MVVM

Architecture et Implémentation Robuste

Objectifs du jour

- Maîtriser le pattern **MVVM** en profondeur
- Implémenter **INotifyPropertyChanged** efficacement
- Créer des **commandes** synchrones et asynchrones
- Utiliser le **Data Binding** avancé avec validation
- Migrer vers **CommunityToolkit.Mvvm**

Partie 1

Pattern MVVM

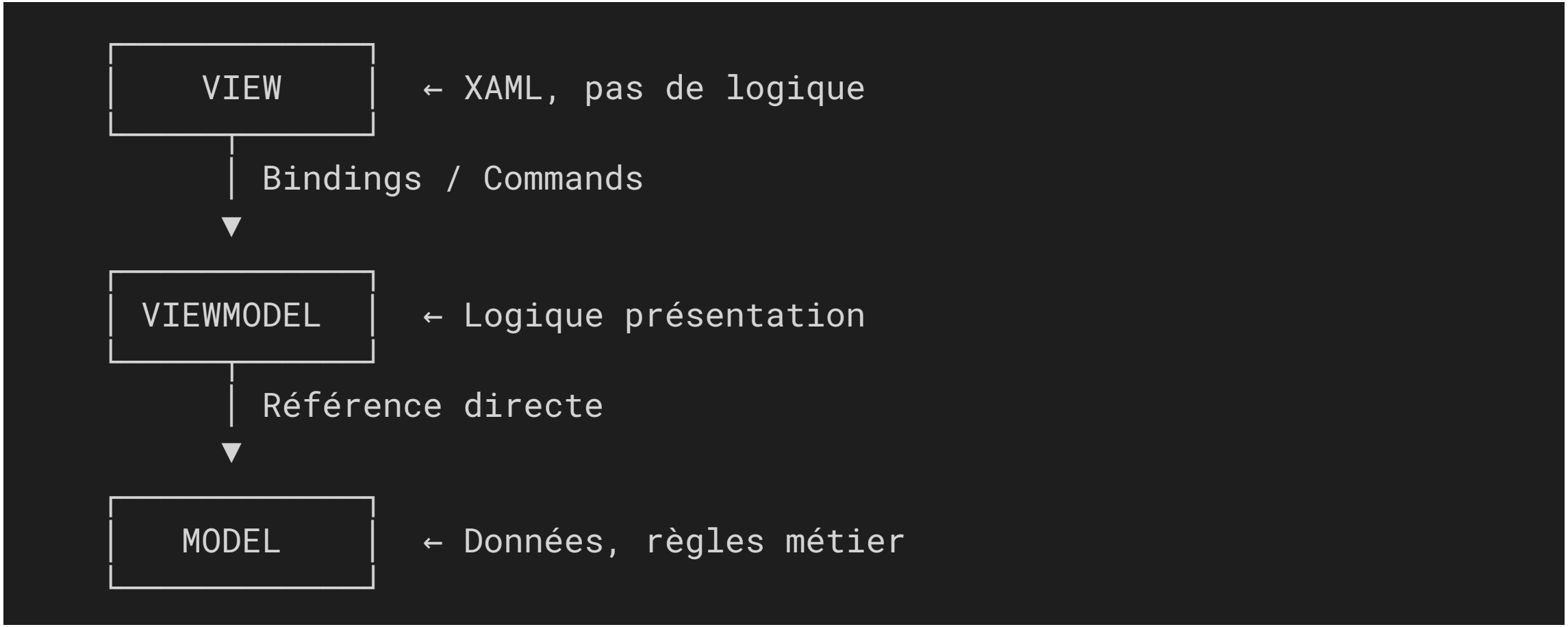
Qu'est-ce que MVVM ?

Model-View-ViewModel - Pattern architectural

Composant	Responsabilité
Model	Données et logique métier
View	Interface utilisateur (XAML)
ViewModel	Logique de présentation

> La View ne connaît que le ViewModel (via DataContext)

Flux de communication



Règles fondamentales

1. La **View** ne connaît que le ViewModel
2. Le **ViewModel** ne référence **jamais** la View
3. Le **Model** est indépendant de la présentation
4. Communication via **Bindings**, **Commands**, **Messaging**

> Objectif : **testabilité** et **maintenabilité**

Avantages de MVVM

Avantage	Description
Testabilité	ViewModels testables sans UI
Maintenabilité	Séparation claire des responsabilités
Réutilisabilité	ViewModels indépendants
Collaboration	Designer/Dev en parallèle
Blendability	Support outils de design

Partie 2

INotifyPropertyChanged

Le problème

Comment la View sait-elle qu'une propriété a changé ?

Sans notification :

- La View affiche la valeur initiale
- Les modifications ne sont pas reflétées
- Le binding est "mort"

> `INotifyPropertyChanged` résout ce problème

Interface INotifyPropertyChanged

Un seul membre :

```
event PropertyChangedEventHandler? PropertyChanged;
```

Notification :

- Déclencher l'événement avec le nom de la propriété
- La View met à jour automatiquement

> 📂 **Démo** : `Demo-MVVM-Base`

Implémentation de base

Pattern SetProperty :

1. Comparer ancienne et nouvelle valeur
2. Si différentes : assigner + notifier
3. Retourner `true` si changé

> Utiliser `[CallerMemberName]` pour éviter les erreurs

Erreurs courantes

❌ Erreur	✅ Correction
Pas de comparaison	Comparer avant notifier
Notification dans getter	Notifier dans setter uniquement
Oublier les propriétés calculées	Notifier manuellement
Typo dans le nom	Utiliser `nameof()`

Propriétés calculées

Une propriété qui dépend d'autres propriétés :

```
NomComplet = Prenom + " " + Nom
```

À notifier quand :

- `Prenom` change → notifier `NomComplet`
- `Nom` change → notifier `NomComplet`

> Double notification nécessaire

Partie 3

ICommand et RelayCommand

Interface ICommand

3 membres :

Membre	Rôle
`Execute(object?)`	Exécute l'action
`CanExecute(object?)`	Peut-on exécuter ?
`CanExecuteChanged`	Notification de changement

> Lie les boutons aux actions du ViewModel

RelayCommand

Implémentation réutilisable de ICommand

- Prend une `Action` pour Execute
- Prend une `Func<bool>` pour CanExecute
- `RaiseCanExecuteChanged()` pour rafraîchir

“ 📂 **Démo** : `Demo-Commands` ”

RelayCommand avec paramètre

Générique : `RelayCommand<T>`

- Le paramètre vient du `CommandParameter`
- Conversion automatique

> Utile pour supprimer un élément d'une liste

AsyncRelayCommand

Pour les opérations longues :

- `Execute` appelle une méthode `async`
- `IsExecuting` pour afficher un loader
- Empêche les exécutions concurrentes
- Support de l'annulation

> Ne jamais bloquer le thread UI !

Notifications CanExecute

Quand rafraîchir ?

- Quand une propriété liée change
- Après une action modifiant l'état
- `CommandManager.InvalidateRequerySuggested()` (global)

> Préférer la notification explicite

Partie 4

Data Binding avancé

Modes de binding

Mode	Direction
`OneWay`	Source → Target
`TwoWay`	Source ↔ Target
`OneWayToSource`	Target → Source
`OneTime`	Une seule fois

> Le mode par défaut dépend du contrôle

UpdateSourceTrigger

Quand mettre à jour la source ?

Valeur	Quand
`PropertyChanged`	À chaque frappe
`LostFocus`	Perte de focus (défaut TextBox)
`Explicit`	Manuellement

> `PropertyChanged` pour recherche temps réel

Validation : INotifyDataErrorInfo

Interface moderne pour la validation

Membre	Rôle
`HasErrors`	Y a-t-il des erreurs ?
`GetErrors(propertyName)`	Erreurs pour une propriété
`ErrorsChanged`	Notification de changement

> Préférer à `IDataErrorInfo` (legacy)

Data Annotations

Attributs de validation standard :

- `[Required]` - Champ obligatoire
- `[StringLength]` - Longueur min/max
- `[Range]` - Plage de valeurs
- `[EmailAddress]` - Format email
- `[RegularExpression]` - Pattern personnalisé

> 📂 **Démo** : `Demo-Validation`

IValueConverter

Transforme les valeurs lors du binding

Méthode	Direction
`Convert`	Source → Target
`ConvertBack`	Target → Source

Exemples : Bool→Visibility, Enum→String, Date→Format

IMultiValueConverter

Combine plusieurs valeurs en une seule

```
<MultiBinding Converter="{StaticResource FullNameConverter}">
    <Binding Path="Prenom"/>
    <Binding Path="Nom"/>
</MultiBinding>
```

> Utile pour les propriétés calculées en XAML

Debugging du binding

Outils de diagnostic :

Outil	Usage
Output Window	Messages d'erreur binding
`PresentationTraceSources.TraceLevel`	Niveau de détail
DebugConverter	Point d'arrêt dans conversion

> Toujours vérifier la fenêtre Output !

Partie 5

CommunityToolkit.Mvvm

Pourquoi CommunityToolkit ?

Bibliothèque officielle Microsoft

Avantage	Description
Léger	Aucune dépendance
Performant	Générateurs de source
Moderne	C# 10+ features
Maintenable	Moins de boilerplate

> NuGet : `CommunityToolkit.Mvvm`

Générateurs de source

Le code est généré à la compilation

```
[ObservableProperty]          →    public string Name
private string _name;              {get; set;} + INPC
```

> Classe **obligatoirement** `partial`

[ObservableProperty]

Remplace les propriétés manuelles :

- Génère le getter/setter
- Génère la notification
- Supporte les callbacks `On*Changed`

> 📂 **Démo** : `Demo-Toolkit`

Notifications automatiques

Attribut	Effet
`[NotifyPropertyChangedFor]`	Notifie une autre propriété
`[NotifyCanExecuteChangedFor]`	Rafraîchit une commande

> Plus de notification manuelle !

[RelayCommand]

Génère la commande depuis une méthode :

- Méthode `void` → `IRelayCommand`
- Méthode `async Task` → `IAsyncRelayCommand`
- `CanExecute` via attribut

> Plus d'instanciation dans le constructeur

Options avancées

Option	Effet
`CanExecute = nameof(...)`	Méthode CanExecute
`AllowConcurrentExecutions`	Autoriser exécutions parallèles
`IncludeCancelCommand`	Générer commande d'annulation

> Gestion native de l'async et annulation

ObservableValidator

Validation avec Data Annotations

- Hériter de `ObservableValidator`
- Ajouter `[NotifyDataErrorInfo]` sur les propriétés
- `ValidateAllProperties()` pour valider tout

> Validation automatique à chaque changement

WeakReferenceMessenger

Communication découplée entre ViewModels

Méthode	Usage
`Register<T>`	S'abonner
`Send<T>`	Envoyer
`Unregister`	Se désabonner

> Références faibles = pas de fuite mémoire