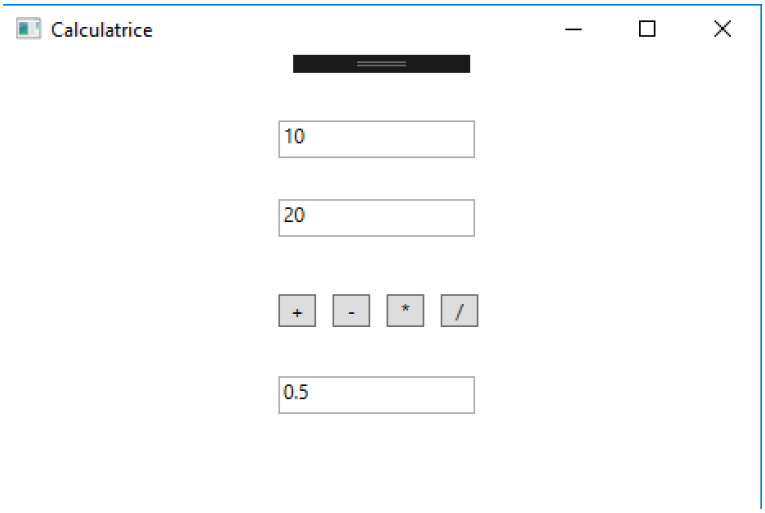
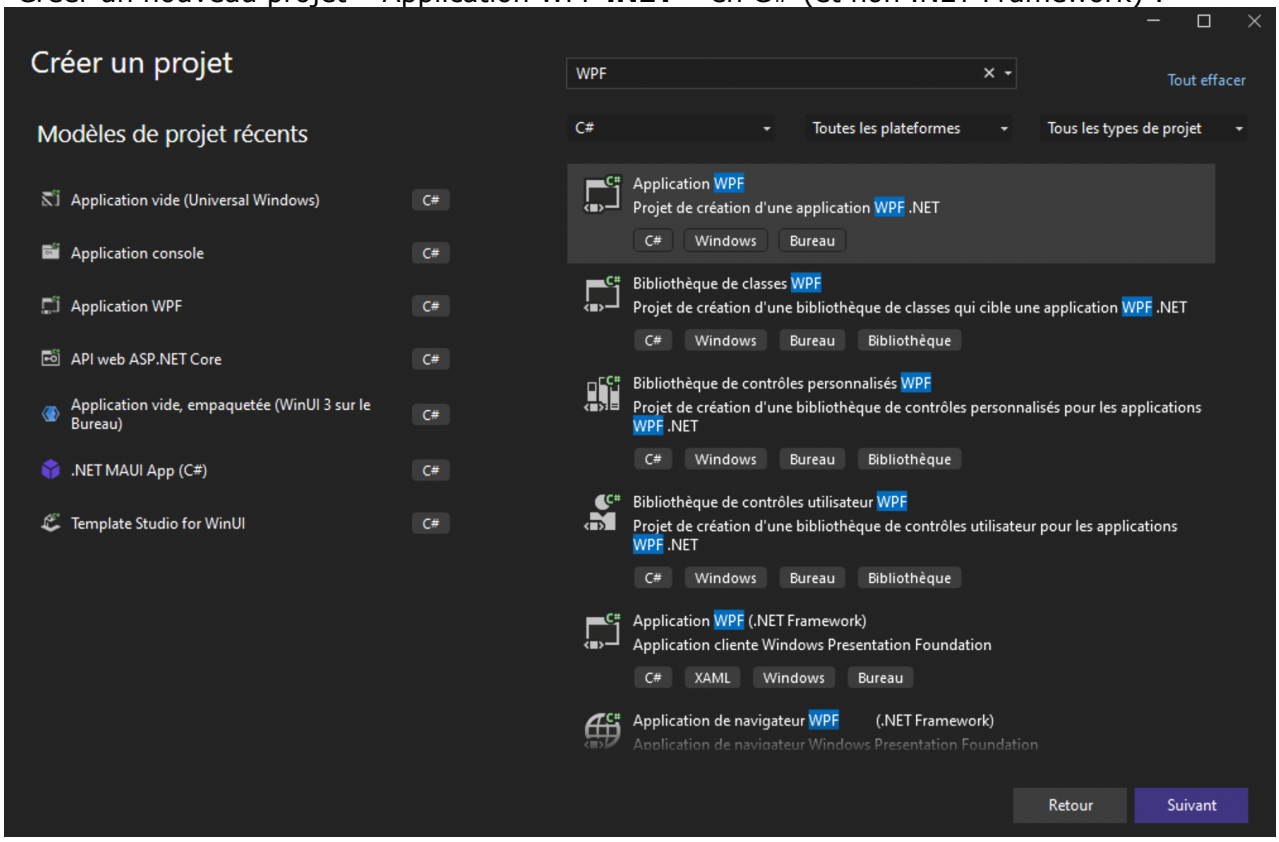
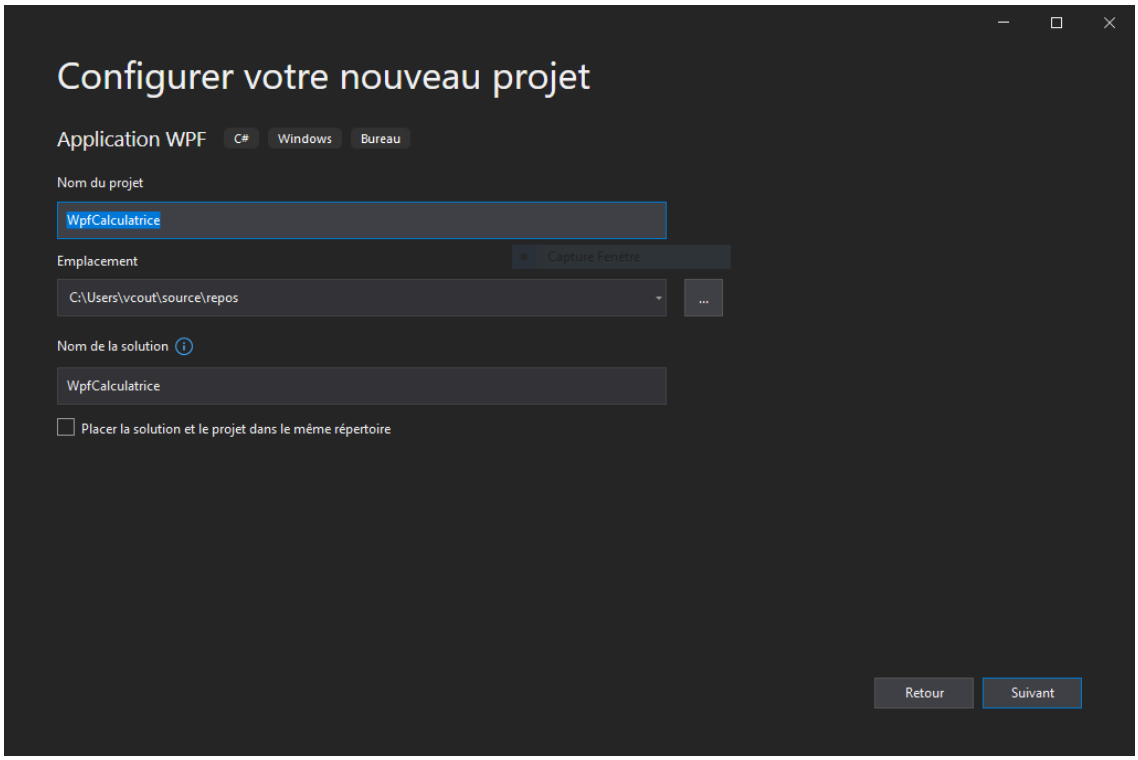
1 Révisions Binding C# : Création d'une calculatrice

Créer une calculatrice prenant en entrée 2 nombres réels affichant le résultat d'une addition, soustraction, multiplication ou division.

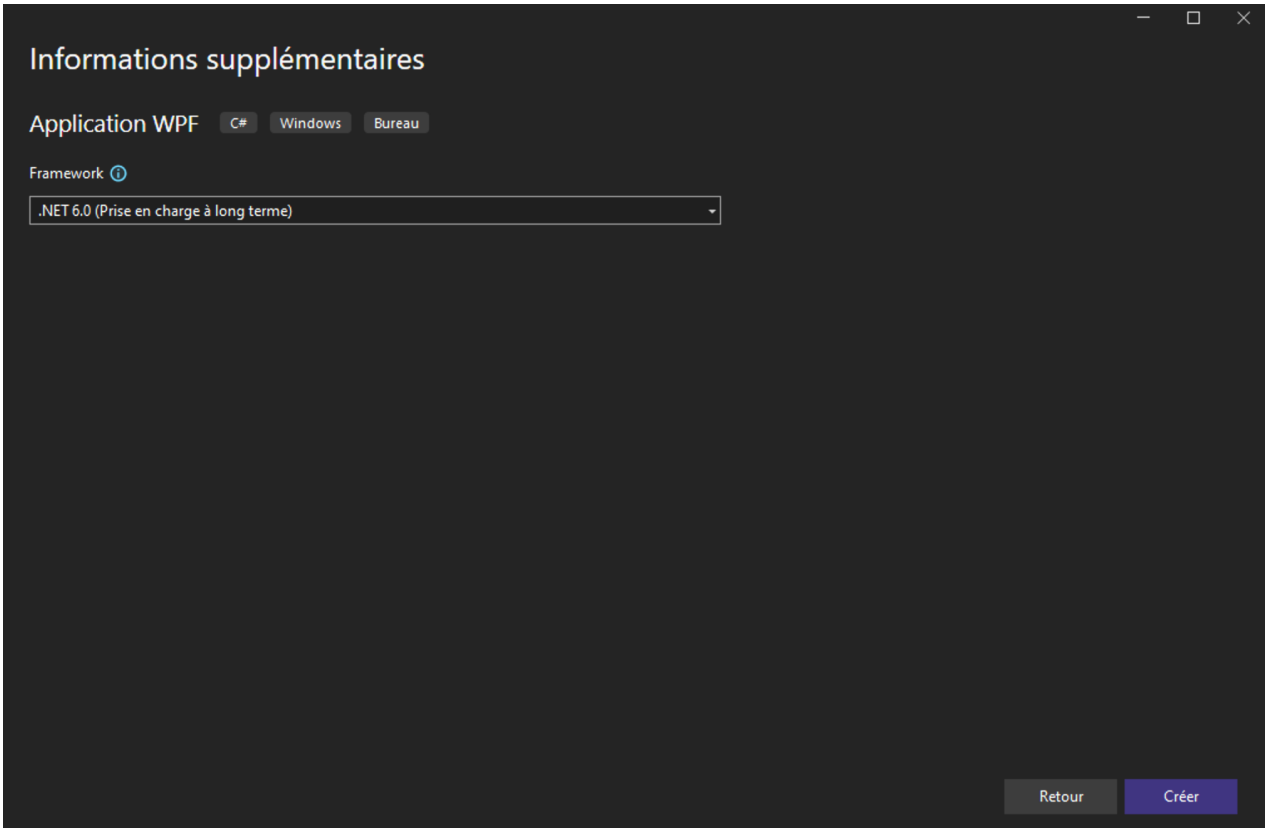


- Créer un nouveau projet « Application WPF **.NET** » en C# (et non .NET Framework) :





Utiliser la version du framework qui vous est proposée par défaut (normalement .NET 6 qui est la version Long Time Support).



- Fichier XAML : utiliser un contrôle `stackpanel` pour positionner les boutons.
  Le TextBox résultat sera readonly.

Coder l'application. Le code de la calculatrice (les 3 Textbox) sera réalisé en utilisant le binding. Cf. CM pour le rappel sur le binding.

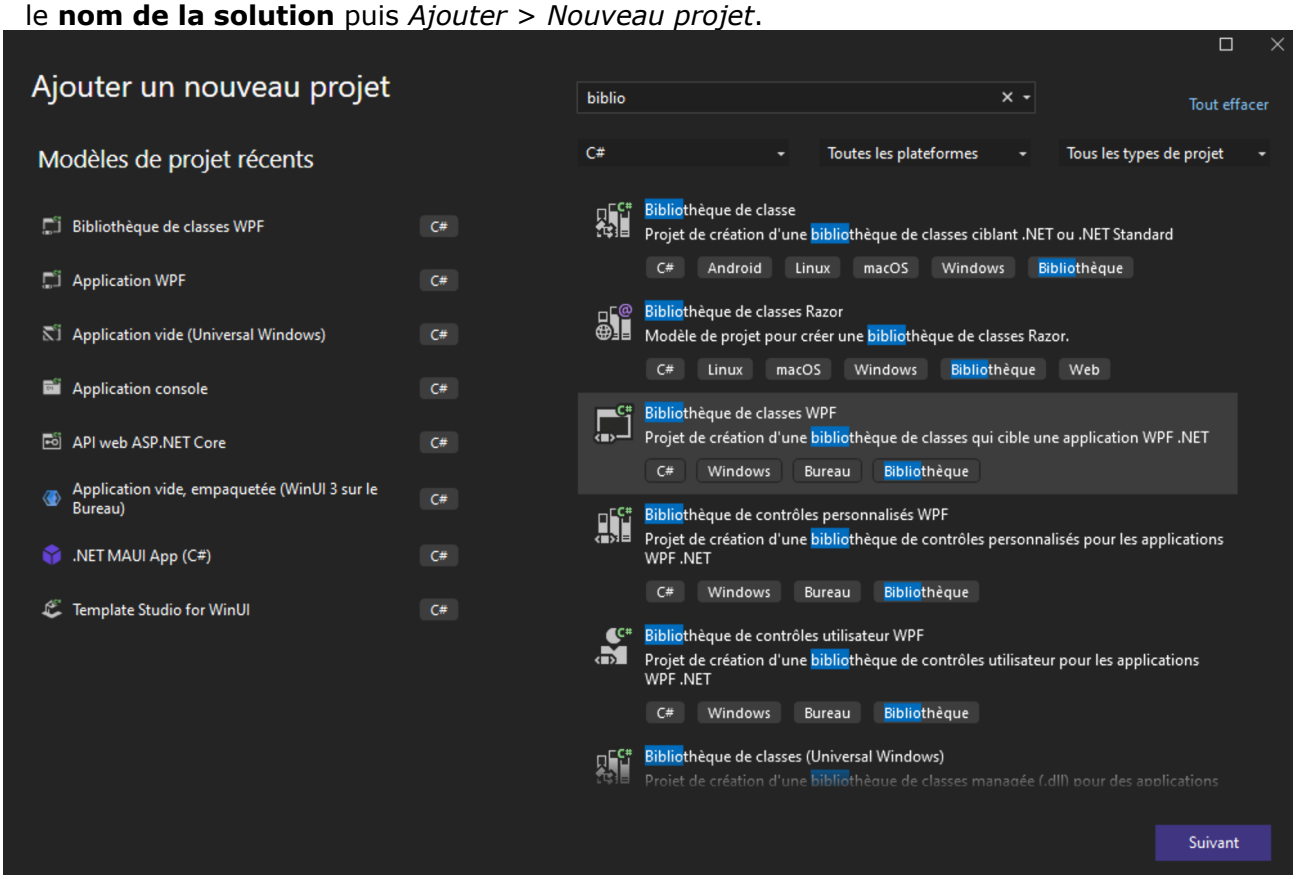
Tester votre application.
Faire valider votre code par votre enseignant.

2 Bonne pratique : architecture de l'application

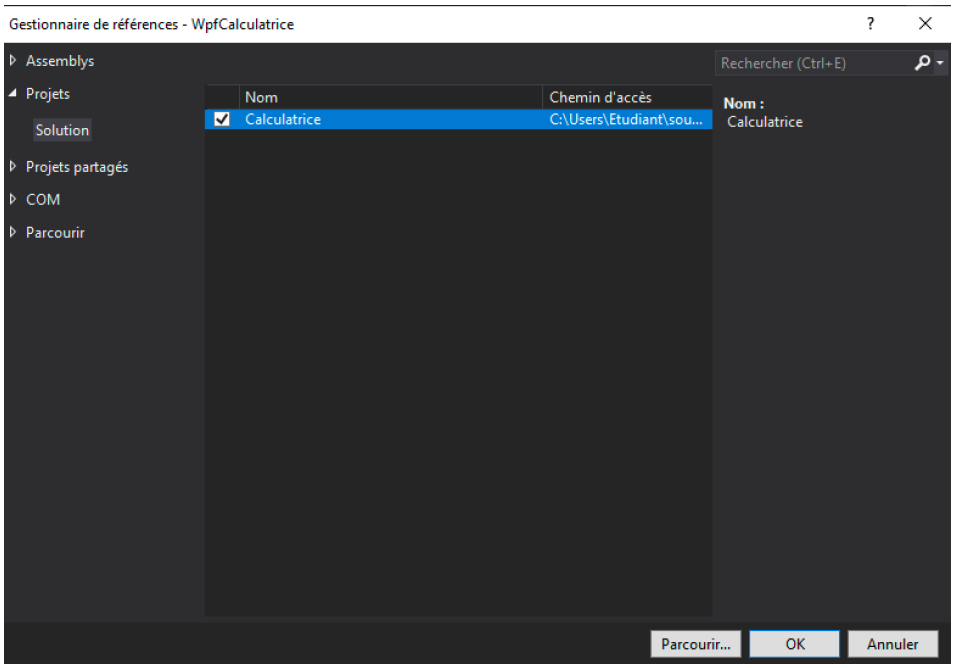
Une application WPF doit normalement appliquer le patron de conception MVVM. Cependant, l'application de celui-ci n'est pas triviale (nous le verrons ultèrieurement).
Nous allons nous limiter à décomposer le code et ainsi mettre le code de traitement (calcul) dans une bibliothèque de classes, afin de pouvoir le tester et surtout le réutiliser dans une autre application.
Actuellement, le code créé n'est pas directement réutilisable dans une application console, une application Windows Form ou une application Web ASP.Net.
En outre, on ne peut tester avec des tests unitaires une application de type Window (héritant de la classe `Window`). En effet, les tests unitaires ne permettent pas d'exécuter des méthodes événementielles (`click`, etc.). La seule façon de le faire étant de réaliser des tests end-to-end avec, par exemple, Selenium. *Nous y reviendrons…*
Remarque : vous pouvez modifier le code du projet précédent ou créer un nouveau projet V2.

Modifications à réaliser :

- Ajouter à la solution un nouveau projet « Bibliothèque de classes **WPF** » nommé `Calculatrice` intégrant une classe **static** `Calcul`.
  Pour ajouter un nouveau projet à la solution existante, cliquer avec le bouton droit de la souris sur le **nom de la solution** puis *Ajouter* > *Nouveau projet*.



- Coder **uniquement** la méthodes **static** `Addition` pour le moment.
- Dans le projet WPF, ajouter une référence vers la bibliothèque de classes (bouton droit de la souris sur *Dépendances* du projet `WpfCalculatrice` puis *Ajouter une référence de projet*)



- Modifier le code de l'application WPF pour appeler la méthode `Addition`.

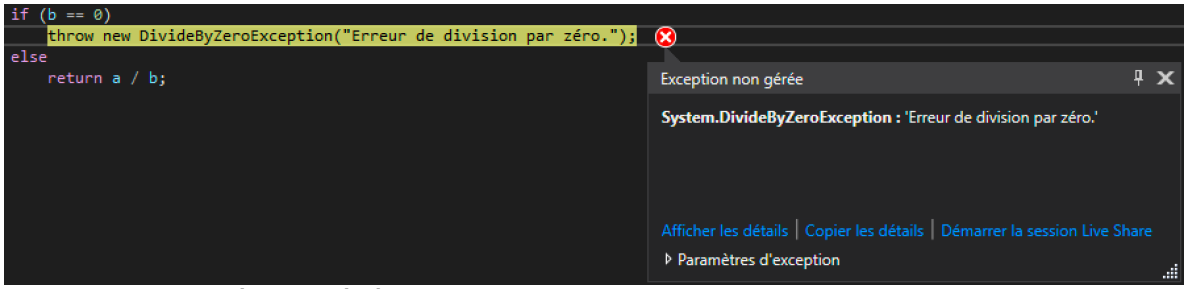
Exécuter votre application. Elle est maintenant mieux architecturée car la bibliothèque de classes est utilisable par une autre application (nous le vérifierons ultérieurement).
Faire de même pour la multiplication, la soustraction et la division. ATTENTION, quand on divise un nombre par 0, .Net renvoie l'infini (+*Infini*). Vous lèverez donc l'exception `DivideByZeroException("Erreur division par zéro")` en cas de tentative de division par zéro.

3 Ajout fonctionnel : factorielle

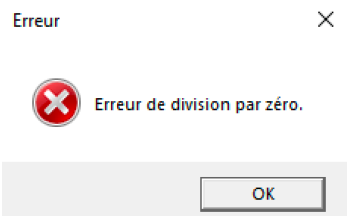
Ajouter la méthode dans la classe `Calcul` **sans utilisation de la récursivité** (boucle `for` par exemple) puis modifier l'application WPF. Le calcul de la factorielle sera réalisé sur le 1er nombre saisi.
Exemple de calculs : 0 (donne 1), 1 (donne 1), 10 (donne 3628800) et nombre < 0 (gérer une exception de type `ArgumentException`). Vous vérifierez si le nombre est bien un entier (sinon exception de type `ArgumentException`).

4 Bonne pratique : gestion des exceptions

Si vous testez la division par zéro avec un diviseur = 0, une erreur va être levée et votre application s'arrêter :



Dans l'application WPF, méthode événementielle `ButtonDivision_Click()`, rajouter un bloc `try catch` récupérant l'exception et affichant le message d'erreur renvoyée par l'exception (`MessageBox.Show()`) :

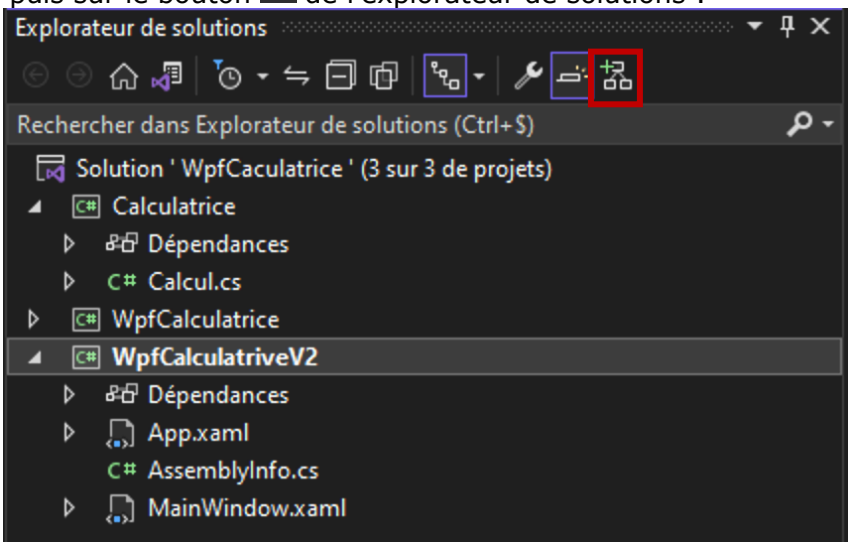


Remarque : pour générer un bloc `try catch`*, utiliser le snipet* `try` *(*`try + tab + tab`*).*

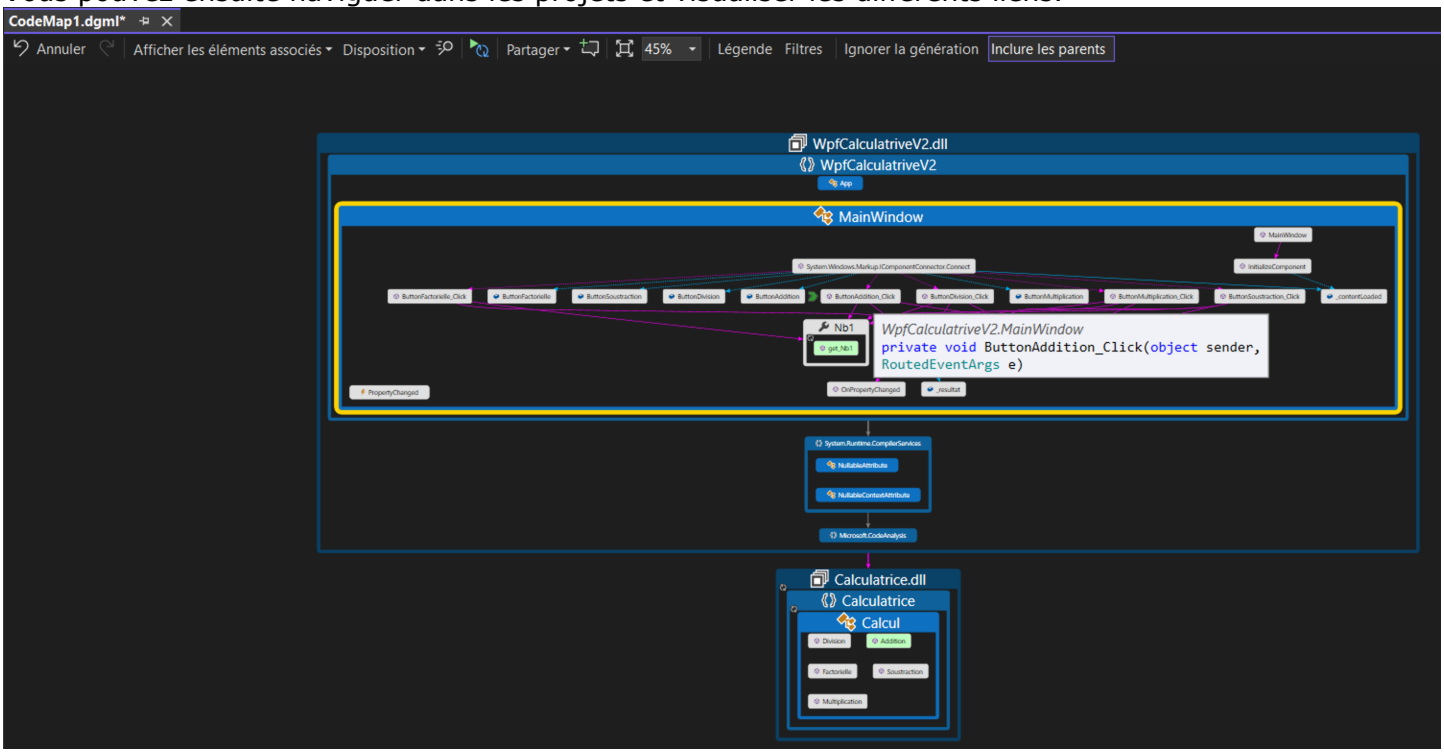
5 Bonne pratique : carte de code

Pour bien visualiser les liens entre les différents projets de votre solution, vous pouvez afficher une carte de code et naviguer dans celle-ci.

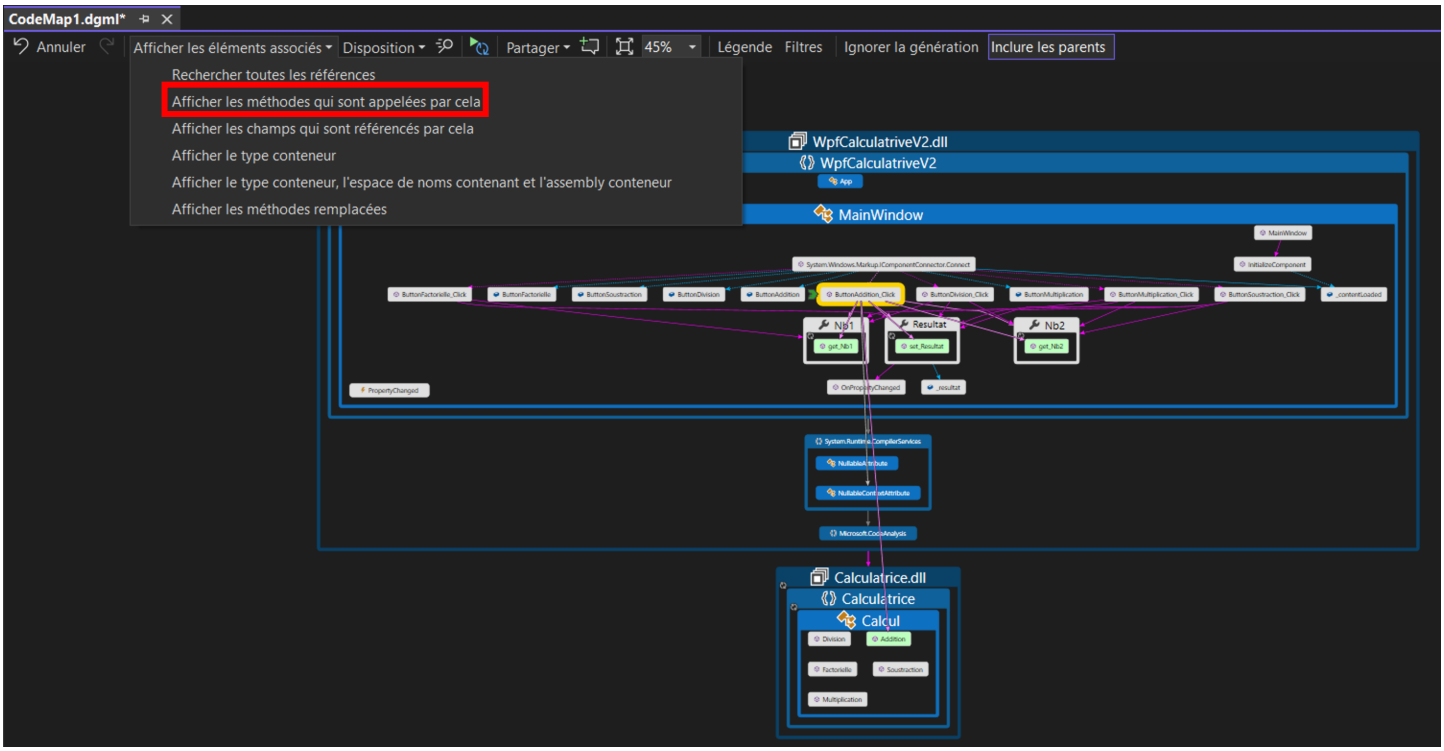
Cliquer sur la solution puis sur le bouton  de l'explorateur de solutions :



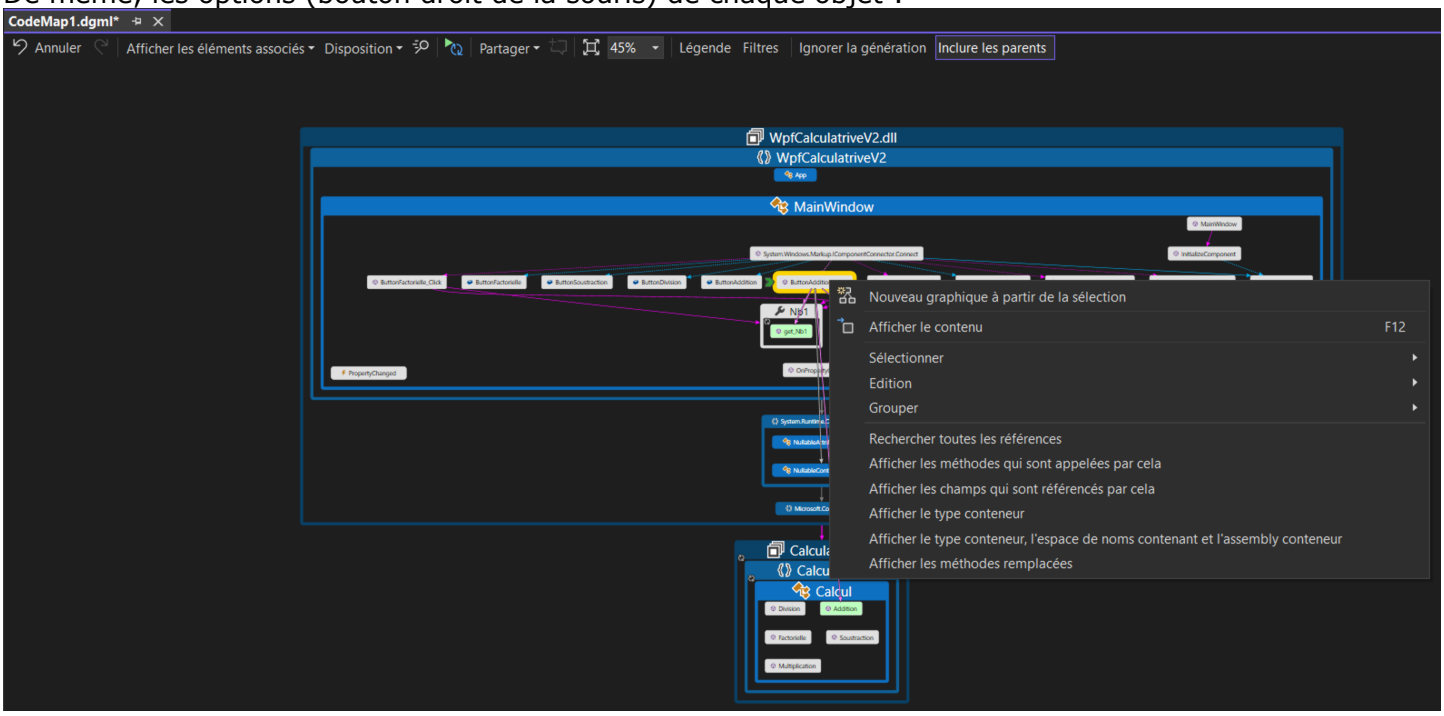
Vous pouvez ensuite naviguer dans les projets et visualiser les différents liens.



Essayer les différentes options :



De même, les options (bouton droit de la souris) de chaque objet :



6 Bonne pratique : documenter le code

Documenter votre code en utilisant les attributs XML suivants (XML DOC) :

- Classe :
  - `<summary></summary>` : Description de la classe.
  - `<remarks></remarks>` : Information supplémentaire, exemple.
- Méthode :
  - `<summary></summary>` : Description de la méthode.
  - `<remarks></remarks>` : Information supplémentaire, exemple.
  - `<returns><returns>` : Description de la valeur de retour
  - `<param name="xxxxx"></param>` : Description d'un paramètre d'entrée
  - `<exception cref="xxxxx"></exception>` : Description d'une exception

Exemple ici : https://docs.microsoft.com/fr-fr/dotnet/csharp/codedoc

Balises XML DOC prises en compte par Doxygen : http://www.doxygen.nl/manual/xmlcmds.html

En général, on documente :
- Le code des bibliothèques de classes car réutilisables.
- Le code des méthodes événementielles de l'application graphique (click, etc.).

On documente rarement les méthodes de test.

Lancer **Doxywizard** (installé sur le PC).

Remplir les parties de l'onglet Wizard :
- *Specify the working directory…* : indiquer un chemin valide
- *Projet* :
  - *Project name* : Remplir le nom du projet.
  - *Project version* : La version du projet.
  - *Source code directory* : Le répertoire contenant le code source à analyser et documenter.
  - *Scan recursively* : En cochant cette case, l'analyse du répertoire source se fera de façon récursive. Les sources contenues dans les sous-répertoires seront alors analysées.
  - Destination directory : Le répertoire dans lequel sera créée la documentation.
- *Mode* (essayer les différents modes) :
  - *Documented entities only* : Seules les entités (fonctions, méthodes, fichiers, structures...) documentées seront analysées et placées dans la documentation résultante.
  - *All Entities* : Toutes les entités présentent dans le code source seront analysées et documentées. Cette option ne peut pas générer du texte de documentation à votre place mais permet en revanche d'inclure toutes les entités dans les structures avec des liens entre elles pour une relecture plus aisée.
  - *Include cross-referenced* : Les fichiers sources seront insérés dans la documentation avec une coloration syntaxique appropriée et des liens vers les documentations précises.
  - *Select programing language to optimize the result for* : Ces options spéciales permettent de choisir la présentation de la documentation résultante la mieux adaptées au langage de votre source.
- *Output* : Cette partie spécifie les options de génération de la documentation, notamment le format de sortie.
  - *HTML* : générer une documentation au format HTML
  - *With navigation panel* (ou selon la version installée, *with frame and navigation tree*) : Ajoute une navigation facilitée par une présentation arborescente.
- *Diagrams* :

  Cette partie permet de configurer la création de diagramme dans la documentation.
  - *No diagrams* : Pas de création de diagramme.
  - *Built-in class diagram generator* : Créé la documentation par défaut.
  - *Use GraphViz* : Permet la création de diagrammes supplémentaires insérés dans la documentation. Les différents diagrammes proposés sont nombreux :
    - Représentation des relations directes et indirectes entre les classes.
    - Relations d'interdépendance des classes et structures.
    - Hiérarchie des classes.
    - Graphes de dépendances des fonctions.
    - Graphes des fonctions dont dépendent directement et indirectement les fonctions documentées.
    - Graphes d'appel direct et indirect des fonctions.
    - Graphes des fonctions appelées directement et indirectement par les fonctions documentées.

  Si *Use GraphViz* ne génère pas de diagramme, essayez avec *Use bult-in class diagram generator*.

Bilan : quand vous codez une application, veillez à appliquer les bonnes pratiques suivantes (non exhaustives) :
- **Définir une architecture permettant une réutilisabilité du code**
- **Gérer les exceptions**
- **Garder une vision claire de l'application : carte de code, diagrammes de classes, etc.**
- **Commenter le code et générer la documentation**