REF : SENT-TECH-00-00

BDO Sentinel

Document technique

Conventions et

Bonnes pratiques

****

Version - 1.0

Table des matières

[A. Convention de nommage 3](#_Toc467089102)

[A.1. SQL 3](#_Toc467089103)

[A.2. C# 3](#_Toc467089104)

[A.3. MVVM 4](#_Toc467089105)

[B. Règles et bonnes pratiques 5](#_Toc467089106)

[B.1. SQL 5](#_Toc467089107)

[B.2. C# 5](#_Toc467089108)

Gestion des versions

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro | Emetteur | Description | Date |
| V1.0 | MBU | Rédaction première version du document | 14/11/2016 |
|  |  |  |  |

Création du document par Michaël Buisson, le 14/11/2016 08:00

Dernière sauvegarde effectuée par DGBUISSM, le 16/02/2017 10:07

Ce document reprend l’ensemble des conventions de développement qui seront utilisées dans la réalisation de l’application Sentinel.

Par conventions de développement, il est entendu l’ensemble des règles et bonne pratiques à suivre tout au long de l’implémentation du projet. En utilisant une ligne directrice commune à tous les développeurs, cela doit permettre une meilleure compréhension par une tierce personne et également mettre en place un code propre, réutilisable et plus facile à mettre à jour.

Ce document est amené à être complété au fur et à mesure des formations et développements.

# Convention de nommage

Tous les noms des éléments (variables, méthodes, SP…) ainsi que les commentaires seront en anglais.

## SQL

* Ne pas utiliser les ‘\_’ dans les noms de tables/vues.
* Index : IX\_NomDeLaTable\_NomDesColonnes
* Default : DF\_NomDeLaTable\_NomDeLaColonne
* Primary Key : PK\_NomDeLaTable
* Foreign Key : FK\_NomDeLaTableNomDeLaColonne\_NomDeLaTableDestination
* Le nom des tables et des champs doivent suivre la convention « CamelCase », avoir un nom clair et toujours au singulier. Il ne doit pas contenir d’abréviation sauf si l’abréviation est claire(Qty=Quantity,etc…).

*Ex: UserAddress, AddressID, StreetNumber*

* Précéder les fonctions SQL par « ufn » (*user function*).
* Précéder les procédures stockées SQL par « usp » (*user stored procedure*)
* Les champs pouvant contenir des caractères spéciaux « Unicode » doivent être du type NVarchar, Nchar (N représentant l’Unicode).

## C#

* Le nom des classes suivra la convention « CamelCase ». Une exception sera faite pour les interfaces et classes abstraites auxquelles on ajoutera respectivement un « I » et « A » en première position. *Ex : ClassName, IClassName, AClassName.*
* Les propriétés et méthodes respecteront la convention « CamelCase ».
* Le nom des méthodes de type fonction devront commencer par un verbe qui énoncera clairement leur rôle. *Ex : IsActive, DoesUserExists, GetPhoneNumber*.
* Les variables et paramètres des méthodes respecteront la convention « pascalCase » et ne contiendront pas de ‘\_’. *Ex : userCity*.
* Une exception pour les variables de type « private » qui commenceront par ‘\_’. *Ex :* *\_userPhoneNumber*.
* Il n’y aura pas de variables publiques, on utilisera des propriétés.
* Le nom des variables et énumérations ne commenceront pas par leur type (ex string strStreetName, Enum enumCodeXxx.)
* Le typage des variables sera fait via le mot clé en minuscule lorsque c’est possible (Ex : string, int).
* Le nom des listes sont toujours au pluriel (Ex : List<User> users -> liste des utilisateurs)
* Les abréviations dans les noms de classes, variables etc… sont autorisées lorsqu’elles sont claires (Qty=Quantity…).

## Préfixes des contrôles statiques

|  |  |  |
| --- | --- | --- |
| Type de champ | Préfixe | Exemple |
| Button | btn |  |
| Block | Blk |  |
| TextBox | txt |  |
| ComboBox | cbx |  |
| Checkbox | chk |  |
| Calendar | cal |  |
| Label | lbl |  |
| Dropdownlist | ddl |  |
| Radiobutton | rbt |  |
| Checkboxlist | cbl |  |
| Groupbox | grp |  |
| Panel | pnl |  |
| Gridview | grv |  |
| Grid | grd |  |
| Statusbar | stb |  |
| Menu | mnu |  |
| MenuItem | mni |  |
| Image | img |  |
| Treeview | tvw |  |
| PasswordBox | pwb |  |

## MVVM

* Dans la structure des classes MVVM il y aura toujours ClassNameView, ClassNameViewModel, ClassNameModel

# Règles et bonnes pratiques

Les documents de ce cahier des charges décrivent le besoin fonctionnel avec une vision utilisateur (description des fenêtres, menus, écrans …). Néanmoins, il convient d’avoir une première vision plus globale des grands axes de développement à envisager.

L’application sera multilingue. Tous les termes visibles par les utilisateurs devront pouvoir être traduits par l’administrateur fonctionnel.

L’application sera du type client-serveur, avec SQL Server comme base de données. Il est important de s’assurer d’un déploiement aisé de l’application sur les postes clients. De même, dans le but d’une commercialisation de la solution, l’installation initiale de l’application ne devra pas dépasser une ½ journée par nouveau client.

Les performances (temps d’affichage, de réaction, d’exécution de l’application) devront être au moins identiques aux performances du logiciel KYC actuellement utilisé par BDO Luxembourg

La solution à développer peut être découpée suivant 8 grands axes (piliers).

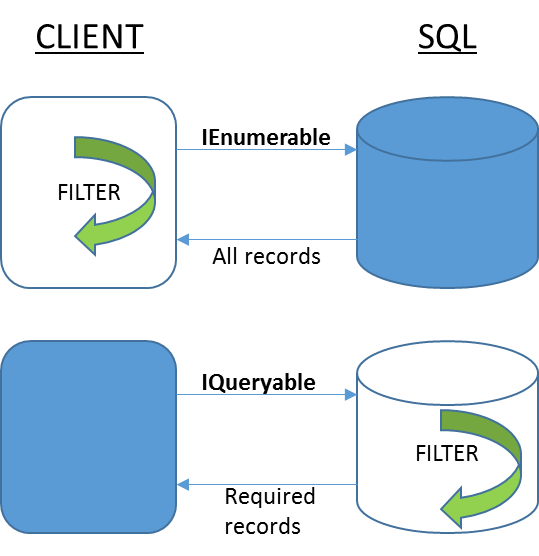
## SQL

* Eviter autant que possible d’implémenter les règles et calculs métiers dans les procédures stockées.
* Toujours s’assurer que les champs impliqués dans des jointures ou comparaisons sont du même type. Cela évite à SQL de procéder à des conversions (*ex : BigInt-BigInt, Varchar()-Varchar, NVarchar()-NVarchar()… PAS DE Varchar()-NVarchar()*)
* La clé primaire doit être une clé de substitution. Elle est définie sur un champ de type BigInt Identity.
* Basé sur la règle précédente, chaque table doit donc contenir un index de type Unique pour éviter les doublons.
* Chaque table devra posséder des champs d’audit qui seront placés à la fin de la liste des champs (CreationDate, CreationUser, LastUpdateDate, LastUpdateUser).
* Toujours nommé explicitement les index, clés, valeurs par défaut etc…
* Dans les procédures stockées d’Insert, pour les tables avec clé de substitution de Identity, il faut ajouter la commande suivante après l’insertion (permet de retourner l’id affecté par SQL vers EF) :

Select SCOPE\_IDENTITY() as *ColName*

## C#

* Pour l’implémentation des propriétés, on utilisera un membre privé uniquement s’il y a besoin de code dans les get/set. Par code, on entend uniquement des évènements à déclencher. *Ex : INotifyPropertyChanged*.
* LINQ : utiliser .Any() au lieu de .Count()>0
* LINQ : les listes LINQ to SQL doivent être de type **IQueryable**. En effet, les conditions sont alors appliquées lors de la requête du côté SQL tandis qu’avec le type IEnumerable, tous les records sont récupérés de la base de données et filtrés en mémoire ce qui peut engendrer des problèmes de performance.



* Design pattern Repository : toujours créer un constructeur avec le contexte en paramètre. Cela permet de gérer les UnitOfWork.