REF : SENT-ARCHI

BDO Sentinel

Cahier des charges (CDC)

Architecture

****

Description d’une architecture sur base du CDC-99-00

Version – 2.0

Table des matières

[A. Lexique 5](#_Toc469296675)

[B. Définition des packages NuGet 6](#_Toc469296676)

[C. Le Pattern MVVM (Modèle – Vue – Vue Modèle) 6](#_Toc469296677)

[C.1. Vue (V) 6](#_Toc469296678)

[C.2. Vue Modèle (VM) 6](#_Toc469296679)

[C.3. Modèle (M) 7](#_Toc469296680)

[A. Schéma de l’architecture 8](#_Toc469296681)

[A.1. Vue générale 8](#_Toc469296682)

[A.2. Shell (Interface utilisateur) 9](#_Toc469296683)

[A.3. Les modules 11](#_Toc469296684)

[A.3.a. Module Navigation 11](#_Toc469296685)

[A.3.b. Module BusinessScreens 11](#_Toc469296686)

[A.3.c. Module Query 12](#_Toc469296687)

[A.3.d. Module Security 12](#_Toc469296688)

[A.3.e. Module Scenario 12](#_Toc469296689)

[A.3.f. Module CrossControl 12](#_Toc469296690)

[A.3.g. Module Dynamic 12](#_Toc469296691)

[A.3.h. Module Configurations 12](#_Toc469296692)

[A.4. Les Services 12](#_Toc469296693)

[A.4.a. LoggerService 13](#_Toc469296694)

[A.4.b. AuthenticationService 13](#_Toc469296695)

[A.4.c. ResourcesService 14](#_Toc469296696)

[A.4.d. GlyphService 14](#_Toc469296697)

[A.4.e. AuditService 14](#_Toc469296698)

[A.4.f. ScenarioService 14](#_Toc469296699)

[A.5. Les projets Infrastructure 14](#_Toc469296700)

[A.5.a. Projet DataAccess 15](#_Toc469296701)

[1. Projet Service.DataAccess.Interfaces 15](#_Toc469296702)

[2. Projet Sentinel.DataAccess 15](#_Toc469296703)

[A.5.b. Projet Common 16](#_Toc469296704)

[A.6. Connexion 17](#_Toc469296705)

[A.7. Exécution d’un événement (Scenario) 18](#_Toc469296706)

[A.8. Prism 19](#_Toc469296707)

[B. Environnement Infrastructure 20](#_Toc469296708)

[B.1. Outils de développement 20](#_Toc469296709)

[B.1.a. Visual Studio 2015 20](#_Toc469296710)

[B.1.b. SQL Server 2012 21](#_Toc469296711)

[B.2. Configuration de déploiement 22](#_Toc469296712)

[B.2.a. Déploiement sur un seul poste 22](#_Toc469296713)

[B.2.b. Déploiement sur plusieurs postes utilisateurs 23](#_Toc469296714)

[C. Documentation technique 24](#_Toc469296715)

[C.1. Organisation d’un appel DB 24](#_Toc469296716)

[C.2. Organisation des événements(scenarii) 24](#_Toc469296717)

[C.2.a. Chargement des scenarii 25](#_Toc469296718)

Gestion des versions

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro | Emetteur | Description | Date |
| V1.0 | XMA | Rédaction première version du document | 16/11/2016 |
|  | XMA | Mise à jour du document | 17/11/2016 |
|  | XMA | Mise à jour du document | 1/12/2016 |
|  | XMA | Mise à jour du document | 2/12/2016 |
|  | XMA | Mise à jour du document | 5/12/2016 |
|  | XMA | Mise à jour du document | 6/12/2016 |
|  | XMA | Mise à jour du document | 8/12/2016 |
|  | XMA | Mise à jour du document | 9/12/2016 |
|  | XMA | Mise à jour du document | 12/12/2016 |

Création du document par Xavier Mahieu, le 16/11/2016 07:14

Dernière sauvegarde effectuée par Xavier Mahieu, le 07/11/2016 16:38

Ce document décrit l’architecture mise en place pour l’application « Sentinel » ainsi que les différents principes et technologies utilisés.

Cette proposition d’architecture se base sur les cahiers des charges en notre possession à la date du 11/11/2016 (CDC-99-00).

Ce document évoluera au fur et à mesure des des avancements des développements de l’application, basé sur le CDC-99-00.

# Lexique

|  |  |
| --- | --- |
| Termes | Définitions |
| MVVM | Abréviation pour la désignation du pattern “ Modéle – Vue – Vue Modèle “ |
| WPF | Abréviation de “Windows Presentation Foundation“ |
| ORM | Abréviation de “Object-Relational Mapping” |
| EF | Abréviation de “Entity Framework” |
| LinQ | Abréviation de Language Integrated Query |
| IoC | Abréviation de " Inversion of Control" |

# Définition des packages NuGet

Les packages NuGet utilisés par l’application seront stockés sur un répertoire partagé en réseau.

Ceci permettra la centralisation des packages NuGet utilisés par les développeurs et les forcera à utiliser les mêmes versions de package.

Cela permet d’éviter d’avoir des soucis d’incompatibilités entre des versions différents d’un même package

Les packages NuGet utilisés sont les suivants :

|  |  |
| --- | --- |
| Noms | Version |
| EntityFramework | 6.1.3 |
| EntityFramework.CodeTemplates.CSharp | 6.1.3 |
| Prism |  |

Le document … décrit la façon d’installer les packages NuGet dans Visual Studio.

# Le Pattern MVVM (Modèle – Vue – Vue Modèle)

Le pattern « MVVM » (Modèle – vue – vue modèle) est une architecture et une méthode de conception utilisée dans le génie logiciel. Il est particulièrement adapté au développement des applications utilisant WPF.

Ce pattern permet de séparer l’interface utilisateur de la logique et de l’accès aux données en accentuant les principes de « binding » et « d’événement ».



## Vue (V)

La couche « Vue » développée en XAML, affiche les données vues par l’utilisateur. Il n’y a pas de logique business dans cette couche. Dès qu’un utilisateur va générer un évènement (bouton, modification de données, etc.), elle va envoyer l’information à la couche « Vue Modèle » qui prendra le relais pour le traitement à effectuer selon l’action. Ce qui signifie qu’il n’y a aucune implémentation dans la vue contrairement aux développements Windows classique (sauf cas spécifique qui ne pourrait être directement traité en XAML comme la logique UI).

## Vue Modèle (VM)

La couche « Vue Modèle » s’occupe de traiter les interactions entre la couche « Vue » et la couche « Modèle ». Elle aura pour rôle de s’assurer du rafraichissement de données présentées dans la vue en cas de modification du modèle. Pour cela la couche « Vue Modèle » notifiera la vue des changements à l’aide d’évènements de type « INotifyPropertyChanged ». Elle aura également pour rôle, à l’aide des bindings, de mettre à jour le modèle en fonction des données renseignées par l’utilisateur dans la vue.

## Modèle (M)

La couche « Modèle » va s’occuper du stockage et de la gestion des données, des accès aux bases de données et de l’implémentation de la logique métier à appliquer. Elle ne communique qu’avec la couche « Vue Modèle ».

# Schéma de l’architecture

## Vue générale



* **Shell** : Interface utilisateur utilisant des composants DevExpress et le framework Prism.
* **Modules** : Les modules sont le regroupement fonctionnel des vues et elles seront réutilisables.
* **Services** : Les services sont les entités qui exécutent la logique fonctionnelle de l’application.
* **Infrastructure** : Regroupe les Helpers et les classes abstraites (RegionManager, etc.).
* **Base** **de données** : Il s’agit d’une base de données hébergée en SQL Server 2012

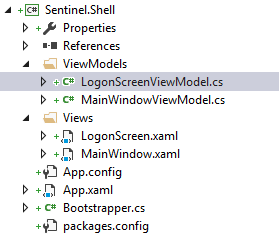
L’utilisation du framework « Prism » permet de faciliter l’affichage et la réutilisation des vues, définies dans les modules. Ces peuvent être réutilisées dans plusieurs écrans différents.

Il permet également de faciliter la gestion de la navigation définie dans le cahier des charges et qui est un point essentiel pour BDO afin de rendre l’application plus convivial et intuitive.

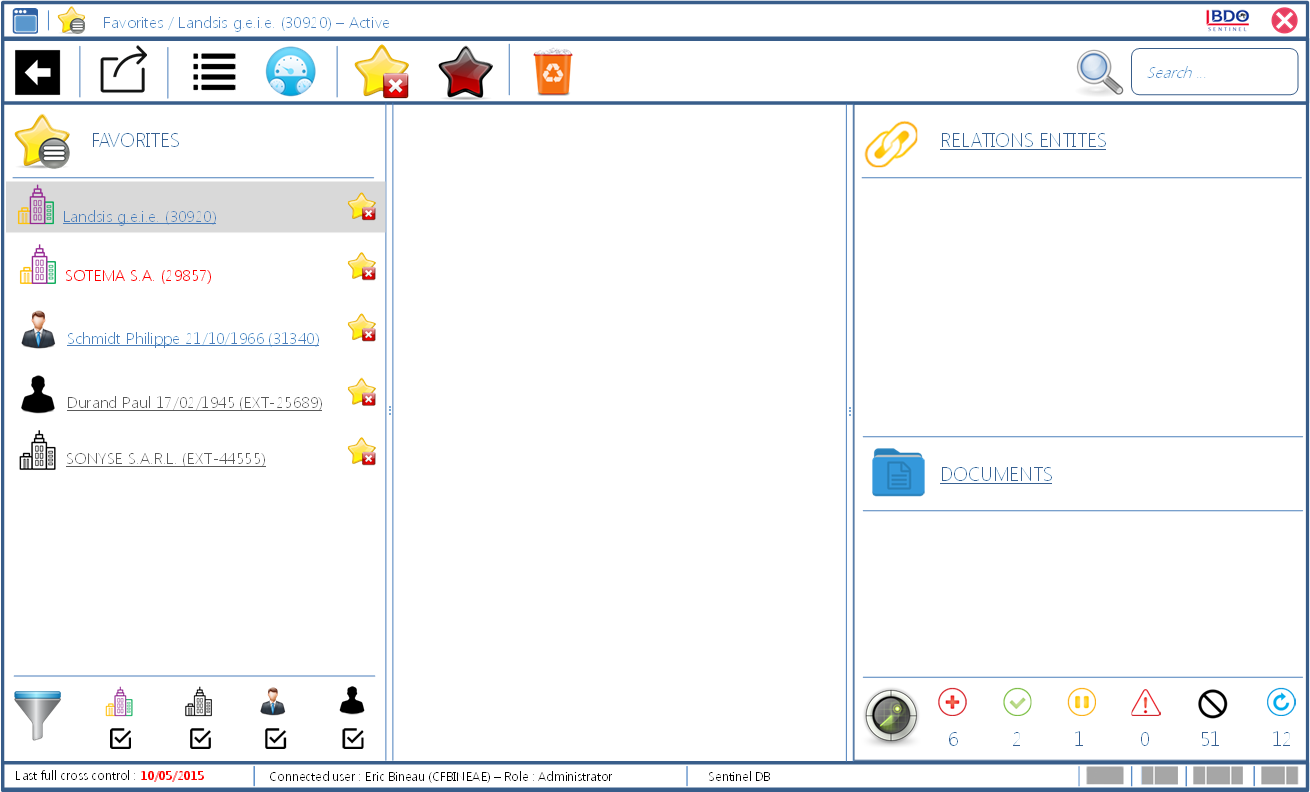
Le Shell et les modules respectent l’utilisation du pattern « MVVM » afin de séparer l’interface utilisateur de l’implémentation.

## Shell (Interface utilisateur)

Il s’agit de l’interface graphique de l’application qui est subdivisé en plusieurs régions. Le projet est « Sentinel.Shell » dans la solution « Sentinel ».



La fenêtre principale « MainWindow » est divisée 4 régions :



**RightRegion**

**MainRegion**

**LeftRegion**

**MenuRegion**

Il existe aussi une fenêtre qui sert à afficher une vue en mode read-only, appelée Fenêtre esclave dans le cahier des charges. Dans le projet elle s’appelle « SlaveWindow ».

La gestion de la navigation entre les vues est gérée par le service de navigation de Prism.

Comme décrit dans le cahier des charges, une vue peut être utilisée dans plusieurs écrans différents, on utilise donc le principe de « View Injection » qui permet une manipulation dynamique des vues dans les régions.

Ces vues et services sont chargées au lancement de l’application et ce chargement se fait dans la classe BootStrapper.

Principe de chargement de l’application :

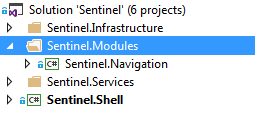
* Chargement des services dans le container
* Chargement des modules dans le catalogue des modules
* Affichage de la fenêtre de connexion
* Si l’utilisateur est authentifié, il affiche la fenêtre principale

## Les modules

Un module permet d’organiser les vues qui sont affichées dans l’application.

Les modules sont des projets de type Prism Module et sont stockés dans le répertoire de solution « Sentinel.Modules ».

Le module peut contacter un ou plusieurs services du container et a aussi accès a tout ce qui concerne l’infrastructure.



Ils sont chargés au démarrage de l’application par le BootStrapper.

Chaque projet de Modules contient une référence au projet Sentinel.Common.

Les modules sont :

|  |  |
| --- | --- |
| Noms | Descriptions |
| Sentinel.Navigation | Contient les vues nécessaires à la gestion de la navigation |
| Sentinel.BusinessScreens | Contient les vues nécessaires aux procédures KYC, 4Eyes Control, Favoris, relations et documents |
| Sentinel.Query | Contient les vues nécessaires au gestionnaire de requêtes |
| Sentinel.Security | Contient les vues nécessaires à la gestion de la sécurité |
| Sentinel.Scenario | Contient les vues nécessaires à la gestion des scenarii |
| Sentinel.CrossControl | Contient les vues nécessaires à la gestion des contrôles croisés |
| Sentinel.Dynamic | Contient les vues nécessaires à la gestion des écrans dynamiques |
| Sentinel.Configurations | Contient les vues nécessaires aux administrateurs fonctionnels |

### Module Navigation

Le module de navigation contient les éléments nécessaires pour la navigation dans l’application (Barres de menu, etc.).

Il contient les éléments suivants :

* Barre de menu principal
* Barre de menu de Navigation

### Module BusinessScreens

Ce module contient tous les éléments nécessaires à l’utilisation ou à la configuration éléments suivants :

* Procédures KYC
* 4 Eyes control
* Favoris
* Relations
* Documents

Pour chaque élément, on a créé un répertoire dans le projet du module afin de mieux organiser le code source.

### Module Query

Le module Query contient tous les éléments nécessaires à l’utilisation ou à la configuration des éléments suivants :

* Requêtes
* Tableaux de bord.

Pour chaque élément, on a créé un répertoire dans le projet du module afin de mieux organiser le code source.

### Module Security

Le module Security contient les éléments nécessaires à l’utilisation et la configuration de la sécurité applicative.

Il contient les vues pour les domaines suivants :

* Gestion utilisateur
* Gestion des groupes

### Module Scenario

Le module Scenario contient les éléments nécessaires à l’utilisation et la configuration d’un scenario.

### Module CrossControl

Le module Cross Control contient les éléments nécessaires à l’utilisation et la configuration d’un cross contol.

### Module Dynamic

Le module Scenario contient les éléments nécessaires à l’utilisation et la configuration du dynamisme.

### Module Configurations

Le module configurations contient les vues nécessaires aux administrateurs fonctionnels.

## Les Services

Il s’agit de classe qui implémente soit des méthodes communes ou soit des fonctionnalités de l’application. Ils sont chargés dans le gestionnaire de container dans la classe « BootStrapper ». Le gestionnaire de container va charger tous les services au lancement de l’application.

*Exemple de service fonctionnel (BU):*

Dans le service AuthenticationSService, il y a la méthode Authenticate qui va effectuer toutes les étapes nécessaires à l’authentification d’un utilisateur dans l’application. Il va donc exécuter les opérations suivantes :

* *Encryptage du mot de passe*
* *Création de la connexion à la db*
* *Création du repository User*
* *Requête via le repository User pour l’authentification*
* *Retour ou non d’un objet User*

Les services sont chargés au démarrage du bootstrapper dans la méthode « ConfigureContainer ».

Un service peut contacter d’autres services et a accès aus projets se trouvant dans le répertoire de solution « Sentinel.Infrastructure »

Un service peut aussi résulter du changement de l’interface graphique (changement de vue dans l’écran principal, affichage d’une nouvelle fenêtre, etc.).

Au niveau de Visual Studio, les Services sont représentés en 2 projets distincts :

|  |  |
| --- | --- |
| Noms | Descriptions |
| Sentinel.Services.Interfaces | Représente les interfaces de chaque service |
| Sentinel.Services | Représente les classes qui implémenteront les interfaces |

Les services sont :

|  |  |
| --- | --- |
| Noms | Descriptions |
| LoggerService | Permet d’écrire des logs techniques |
| AuthenticationService | Service permettant l’accès aux fonctionnalités de sécurité |
| ResourcesService | Permet d’effectuer les opérations sur les ressources utilisées tout au long de l’application |
| AuditService | Service permettant l’accès aux fonctionnalités d’audit |
| GlyphService | Service permettant l’accès aux images |
| ScenarioService | Service permettant l’accès aux fonctionnalités des scenarii |
| DynamicService | Service permettant l’accès aux fonctionnalités de la génération dynamique |
| KYCService | Service permettant l’accès aux fonctionnalités du KYC |
| FourEyesCheckService | Service permettant l’accès aux fonctionnalités du processus de validation « 4 Eyes » |
|  |  |

### LoggerService

Il s’agit d’un service permettant d’effectuer des opérations sur le journal technique afin de facilité le débogage de l’application.

Il contient les méthodes suivantes

|  |  |
| --- | --- |
| Noms | Descriptions |
| Write | Méthode permettant d’écrire un message dans le fichier de log dépendant du niveau de criticité passé en paramètre. |

### AuthenticationService

Le service permet d’effectuer des opérations de sécurité comme l’authentification, etc.

Il contient les méthodes suivantes :

|  |  |
| --- | --- |
| Noms | Descriptions |
| Authenticate | Permet d’authentifier un utilisateur. Si l’utilisateur est authentifié, alors il renvoi un objet User qui sera accessible via le service autrement il renvoi « null ». |
| AddUser | Permet d’ajouter un nouvel utilisateur |
| UpdateUser | Permet de mettre à jour un utilisateur |
| DeleteUser | Permet d’effacer un utilisateur |

### ResourcesService

Le service permet d’accéder aux ressources de l’application telles que les traductions

|  |  |
| --- | --- |
| Noms | Descriptions |
| GetTranslation | Permet de retourner les traductions pour un contrôle dans une langue. Ces paramètres sont passés en paramètre |
| DeleteTranslations | Permet d’effacer une traduction |
| AddTranslations | Permet d’ajouter des traductions |
| UpdateTranslations | Permet de mettre à jour des traductions |

### GlyphService

Le service permet d’effectuer les opérations sur les icônes de l’application. C’est-à-dire que l’on pet soit récupérer les images pour un contrôle ou en ajouter. Toutes ces actions sont faites via le service Glyph.

|  |  |
| --- | --- |
| Noms | Descriptions |
| GetImage | Retourne une image pour un contrôle passé en paramètre |
| AddImages | Ajoute une ou plusieurs images dans le fichier des ressources |
| DeleteImages | Efface une ou plusieurs images du fichier des ressources |

### AuditService

Le service permet d’effectuer les opérations d’audit de l’application. L’audit est une partie importante d’une application afin de savoir qui a modifié des informations ou ajouté ou encore effacé, etc.

|  |  |
| --- | --- |
| Noms | Descriptions |
| Write | Permet d’ajouter une trace dans le fichier d’audit. |

### ScenarioService

Le service permet de récupérer le scenario à exécuter sur l’événement d’un contrôle passé en paramètre.

Il permet également d’exécuter , ajouter, modifier ou effacer des scenarii.

|  |  |
| --- | --- |
| Noms | Descriptions |
| GetScenario | Retourne un scénario pour un contrôle et un événement passés en paramètre |
| UpdateScenario | Permet de mettre à jour un scenario |
| DeleteScenario | Permet d’effacer un scenario. |
| ExecuteScenario | Permet d’exécuter un scénario pour un événement d’un ABC passé en paramètre. |

## Les projets Infrastructure

Ce sont les composants transversaux qui seront utilisables par tous les éléments de l’application (Modules, Service, View Models, etc.)

Il est composé de 2 projets distincts

* Sentinel.DataAccess
* Sentinel.Common

### Projet DataAccess

Ce projet centralise toutes les classes qui permettent d’effectuer des opérations vers la base de données. Il est chargé au départ de l’application et n’est accessible que par les services.

Certains autres services renferment les fonctionnalités spécifiques de certains domaines fonctionnels et ils sont les seuls à pouvoir accéder à la base de données si nécessaire.

Pour ce faire on intègre l’entity Framework afin de générer les entités sur base des tables de la base de données. Ces entités sont utilisées dans tous les composants de l’application.

En ce qui concerne la façon d’exécuter les opérations sur les entités, on utilise le pattern « Repository » qui permet de définir les opérations possibles sur une entité.

Il contient également des fichiers « T4 », qui permettent de définir la façon dont Entity Framework va générer les entités et la classe du context.

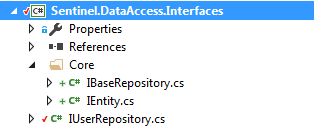
En effet, il est important d’utiliser le package Nuget « EntityFramework.CodeTemplates.CSharp » afin de définir la façon dont EF va générer les entités et le contexte (Exemple : définition des procédures stockées à lier, renommage des champs pointant vers la table ChoiceList, redéfinition de la méthode SaveChanges de la classe context afin d’y intégrer le comportement par défaut pour les champs d’audit techniques, etc …).

Ceci permet de gagner du temps lors de la regénération des entités avec EF.

En ce qui concerne l’organisation du projet dans VS2015, le service est divisé en 2 projets :

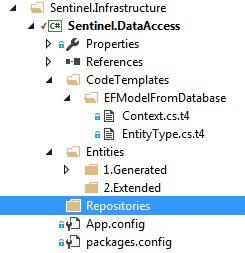
* Sentinel.DataAccess.Interfaces : *Regroupe tous les interfaces du service Data Access*
* Sentinel.DataAccess : *Regroupe l’implémentation des interfaces du projet Sentinel.DataAccess.Interfaces. Il doit donc référencer le projet Sentinel.DataAccess.Interfaces*

### Projet Service.DataAccess.Interfaces



Dans le répertoire « Core » se trouvent les interfaces de base des autres interfaces se trouvant à la racine du projet. Il existe une interface de repository par entité.

### Projet Sentinel.DataAccess



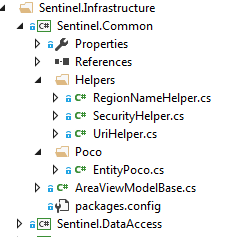
L’arborescence du projet contiendra les répertoires suivants :

* CodeTemplates : Répertoire utilisé par le package « T4 ». Il contient les fichiers T4 nécessaires à la personnalisation de la génération des entités et du contexte par « EF ».
* Entities : Regroupe toutes les entités générées et étendues.
  + 1.Generated : Contient toutes les entités et le model DB générés par EF. C’est dans ce répertoire que l’on régénèrera les entités et la classe de contexte si nécessaire.
  + 2.Extended : Contient toutes les classes entités redéfinies si nécessaire. Il s’agit de classes partielles.
* Repositories : Contient toutes les classes de repository, Pour rappel une classe de repository représente l’ensemble des actions possibles sur une entité. Il en faudra donc une par entité

Le projet possède des références vers les package NuGet suivants :

* EntityFramework
* EntityFramework.CodeTemplates.CSharp

### Projet Common



Cette partie de l’architecture met en exergue les classes helper et statiques qui seront disponibles pour tous les composants l’application (Helper, objets POCO, etc.).

Cette liste est non-exhaustive et peut évoluer au fur et à mesure des développements.

## Connexion



Etape 1 : L’utilisateur clique sur le bouton « Connexion », automatiquement la couche ViewModel intercepte l’évènement

Etape2 : La couche ViewModel Exécute la logique UI suivant le bouton qui a été appellé. Il y a 2 cas possibles

* Cas 1 : L’utilisateur à appuyer sur le bouton « Annuler » et l’application s’arrête
* Cas 2 : L’utilisateur a cliquer sur le bouton « Connexion » et la couche viewModel appelle la méthode d’authentification de la couche Model

Etape 3 : La couche Model Va appeler le service Security afin d’effectuer le processus d’authentification

Etape 4 : Le servie va effectuer la logique business pour l’authentification, à savoir :

1. Encryption du mot de passe
2. Appelle à la base de donnés pour récupérer l’utilisateur sur base du login et du mot de passe encrypté. Si l’utilisateur n’a pas été trouvé il renvoi null autrement il renvoi un objet Utilisateur

Etape 5 : La couche ViewModel recoit le réponse de la couche Model. Si le résultat est « null » alors il affichera un message d’erreur sur l’écran de Login autrement il ouvrira la page principale

Etant donné que la vue dédiée au logon de l’application n’est pas réutilisée dans d’autres écrans, il n’est pas nécessaire de créer un User control. Ceci veut dire la fenêtre de Logon est décrite comme une fenêtre à afficher.

La fenêtre se trouve dans le projet Shell et fait appel au service d’authentification.

Elle est affichée au démarrage de l’application et renvoie 2 valeurs possibles :

* Ok -> qui traduit que l’authentification de l’utilisateur s’est bien déroulée, ceci a pour effet d’ouvrir la fenêtre principale (définie par l’utilisateur dans ses préférences) en respectant ses droits par rapport à son/ses groupe(s) auxquels il appartient.
* Cancel -> qui traduit le fait d’appuyer sur le bout Annuler, ceci a pour effet de fermer l’application

L’affichage de l’écran se compose comme suit :

* Liste des bases de données : Liste résultant des chaînes de connexion (balises « ConnectionStrings » dans le fichier de configuration de l’application) déclarées dans le fichier de configuration. Par défaut on se positionne sur la base de données définie comme étant par défaut.
* Identifiant : Zone d’introduction de l’identifiant de l’utilisateur. Le focus est mis par défaut sur cette zone lors de l’ouverture de l’écran de login
* Mot de passe : Zone permettant à l’utilisateur d’entre son mot de passe. Cette zone cache le mot de passe à l’écran

## Exécution d’un événement (Scenario)

L’exécution d’un scénario se fait pour les cas suivants :

* Evénements sur une fenêtre (Loaded, Closes, etc…)
* Evènements sur un contrôle (click, LostFocus, SelectedItemChanged, etc.)



Pour ce faire on passe par un service qui s’occupe de récupérer le scénario a exécuté en fonction des paramètres de composants entrés en paramètre.

## Prism

Cette partie désigne l’utilisation du framework « Prism » et tous les concepts qui y découlent (BootStrapper, Module Manager, etc …). Ces services seront chargés au démarrage de l’application dans le container de module.

# Environnement Infrastructure

## Outils de développement

### Visual Studio 2015

Microsoft Visual Studio est une suite de logiciels de développement pour Windows conçue par Microsoft. La dernière version s'appelle Visual Studio 2015.

Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# utilisent tous le même environnement de développement intégré (IDE), qui leur permet de partager des outils et facilite la création de solutions faisant appel à plusieurs langages. Par ailleurs, ces langages permettent de mieux tirer parti des fonctionnalités du framework .NET, qui fournit un accès à des technologies clés simplifiant le développement d'applications web ASP et de services web XML grâce à Visual Web Developer.

Disponible depuis le 20 juillet 20155, cette nouvelle version apporte6 :

* la possibilité de se connecter avec plusieurs comptes ;
* le développement multiplateforme mobile (IOS, Android, Windows Phone) ainsi que le débogage ;
* la prise en charge du débogage des applications DirectX 12 pour les diagnostics des graphiques ;
* la possibilité de se connecter à divers services (Azure, Salesforce, Office 365) ;
* l'ajout de l'analyse dynamique ;
* l'amélioration de l'installation d'outils d'extensibilité ;
* l'ajout d'une fonctionnalité pour envoyer des commentaires auprès de Microsoft ;
* Cette version marque une fusion entre les éditions Premium et Ultimate pour simplifier le choix : trois éditions sont disponibles : Community, Professionnel et Enterprise

Cet outil de développement est utilisé pour le développement des nouvelles applications au sein de BDO. Il a été choisi, car toutes les applications windows et web sont déjà développées en C# au sein de BDO.

Le fait qu’il soit déjà utilisé au sein de BDO facilite le coaching en se concentrant que sur les parties non connues (Prism, EF, T4, etc…).

|  |  |
| --- | --- |
| **Requis** | **OS supportés** |
| **Composants Hardware**   * 1.6 GHz ou plus de vitesse de processeur * 1 GB de RAM (1.5 GB si on l’exécute sur une machine virtuelle) * 10 GB d’espace disque disponible * 5400 vitesse de disque dur * Carte video compatible avec DirectX 9 (1024 x 768 ou plus) | Windows 10 Windows 8.1 Windows 8 Windows 7 SP 1 Windows Server 2012 R2 Windows Server 2012 Windows Server 2008 R2 SP1 |

### SQL Server 2012

Microsoft SQL Server 2012 est un système de gestion de bases de données relationnelles (RDBMS) conçu pour l'environnement d'entreprise. Comme ses prédécesseurs, SQL Server 2012 comprend un ensemble d'extensions de programmation pour améliorer le langage SQL (Structured Query Language), un langage de programmation et d'apprentissage standard pour obtenir des informations et mettre à jour une base de données.

Ce produit est déjà utilisé au sein des projets chez BDO.

Les spécifications de mémoire et de processeurs suivants s'appliquent à toutes les éditions de SQL Server 2012 :

|  |  |
| --- | --- |
| **Composant** | **Conditions préalables** |
| Mémoire[1] | **Minimum :**  Éditions Express : 512 MO  Toutes les autres éditions : 1 GO  **Recommandé :**  Éditions Express : 1 GO  Toutes les autres éditions : au moins 4 GO, qui doivent être augmentés au fur et à mesure de l'augmentation de la taille de la base de données pour garantir des performances optimales. |
| Vitesse du processeur | **Minimum :**   * Processeur x86 : 1,0 GHz * Processeur x64 : 1,4 GHz   **Recommandé** : 2,0 GHz ou plus |
| Type de processeur | * Processeur x64 : AMD Opteron, AMD Athlon 64, Intel Xeon avec prise en charge Intel EM64T Intel Pentium IV avec prise en charge EM64T * Processeur x86 : compatible Pentium III ou supérieur |

## Configuration de déploiement

Cette partie décrit les différents environnements techniques possibles et le détail de leurs configurations minimum.

### Déploiement sur un seul poste



Ce chapitre décrit la configuration minimum nécessaire lorsque le client veut tout installer sur un seul poste.

Il faudra la configuration suivante :

* Type de pc : portable ou desktop
* Processeur x64 : 1,4 GHz minimum
* RAM : 8Go minimum
* Disque dur : 500Go minimum séparé en 2 partitions (« c: » Pour le système (200 Go) et « D: » (le reste) pour les datas)

### Déploiement sur plusieurs postes utilisateurs



Ce chapitre décrit la configuration minimum quant à l’installation de l’application sur plusieurs postes communiquant sur une base de données distantes.

La configuration du serveur peut être appliquée soit sur une machine réelle ou sur une machine virtuelle.

Configuration requise pour les machines clients :

* Technique
  + pc portable ou desktop
  + Processeur x64 : 1,4 GHz minimum
  + RAM : 8Go minimum
  + Disque dur : 500Go minimum séparé en 2 partitions (« c: » Pour le système (150 Go) et « D: » (le reste pour les datas)
* Logiciels
  + Windows 7 Pro minimum
  + Framework 4.5 installé
  + Application « Sentinel » installée

Configuration requise pour le serveur :

* Technique
  + Espace disque nécessaire :
    - Partition C (150 Go) : qui héberge les logiciels installés sur le serveur
    - Partition D (100 Go) : qui héberge la base de données 🡺 un backup de type journalier doit être activé
  + Processeur x64 : 1,4 GHz minimum
  + RAM : 8Go minimum
  + Disque dur : 500Go minimum séparé en 2 partitions (« c: » Pour le système (200 Go) et « D: » (le reste pour les datas)
* Logiciels
  + Système d’exploitation :Windows Server 2008 minimum (Conseillé 2012 DataCenter)
  + Sql Server 2012
  + Framework .Net 4.5 redistributable package

# Documentation technique

## Organisation d’un appel DB

Lorsque l’application doit récupérer des informations contenues en base de données, il suit les étapes suivantes :

* Appel du Service qui lui permettra de récupérer les informations
* Le service effectue les appels vers la base de données en utilisant les repository
* Le service récupère le contenu et retourne le résultat
* La vue utilise les objets retournés pour l’affichage dans les vues

## Organisation des événements(scenarii)

L’application est organisée en composants suivants :

* Area (A) : Vue qui sera affichée
* Body (B) : Rubrique qui contiendra les contrôles
* Control (C) : Contrôle à afficher dans la rubrique

Pour chaque contrôle de type « ABC » il existe une liste d’événements possibles suivant le type.

Exemple:

* Button 🡪 BeforeClick, Afterclick
* Area 🡪 AfterLoad
* Etc.

A un événement est toujours lié un et un seul scenario. Chaque scénario comporte des conditions et pour chaque condition une liste d’opérations à exécuter. Ces opérations seront exécutées si la condition est vraie.

### Chargement des scenarii

La durée de vie de la liste de scenarii à exécuter pour une « Area » est la durée de vie d’une « Area ». Cela veut dire la liste est créée à la création de la vue et elle est détruite lors du « dispose » de la vue.

Au chargement, la vue contacte le service « Scenarii » afin de recevoir la liste des événements possibles pour la vue et ses sous-éléments.

Pour chaque élément de la vue, on va initialiser un objet de classe « SentinelCommand ». Cette classe implémente l’interface « ICommand » et permet d’encapsuler le comportement par défaut des événements.

Le comportement d’un événement lié à une instance de classe « SentinelCommand » est le suivant :

* Exécution du scenario before de l’événement s’il a été défini dans la liste
* Exécution du code de l’événement
* Exécution du scénario After de l’événement s’il a été défini dans la liste