Bonnes Pratiques & .NET 9

Architecture Moderne WPF

Objectifs du jour

- Implémenter l'**injection de dépendances**
- Écrire des **tests unitaires** pour ViewModels
- Optimiser les **performances** WPF
- Maîtriser les **patterns de navigation**
- **Migrer** vers .NET 9

Partie 1

Injection de Dépendances

Pourquoi l'injection de dépendances ?

Sans DI	Avec DI
Couplage fort	Couplage faible
Tests difficiles	Tests faciles (mocks)
Dépendances cachées	Dépendances explicites
Code rigide	Code flexible

> La DI est essentielle pour une architecture testable

Microsoft.Extensions.DependencyInjection

Package officiel Microsoft - léger et performant

Lifetime	Usage
`Singleton`	Une instance unique
`Scoped`	Par scope (request)
`Transient`	Nouvelle instance à chaque fois

> 📂 **Démo :** `Demo-DI-Setup`

Configuration dans App.xaml.cs

Étapes de configuration :

1. Créer le `ServiceCollection`
2. Enregistrer les services
3. Construire le `ServiceProvider`
4. Résoudre la fenêtre principale

> Le ServiceProvider est le conteneur DI

Enregistrement des services

Méthode	Usage
`AddSingleton<T>`	Services partagés
`AddTransient<T>`	ViewModels, Services légers
`AddDbContext<T>`	Entity Framework

> Toujours programmer contre des **interfaces**

Résolution dans les ViewModels

Injection par constructeur :

- Le conteneur injecte automatiquement
- Dépendances déclarées explicitement
- Facilite les tests avec mocks

> 📂 **Démo** : `Demo-DI-ViewModels`

Partie 2

Tests Unitaires

Pourquoi tester les ViewModels ?

Avantage	Description
Confiance	Code validé automatiquement
Refactoring	Changements sans régression
Documentation	Tests comme exemples
Design	Force une bonne architecture

> Les ViewModels MVVM sont facilement testables

Frameworks de test

Framework	Rôle
xUnit	Framework de test
Moq	Création de mocks
FluentAssertions	Assertions lisibles

> “ xUnit est le standard pour .NET moderne ”

Structure d'un test

Pattern AAA :

Phase	Action
Arrange	Préparer le contexte
Act	Exécuter l'action
Assert	Vérifier le résultat

> 📂 **Démo :** `Demo-Tests-ViewModels`

Mocker les dépendances

Moq permet de simuler les interfaces :

- `Setup()` - Définir le comportement
- `Returns()` - Valeur de retour
- `Verify()` - Vérifier les appels

> Isoler le ViewModel de ses dépendances

Tester les commandes

Vérifications importantes :

1. `CanExecute` retourne la bonne valeur
2. `Execute` modifie l'état correctement
3. `CanExecuteChanged` est déclenché

> Les commandes encapsulent la logique testable

Tester les notifications

INotifyPropertyChanged :

- S'abonner à `PropertyChanged`
- Vérifier les propriétés notifiées
- Utiliser des helpers de test

> 📂 **Démo** : `Demo-Tests-INPC`

Partie 3

Performance WPF

Virtualization

Problème : Afficher 10 000 éléments

Sans virtualisation	Avec virtualisation
10 000 contrôles	~20 contrôles visibles
Lent au démarrage	Instantané
Mémoire élevée	Mémoire constante

> Activer `VirtualizingPanel.IsVirtualizing`

Types de virtualisation

Mode	Description
Standard	Recycle les conteneurs
Deferred	Scrolling différé
Recycling	Réutilise les conteneurs

> 📂 **Démo :** `Demo-Virtualization`

Binding Performance

Optimisations :

Technique	Impact
`OneTime` binding	Pas de tracking
`x:Static`	Résolution compile-time
Éviter `StringFormat` complexe	Moins de parsing

> Chaque binding a un coût

Async/Await dans WPF

Règles importantes :

1. Ne jamais bloquer le thread UI
2. Utiliser `async Task` (pas `async void`)
3. `ConfigureAwait(false)` pour le travail CPU

> 📂 **Démo** : `Demo-Async-Loading`

Détecter les problèmes

Outil	Usage
Visual Studio Profiler	Performance générale
WPF Performance Suite	Rendu et layout
PerfView	Analyse approfondie

> Mesurer avant d'optimiser !

Partie 4

Navigation et Dialogues

Pattern de Navigation

Service de navigation découplé :

Composant	Rôle
`INavigationService`	Interface
`NavigationService`	Implémentation
`MainViewModel`	Conteneur de vue courante

> Le ViewModel ne connaît pas les Views

Navigation par ViewModel

Approche recommandée :

1. Naviguer vers un type de ViewModel
2. Le service résout la View appropriée
3. DataTemplates associent View ↔️ ViewModel

> 📂 **Démo** : `Demo-Navigation`

Service de dialogues

Abstraction des dialogues :

Méthode	Usage
`ShowMessage`	Information
`ShowConfirmation`	Oui/Non
`ShowError`	Erreur
`ShowDialog<T>`	Dialogue personnalisé

> Permet de tester sans afficher de dialogues

Messaging entre ViewModels

WeakReferenceMessenger (CommunityToolkit) :

Opération	Méthode
S'abonner	`Register<TMessage>`
Publier	`Send<TMessage>`
Se désabonner	`Unregister`

> Communication découplée, pas de fuite mémoire

Partie 5

Migration .NET 9

Pourquoi migrer ?

Avantage	Détail
Performance	JIT amélioré, moins de mémoire
C# 12-13	Nouvelles fonctionnalités
Support	LTS jusqu'en 2027
Écosystème	Packages modernes

> .NET Framework n'évolue plus

Upgrade Assistant

Outil officiel Microsoft :

1. `dotnet tool install -g upgrade-assistant`
2. `upgrade-assistant analyze MonProjet.csproj`
3. `upgrade-assistant upgrade MonProjet.csproj`

> 📂 **Démo** : `Demo-Migration`

Conversion du .csproj

Avant : Format XML verbeux (100+ lignes)

Après : Format SDK (~10 lignes)

Élément	Changement
`TargetFramework`	`net9.0-windows`
`UseWPF`	`true`
Références	PackageReference

Configuration moderne

Ancien	Nouveau
`App.config`	`appsettings.json`
`ConfigurationManager`	`IConfiguration`
Sections XML	JSON structuré

> 📂 **Démo :** `Demo-Configuration`

Packages à migrer

.NET Framework	.NET 9
EntityFramework 6	EF Core 9
System.Configuration	Microsoft.Extensions.Configuration
Unity/Ninject	Microsoft.Extensions.DependencyInjection

Nullabilité

Activation : `<Nullable>enable</Nullable>`

Pattern	Usage
`string?`	Nullable explicite
`required`	Propriété obligatoire
`= string.Empty`	Valeur par défaut

> “ Élimine les NullReferenceException ”

Partie 6

C# 12-13 Features

Primary Constructors

Avant : Constructeur + champs + assignation

Après : Paramètres dans la déclaration de classe

Avantage	Détail
Moins de code	Supprime le boilerplate
Immutabilité	Paramètres readonly

“ 📂 **Démo** : `Demo-CSharp12` ”

Collection Expressions

Syntaxe unifiée pour les collections :

Ancien	Nouveau
`new List<int> { 1, 2 }`	`[1, 2]`
`Array.Empty<int>()`	`[]`
`list.ToArray()`	`[..list]`

> Plus lisible et performant

Required Members

Garantir l'initialisation :

Mot-clé	Effet
`required`	Doit être initialisé
`init`	Settable à l'init seulement

> Remplace les constructeurs volumineux

Raw String Literals

Multi-ligne sans échappement :

```
"""
Du texte sur
plusieurs lignes
sans \n ni \"
"""
```

> Idéal pour SQL, JSON, HTML