52

O

Objets

- appeler une méthode sur 207

- métiers

 - représentation 183

OnApplyTemplate 254, 264

OneTime 193

OneWay 193

OneWayToSource 193

P

Passive View 14

Philosophies 42

Plain Old CLR Object 69

POCO 69

Presentation Model 10

Presenter 13, 31

Prism 17

PropertyChanged 81, 121

PropertyChanging 73

Propriétés de dépendance

- contrôles personnalisés 247

R

RAD 60

RadioButton 202

RaisePropertyChanged 73

Ramasse-miettes 161

Ramora 210

- avantages 211

- implémenter 212

Rapid Application Development 60

Réagir aux actions de l'utilisateur 201

Règles métier 110

Régression 273

RelayCommand 150

Repository 83

Représentations graphiques du

- modèle de présentation 183

ResourceDictionary 173

Ressources graphiques 290

- emplacement 174

- transverses 172

- types de 173

RoutedEvent 249

RoutedEventArgs 248

Runtime 47

S

SelectedItem 217

SelectedItemSyncher 218

SelectionChangedEventManager 218

service d'affichage 186

ServiceLocator 134, 136, 137, 277

Services d'accès aux données 65, 140

- abstraction 83

- blocage de l'affichage 96

- concevoir 83

- implémenter 86

- sous la forme d'interfaces 66

- Unit Of Work 93

ShellViewModel 129

Silverlight 281
Silverlight Test Class 282
Silverlight Unit Test Application 282
Smalltlak 11
Smith, Josh 159
SortDescriptions 146
Spring 52
StartListening 215
StaticResource 173
StopListening 215
String.Empty 81
Suivre l'élément courant 146
Supervising Controller 14
SynchronizationContext 97
System.ComponentModel.
 DataAnnotations 104
Système de commandes 148
 améliorer 155

T

TabControl 179
TestClass 284
TestCleanup 278
TestInitialize 278
TestMethod 284
Tests 271
 de non-régression 272
 fonctionnels 272
 interfaces graphiques 273
 Modèle 274
 unitaires 272
 bouchons 280
 initialiser 276
 ViewModel 275

 avec Silverlight 281
 avec WPF 275
 vues 285
Tests unitaires 290
 erreur (Silverlight) 285
 exception 277
 View First 50
ThreadPool 98
Thread UI 95
ToggleButton 202
Top Down 43
Transitions
 avec animations 233
 sans animation 232
Trigger 207
TryValidateProperty 106
Tunneling 248
TwoWay 193
Types particuliers 225

U

Unit Of Work 92
 implémenter dans les services 93
Unity 52
UserControl 35, 188

V

Valeurs du modèle client 113
Valeurs énumérées 225
ValidatesOnDataErrors 222
Validation des données 100
 fonctionnelle 110
 format 101
 types de 100

- Validation.HasError 223
 - Validation, informations de 222
 - Validator 106
 - View 22
 - View First 43
 - avantages 48
 - définition 43
 - implémenter 45
 - inconvénients 49
 - instanciation des objets 44
 - tests unitaires 50
 - ViewModel 22, 36
 - acteurs 40
 - architecture et relation 121
 - classe de base 121
 - communication 156
 - construire 120
 - fonctionnalités 129
 - héritage 124
 - rôles 36
 - utilisation simple 193
 - ViewModelBase 121
 - ViewModel First 51
 - avantages 56
 - définition 51
 - implémenter 52
 - inconvénients 56
 - type d'applications 58
 - ViewModelLocator 46, 177
 - alternative à 47
 - VisualStateManager 234
 - Visual Studio 2010
 - tests 273
 - Visual Studio LightSwitch 59
 - Vues 22, 28–30
 - acteurs 33
 - architecture 171
 - catégories 172
 - construire 166–243
 - définition 167
 - maître/détails 182
 - manipulable en XAML 48
 - naviguer entre 58
 - rôles 30
 - sans ViewModel concret 238
 - ViewModel et 31
- ## W
- WeakAction 161
 - WeakEventListener 215
 - WeakEventManager 215
 - WeakEvents 215
 - WeakReference 161
 - Web Forms 21
 - Window 188
 - Windows Forms 20
 - Windows Phone
 - tests 274
 - Workarounds 62
 - WPF
 - tests 275
 - Wrapping 80
 - avantages 80
 - inconvénients 80
- ## X
- XAML 45, 168, 290
 - x:Code 170
 - x:Key 173
 - x:Shared 48