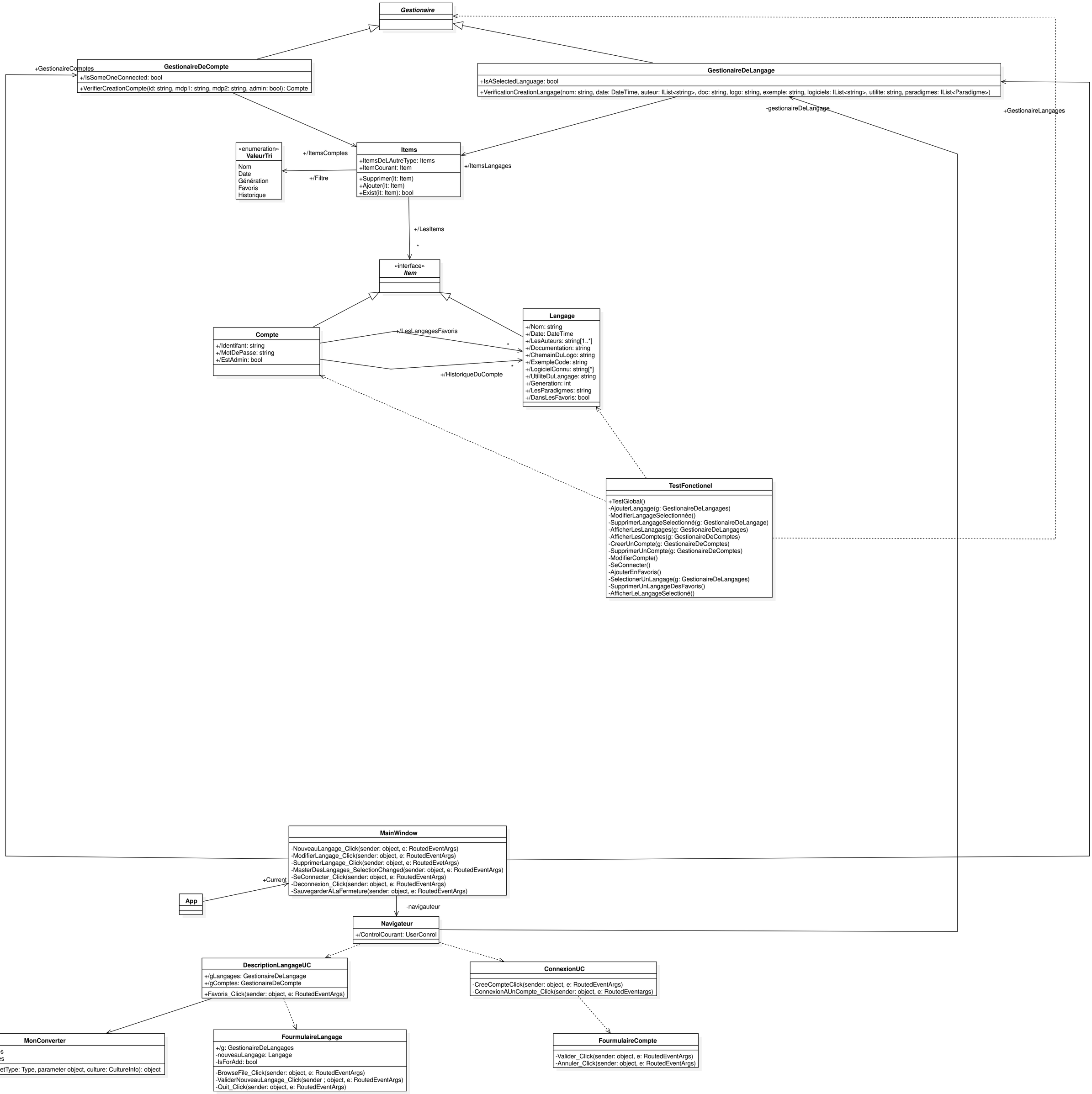
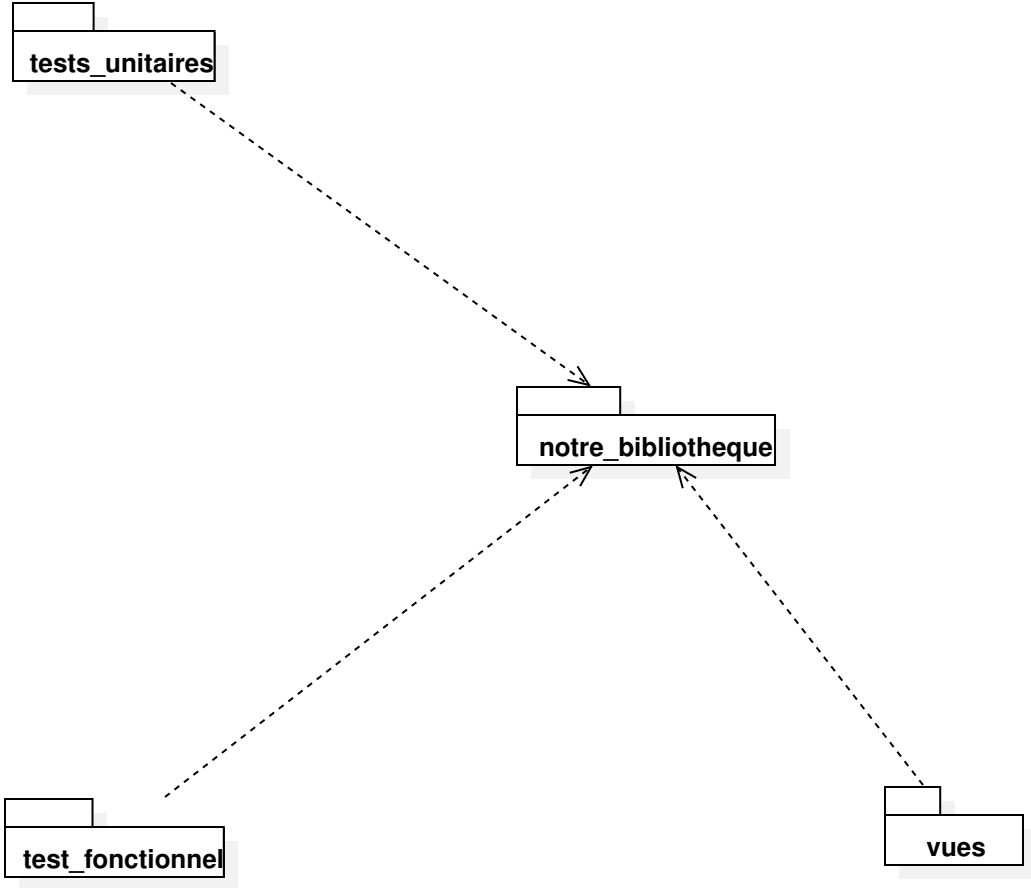
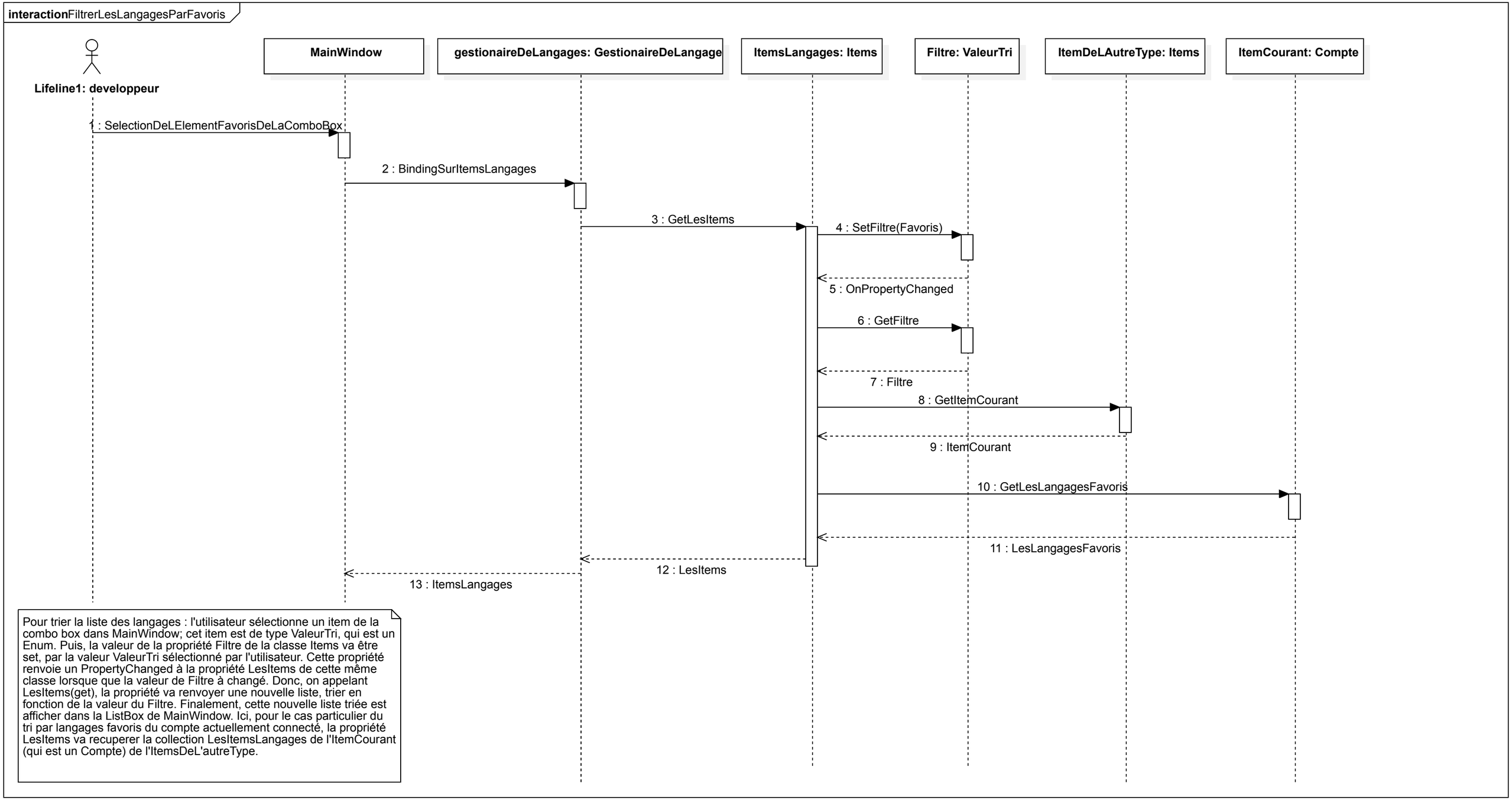

Gestionaire
GestionaireDeCompte
+/IsSomeOneConnected: bool
+VerifierCreationCompte(id: string, mdp1: string, mdp2: string, admin: bool): Compte
+GestionaireComptes
GestionaireDeLangage
+IsASelectedLanguage: bool
+VerificationCreationLangage(nom: string, date: DateTime, auteur: IList<string>, doc: string, logo: string, exemple: string, logiciels: IList<string>, utilite: string, paradigmes: IList<Paradigme>)
-gestionaireDeLangage
+GestionaireLangages
«enumeration»
ValeurTri
Nom
Date
Génération
Favoris
Historique
+/ItemsComptes
+/Filtre
Items
+ItemsDeLAutreType: Items
+ItemCourant: Item
+Supprimer(it: Item)
+Ajouter(it: Item)
+Exist(it: Item): bool
+/ItemsLangages
+/LesItems
*
«interface»
Item
Compte
+/Identifant: string
+/MotDePasse: string
+/EstAdmin: bool
+/LesLangagesFavoris
+/HistoriqueDuCompte
*
*
Langage
+/Nom: string
+/Date: DateTime
+/LesAuteurs: string[1..*]
+/Documentation: string
+/ChemainDuLogo: string
+/ExempleCode: string
+/LogicielConnu: string[*]
+/UtiliteDuLangage: string
+/Generation: int
+/LesParadigmes: string
+/DansLesFavoris: bool
TestFonctionel
+TestGlobal()
-AjouterLangage(g: GestionaireDeLangages)
-ModifierLangageSelectionnée()
-SupprimerLangageSelectionné(g: GestionaireDeLangage)
-AfficherLesLanagages(g: GestionaireDeLangages)
-AfficherLesComptes(g: GestionaireDeComptes)
-CreerUnCompte(g: GestionaireDeComptes)
-SupprimerUnCompte(g: GestionaireDeComptes)
-ModifierCompte()
-SeConnecter()
-AjouterEnFavoris()
-SelectionerUnLangage(g: GestionaireDeLangages)
-SupprimerUnLangageDesFavoris()
-AfficherLeLangageSelectioné()
MainWindow
-NouveauLangage_Click(sender: object, e: RoutedEventArgs)
-ModifierLangage_Click(sender: object, e: RoutedEventArgs)
-SupprimerLangage_Click(sender: object, e: RoutedEvetArgs)
-MasterDesLangages_SelectionChanged(sender: object, e: RoutedEventArgs)
-SeConnecter_Click(sender: object, e: RoutedEventArgs)
-Deconnexion_Click(sender: object, e: RoutedEventArgs)
-SauvegarderALaFermeture(sender: object, e: RoutedEventArgs)
+Current
App
-navigauteur
Navigateur
+/ControlCourant: UserConrol
DescriptionLangageUC
+/gLangages: GestionaireDeLangage
+/gComptes: GestionaireDeCompte
+Favoris_Click(sender: object, e: RoutedEventArgs)
ConnexionUC
-CreeCompteClick(sender: object, e: RoutedEventArgs)
-ConnexionAUnCompte_Click(sender: object, e: RoutedEventargs)
MonConverter
+/gc: GestionaireDeComptes
+/gl: GestionaireDeLangages
+Convert(value: object, targetType: Type, parameter object, culture: CultureInfo): object
FourmulaireLangage
+/g: GestionaireDeLangages
-nouveauLangage: Langage
-IsForAdd: bool
-BrowseFile_Click(sender: object, e: RoutedEventArgs)
-ValiderNouveauLangage_Click(sender ; object, e: RoutedEventArgs)
-Quit_Click(sender: object, e: RoutedEventArgs)
FourmulaireCompte
-Valider_Click(sender: object, e: RoutedEventArgs)
-Annuler_Click(sender: object, e: RoutedEventArgs)


tests_unitaires
notre_bibliotheque
test_fonctionnel
vues


interactionFiltrerLesLangagesParFavoris
Lifeline1: developpeur
MainWindow
gestionaireDeLangages: GestionaireDeLangage
ItemsLangages: Items
Filtre: ValeurTri
ItemDeLAutreType: Items
ItemCourant: Compte
1 : SelectionDeLElementFavorisDeLaComboBox
2 : BindingSurItemsLangages
3 : GetLesItems
4 : SetFiltre(Favoris)
5 : OnPropertyChanged
6 : GetFiltre
7 : Filtre
8 : GetItemCourant
9 : ItemCourant
10 : GetLesLangagesFavoris
11 : LesLangagesFavoris
12 : LesItems
13 : ItemsLangages
Pour trier la liste des langages : l'utilisateur sélectionne un item de la combo box dans MainWindow; cet item est de type ValeurTri, qui est un Enum. Puis, la valeur de la propriété Filtre de la classe Items va être set, par la valeur ValeurTri sélectionné par l'utilisateur. Cette propriété renvoie un PropertyChanged à la propriété LesItems de cette même classe lorsque que la valeur de Filtre à changé. Donc, on appelant LesItems(get), la propriété va renvoyer une nouvelle liste, trier en fonction de la valeur du Filtre. Finalement, cette nouvelle liste triée est afficher dans la ListBox de MainWindow. Ici, pour le cas particulier du tri par langages favoris du compte actuellement connecté, la propriété LesItems va recuperer la collection LesItemsLangages de l'ItemCourant (qui est un Compte) de l'ItemsDeL'autreType.

POINT Allan – LABUSSIERE Lou

Descriptions

Table des matières

Paquets 2
Notre_Bibliothèque 2
Vues 2
Test_Fonctionnel 2
Test_Unitaire 2
Données 2
PersDataContract 2
Architecture 2
Classes 3

Paquets

Notre_Bibliothèque

Le package *Notre_bibliothèque* représente le modèle de notre application. C'est donc le centre de l'application. Ce package possède entre autres les classe Langage, Compte ou Gestionnaire.

Vues

Le package *vues* contient les classes qui permettent d'avoir une interface graphique. Il est donc dépendant de *notre_Bibliothèque* car il permet d'afficher graphiquement des objets du modèle. Ces objets sont stockés dans les packages *Données* et *PersDataContract*. Pour la vue, il à été choisi de représenter les objets de *PersDataContract*.

Test_Fonctionnel

Le package *test_fonctionnel* est utilisé pour pouvoir tester les fonctionnalités de l'application. Elle test donc la capacitée du model à encaisser les fonctionnalités prévues. C'est pour ça que ce package dépend de *notre_bibliothèque*. Les données utilisées sont stockées par le Stub. C'est pour ça que *test_fonctionnel* dépend aussi de *données*.

Test_Unitaire

Le package *test_unitaire* est utiliser pour tester le comportement des classes de *notre_bibliothèque*. Elle utilise les objets stockés dans le Stub. C'est pour cela que *test_unitaire* dépend de *notre_bibilothèque* et de *données*.

Données

Le package *données* contient la classe Stub, qui permet un chargement de données. En effet, ce Stub charge des objets de *notres_bibliothèque*. C'est pour ça que le package *données* dépend de *notre_bibliothèque*.

PersDataContract

Ce package permet de charger des données de *notre_bibliothèque* depuis un fichier xml. C'est pour ça que *PersDataContract* dépend de *notre_bibliothèque*.

Architecture

Le modèle est le cœur de l'application. C'est pour ça que la plupart des classes qui sont hors du modèle y font référence. Le modèle est composé de plusieurs classes, énumération et interface. Le centre du modèle sont les classes dérivées de l'interface *Item* (*Compte* et *Langage*). Les objets de chacune des classes dérivées d'Item sont contenus dans la classe *Items* qui sert de point de sérialisation. En suites les classes dérivées de la classe abstraire *Gestionnaire* permette la

communication entre la vue et le modèle. Enfin, l'interface *IPersistance* permet de faire persister les données utilisées par le modèle.

La vue est composée d'une fenêtre principale (*MainWindow*) qui contient le Master. Pour le Detail, il est contenu dans le contrôle utilisateur *DescriptionDesLangagesUC*. Ainsi le Detail peut être remplacer par le contrôle utilisateur *ConnexionUC*. La navigation entre ces deux contrôle utilisateurs est gérer par un *Navigateur*. La vue communique avec le modèle depuis la fenêtre principal grâce aux gestionnaires.

Les tests sont rangés dans différents packages selon leurs types. Ainsi, les tests unitaires ne viennent pas influencer les tests fonctionnels et vice versa.

Classes

Les deux classes principales de notre projet sont les classes **Compte** et **Langage**. En effet, notre application est centrée sur les langages informatiques, et certaines actions sur l'application ne sont possibles que si l'utilisateur est connecté à un compte.

La classe **Compte** a cinq propriétés : Identifiant et MotDePasse de type string, EstAdmin de type bool (qui indique si le **Compte** à le droit d'accès au mode administrateur), LesLangagesFavoris et HistoriqueDuCompte qui sont des collections IList de Langage.
Un **Compte** est indentifiable grâce à son Identifiant et son MotDePasse (les méthodes Equals utilisent ces propriétés).

La classe **Langage** elle a dix propriétés : Nom, Documentation, CheminDuLogo, ExempleCode, UtiliteDuLangage qui sont de type string, Date de type DateTime, LesAuteurs et LogicielConnu qui sont des collections IList de string, Generation de type int, LesParadigmes qui est une collection de string, et enfin DansLesFavoris de type bool (qui indique si le **Langage** sélectionné est actuellement dans la collection LesLangagesFavoris du **Compte** connecté ; cette propriété n'est qu'utile pour le binding avec le bouton favoris dans la vue).
Un **Langage** est indentifiable grâce à son Nom (les méthodes Equals utilisent cette propriété pour les comparer)

Les deux classes **Compte** et **Langage** implémente l'interface ***Item***.

La classe **Items** sert à regrouper tous les ***Item*** (**Compte** et **Langage**) dans une collection. Cela simplifie le codage, et permet de gérer les Item plus facilement. Cette classe contient plusieurs propriétés :

ItemCourant de type ***Item***, qui indique l'***Item*** actuellement en cours d'utilisation : si l'***Item*** est un **Compte**, ItemCourant sera le **Compte** connecté, et si l'***Item*** est un **Langage**, il sera le **Langage** sélectionné.

LesItems est une collection IList d'***Item***, cette propriété regroupe tous les ***Item*** (**Compte** ou **Langage**).

ItemsDeLAutreType est une propriété de type Items, qui correspond au type de l'autre Items que celui en cours d'utilisation : dans la classe Items, si l'ItemCourant est un Item de type Langage, l'Items retournés par la propriété ItemsDeLAutreType sera un Compte, et vice-versa. Cette propriété est utile lorsqu'il s'agit de filtrer la liste LesItems : elle permet de récupérer la liste LesLangagesFavoris ainsi que l'HistoriqueDuCompte du Compte actuellement connecté (ce Compte sera l'ItemCourant de ItemsDeLAutreType).

Filtre est une propriété de type **ValeurTri**, qui est un enum. Cette enum contient les valeurs Nom, Date, Génération, Favoris et Historique. La collection LesItems sera triée en fonction du Filtre.
Cette classe contient également plusieurs méthodes :

Supprimer permet la suppression d'un ***Item*** donné de la collection LesItems.

Ajouter permet l'ajout d'un ***Item*** donné à la collection LesItems.

Exists permet de savoir si un ***Item*** donné est présent dans la collection LesItems, en rendant un bool.

Les classes **GestionaireDeCompte** et **GestionaireDeLangage**, qui sont des classe filles de la classe abstraite ***Gestionnaire***, vont permettre la gestion des **Items**.
Ces deux classes sont assez similaires. En effet elles possèdent toutes les deux une propriété de type bool qui indique si un ***Item*** est sélectionné (IsSomeoneConnected pour GestionaireDeCompte, et IsASelectedLanguage pour GestionaireDeLangage), ainsi qu'une propriété de type **Items** qui permet de récupérer la collection LesItems ainsi que l'ItemCourant correspondants (ItemsComptes pour GestionaireDeCompte, et ItemsLangages pour GestionaireDeLangage).
Elles possèdent également des méthodes qui vérifient la création d'un ***Item***.

VerificationCreationCompte vérifie d'abord si l'identifiant saisi est attribué ou non à un **Compte** de la collection, puis vérifie si les deux mots de passes saisis sont bien identiques. S'il n'y a pas d'erreurs, la méthode renvoie une instance du nouveau **Compte** avec l'identifiant et mot de passe saisis.

VerificationCreationLangage vérifie si le nom saisi est attribué ou non à un **Langage** de la collection, et que toutes informations saisis soient bien conformes. S'il n'y a pas d'erreurs, la méthode renvoie une instance du nouveau **Langage** avec toutes informations saisies.

Toutes ces classes font parties du notre modèle notre_bibliotheque. Ce modèle va nous servir à faire fonctionner notre application. Concernant les vues de notre application, c'est-à-dire l'aspect graphique, nous avons un paquet vues qui contient tous les fichiers .xaml (les fenetres et les user control) ainsi que leurs code behind .xaml.cs.

MainWindow est la fenêtre principale de l'application. C'est elle qui sera directement lié au modèle, elle implémente un **GestionaireDeLangage** ainsi qu'un **GestionaireDeCompte**. De plus, elle possède un attribut de type **Navigateur**. Cette classe **Navigateur** va permettre de changer les user control du content control sur **MainWindow** en fonction des actions de l'utilisateur (c'est pour cette raison que les User Control dépendent de cette classe).
Les ***Gestionnaire*** seront accessibles depuis **MainWindow** sur toutes les autres fenêtres ou User Control, il n'y a pas besoin d'en instancier d'autres à chaque fois.
Nous avons deux User Control; **DescriptionLangageUC** et **ConnexionUC**. Par défaut, si l'utilisateur n'a pas sélectionné de Langage, c'est **ConnexionUC** qui est affiché dans le content control de **MainWindow** (ce User Control est également accessible depuis le bouton connexion du menu).
Sinon, dès qu'un **Langage** est sélectionné depuis la ListeBox, le content control change. Par ailleurs, quand un **Langage** est sélectionné, et si l'utilisateur est connecté et a accès au mode administrateur, il aura la possibilité de modifier ou supprimer ce **langage** sélectionné. Le mode administrateur permet également d'ajouter un **Langage**. Modifier ou ajouter un **Langage** donne accès à une nouvelle fenêtre **FormulaireLangage**. Si l'utilisateur veut modifier un **Langage** les champs seront prérempli avec les informations actuelles du **Langage** sélectionné (c'est pour cette raison que cette fenêtre dépend du user control **DescriptionLangageUC**, pour récupérer les informations). Pour l'ajout, les champs restent vides.
Le User Control **DescriptionLangagUC** permet de consulter les informations d'un langage, et il est aussi possible d'ajouter ce même langage dans la liste des favoris (si l'utilisateur est connecté).
Une classe fille de l'interface IValueConverter nommé **MonConverteur** à été ajouté. En effet, cette classe va servir à convertir un type Booléen en BitmapImage. Dans la partie xaml du user control **DescriptionLangageUC**, le bouton favoris contient une image, et la source de cette image est en binding avec la propriété DansLesFavoris de la classe Langages. Le converteur va alors permettre de convertir un type bool en BitmapImage (l'image est une étoile jaune si la propriété rend vrai, une étoile noire sinon).

Le User Control **ConnexionUC**, lui, permettra à l'utilisateur de se connecter à son compte, mais aussi de créer un compte. La création d'un compte donne accès à la page **FormulaireCompte** (cette page n'est qu'accessible que depuis le User Control, c'est pour cela qu'ils sont en dépendance). Et lorsqu'un **Compte** est créé, l'utilisateur est immédiatement connecté à ce **Compte** sur l'application.