******

**TPI 2017**

*Travail professionnel individuel*

*Informaticien-ne CFC*

StreamScan

Table des matières

Résumé du rapport du TPI 3

1 Introduction 4

1.1 Analyse de la situation initiale 4

1.2 Analyse de l’état désiré 4

1.3 Cahier des charges 4

2 Analyse 5

2.1 Objectifs 5

2.2 Rentabilité 5

2.3 Fonctionnalités 5

2.4 Planification des étapes du développement 6

2.5 Diagramme d’utilisation 6

2.6 Diagrammes d’activité 7

2.7 Schéma global de base de données 14

2.8 Modèle de Conception de Données (MCD) 14

3 Conception 15

3.1 Architecture du système 15

3.2 Modèle Logique de Données (MLD) 16

3.3 Maquettes 17

3.4 Diagramme de classes 20

3.5 Planification des tests 21

3.6 Tests spéciaux 25

3.7 Concept d‘implémentation 26

3.8 Tests technologiques 28

3.9 Librairie utilisées 31

3.10 Utilisation de la librairie QueriesManager 31

4 Implémentation 32

4.1 DAL 32

5 Tests 35

6 Problèmes rencontrés 36

6.1 Version de la librairie StreamScanCommon différente 36

7 Bibliographie: liste des sources et références 36

8 Glossaire 36

9 Signatures 36

10 Annexes 37

10.1 Première visite des experts 37

10.2 Deuxième visite des experts 37

# Résumé du rapport du TPI

Le Management Summary est une présentation conceptuelle du travail effectué et du résultat attendu d’au maximum une page A4 qui permettent au lecteur une rapide compréhension du rapport de travail.

# Introduction

StreamScan est une application permettant l’obtention et l’enregistrement de diverses données matérielles et logicielles depuis une application serveur lancée sur la machine dont on souhaite extraire les informations.

Le produit StreamX, développé par Infoteam, est une passerelle de communication pour différents protocoles. Il est installé sur plusieurs types de machines (MOXA, Beckhoff,…), sous différents OS (Windows XP, Windows 7, Linux,…) et est configuré selon le client.

L’installateur de StreamX est mis à jour régulièrement et mis à disposition des clients sur la plate-forme d’échange afin qu’ils aient accès aux dernières fonctionnalités.

L’application **StreamScan** sera utilisée afin de facilité l’obtention des informations des machines ainsi que leur version de StreamX.

L’application va permettre de :

* Reproduire des situations sur la plate-forme de tests dans un environnement similaire à l’environnement des clients
* Identifier les installations à mettre à jour
* Montrer au client son acquis et son évolution potentielle
* Gagner du temps lors d’intervention chez les clients

## Analyse de la situation initiale

En l’état actuel, aucun moyen ne permet de connaître ces informations de manière centralisée ce qui engendre une perte de temps lors d’interventions.

La partie serveur, installée sur chaque passerelle de communication, a déjà été réalisée en prérequis au travail, afin d’acquérir les connaissances nécessaires à la réalisation de ce projet.

Celle-ci utilise diverses librairies afin de récupérer les informations système (OS, nom du produit, version,…), matérielles (nom du produit, fabriquant,…), de la carte réseau (IP, passerelle, masque de sous-réseau,…) et les informations relatives à **StreamX**. Le client peut alors s’y connecter afin de récupérer ces informations.

Pour plus d’informations concernant la partie serveur, se référer à la documentation de cette dernière (cf. : Annexes).

## Analyse de l’état désiré

Le produit fini permettre d’obtenir les informations d’une machine StreamX grâce à son IP et stockant les données dans une base de données MySQL.

Celles-ci sont ensuite listées selon l’entreprise et l’ouvrage de la machine. Un filtre (recherche) peut être appliqué permettant ainsi une meilleure lisibilité. Les situations critiques sont mises en évidence (ex : ancienne version).

Les informations peuvent être mises à jour en sélectionnant la machine.

## Cahier des charges

Voici les exigences de l’application :

* Obtention des informations d’une machine StreamX grâce à son IP
* Sauvegarde des informations récoltées dans une base de données MySQL selon l’ouvrage et l’entreprise
* Présentation des informations de la base de données sur une interface web en mettant en évidence certaines situations critiques comme un disque plein ou une version ancienne du produit.
* Recherche des informations d’une machine selon un critère
* Développement de l’application en ASP.NET
* Accès à l’application géré par une page de login
* Rédaction d’un manuel utilisateur, en plus de la documentation technique
* Réalisation de l’application en anglais

Le cahier des charges **complet** est disponible en annexes

Le point suivant du cahier des charges a été modifié :

 Standards d’entreprise – Langue anglaise pour la documentation du code

Afin d’éviter une perte de temps quant à la rédaction en anglais, celle-ci se fera en français.

### Informations à obtenir

Les informations à obtenir depuis l’application serveur sont les suivantes :

* Le système d’exploitation utilisé
* Le fabricant et le modèle du PC
* La capacité de la mémoire vive
* La capacité du disque dur
* Le taux d’utilisation du disque dur
* Les composants StreamX installés
* La version du setup
* La date de la dernière mise à jour du setup
* La version de l’applicatif du produit StreamX
* Les protocoles employés
* Les dossiers de Log créés

# Analyse

## Objectifs

Les objectifs sont les mêmes que spécifiés dans le cahier des charges, à savoir :

* Pouvoir obtenir les infos d’une machine StreamX
* Stocker ces infos dans la base de données
* Récupérer l’ensemble des infos des machines et les afficher au client
* Permettre à ce dernier d’effectuer une recherche sur ces infos
* Ajout/modification/suppression d’une entreprise/ouvrage
* Réaliser une documentation utilisateur

## Rentabilité

Le projet étant interne à l’entreprise et non pas pour un client, le critère de rentabilité ne rentre pas en compte. Il doit cependant être réalisé avant la fin du stage.

## Fonctionnalités

À partir du cahier des charges, on peut définir les fonctionnalités suivantes :

* Obtention des infos d’une machine StreamX (scan)
* Voir la liste des machines StreamX à partir de la base de données en fonction de l’entreprise et de l’ouvrage
* Mettre à jour les informations de la base de données à partir d’un scan

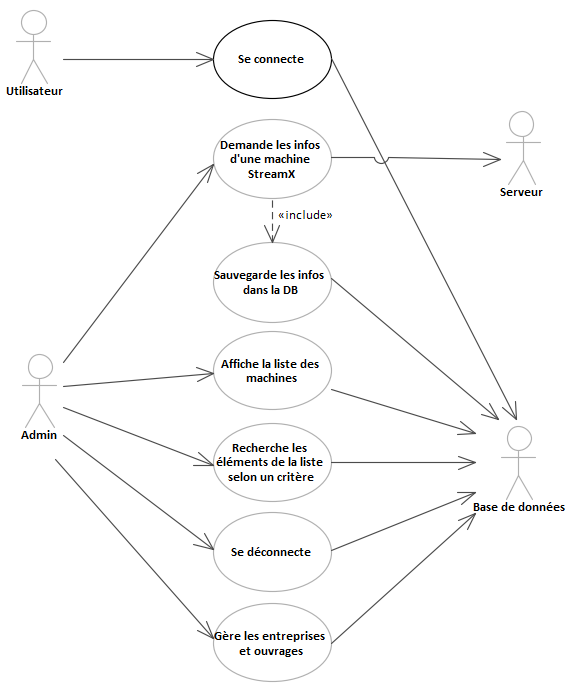
## Planification des étapes du développement

La partie développement sera organisée dans l’ordre suivant :

1. Structuration de l’application (création des pages, méthodes, fonctionnalités globales)
2. Partie authentification (page de connexion)
3. Partie récupération des infos d’une machine StreamX (scan) + Tests fonctionnels
4. Partie ajout, modification et suppression des entreprises/ouvrages
5. Partie mise à jour des données du scan dans la base de données
6. Partie récupération des informations des machines à partir de la base de données
7. Tests fonctionnels de l’application

## Diagramme d’utilisation

L’outil « Enterprise Architect » a été utilisé car il est simple et c’est l’outil que l’on utilise à l’école.



**Acteurs**

Utilisateur Utilisateur non-authentifié

Admin Utilisateur authentifié

Serveur Serveur de l’application, disposé sur la machine StreamX

Base de données Base de données MySQL de l’application

**Cas d’utilisation**

Se connecte

Connecte l’utilisateur en vérifiant ses identifiants dans la base de données

Demande les infos d’une machine StreamX

Demande au serveur ses informations

Affiche la liste des machines

Affiche la liste des machines en contactant la base de données

Recherche les éléments de la liste selon un critère

Affiche les éléments de la base de données selon un critère spécifié par l’admin

Se déconnecte

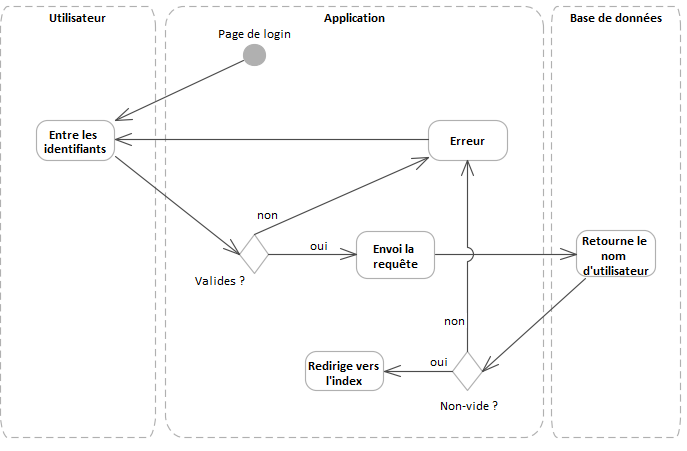
Déconnecte l’admin de l’application

Gère les entreprises et ouvrages

Permet la création, modification et suppression d’entreprises ou d’ouvrages

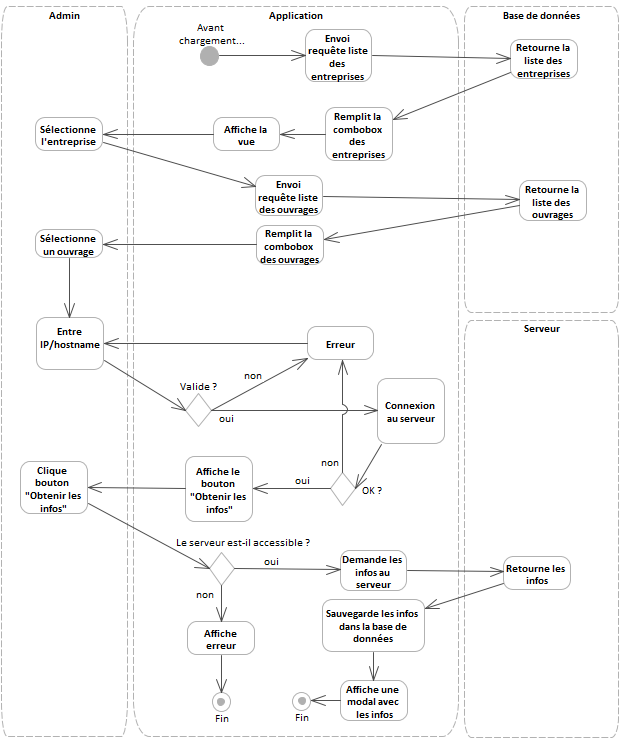
## Diagrammes d’activité

### Se connecte



Suite au chargement de la page, l’utilisateur saisit ses identifiants. S’ils sont invalides, on affiche une erreur. Dans le cas contraire, une requête à la base de données afin de contrôler l’existence de l’utilisateur. Elle retourne le nom de l’utilisateur en question. Si celui-ci est vide, on dit à l’utilisateur que ses identifiants sont incorrects. Sinon, on le redirige vers l’index.

### Demande les infos d’une machine StreamX



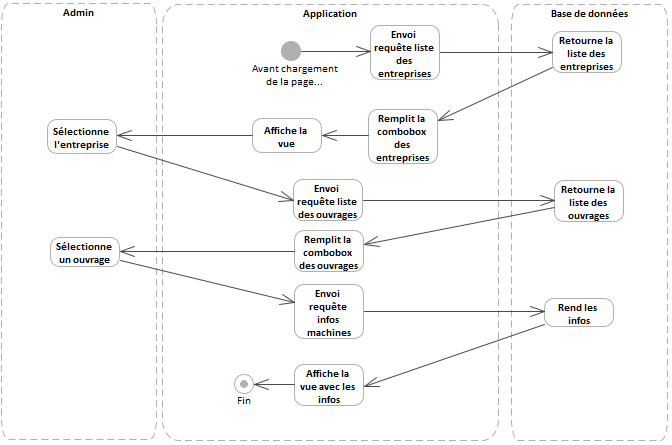
Premièrement, la liste des entreprises est obtenue à partir de la base de données. Elle est ensuite appliquée à la combobox des entreprises et on affiche la vue.

Le client sélectionne l’entreprise voulue et valide. La liste des ouvrages est ensuite récupérée et est ajoutée à la combobox des ouvrages.

Le client sélectionne alors un ouvrage ainsi que l’adresse de la machine dont il souhaite récupérer les informations. Si l’adresse entrée est valide, on contrôle que le serveur soit accessible. Si ce n’est pas le cas, on avertit l’admin via une erreur. Dans le cas contraire, on affiche le bouton « Obtenir les infos ».

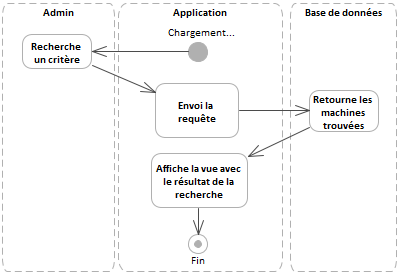
En cliquant sur le bouton, on vérifie que le serveur soit toujours accessible et si c’est le cas, on envoie la requête au serveur. Les infos sont ensuite sauvegardées dans la base de données puis on affiche une fenêtre modal avec les infos.

### Affiche la liste des machines



Avant le chargement de la page, on récupère la liste des entreprises qu’on met dans une combobox puis on affiche la vue. L’admin sélectionne l’entreprise. Les ouvrages de cette entreprise sont récupérés depuis la base de données puis on remplit une autre combobox. Suite à la sélection de l’ouvrage, on récupère les machines correspondantes et on les affiche.

### Recherche les éléments de la liste selon un critère

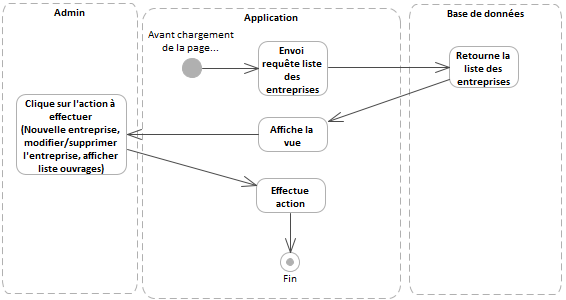


Suite au chargement de la liste des machines, le client peut rechercher un critère. En le validant, la requête est envoyée au serveur qui retourne les machines correspondantes au critère. La vue est ensuite mise à jour pour correspondre au résultat.

### Se déconnecte

Aucun diagramme d’activité n’a été conçu en raison de sa très grande simplicité.

### Gestion des entreprises et des ouvrages

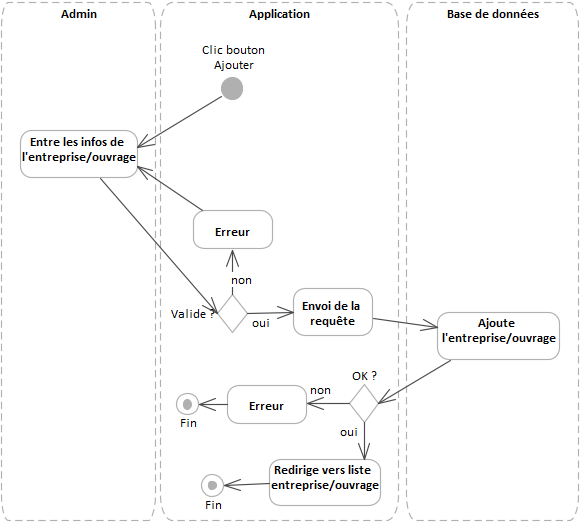


Premièrement, on récupère la liste des entreprises et on affiche la vue. Le client peut ensuite effectuer plusieurs actions :

* Ajouter une nouvelle entreprise
* Modifier une entreprise existante
* Supprimer une entreprise existante
* Afficher les ouvrages correspondants à une entreprise

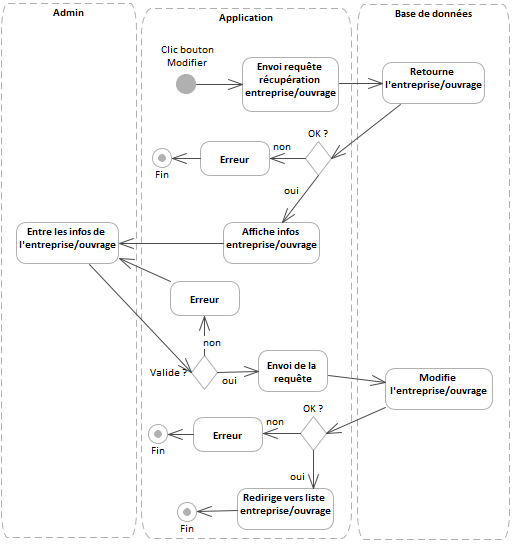
Note : La dernière action affiche une vue similaire à celle-ci, c’est pourquoi aucun diagramme n’y est consacré.

#### Ajoute une entreprise/ouvrage



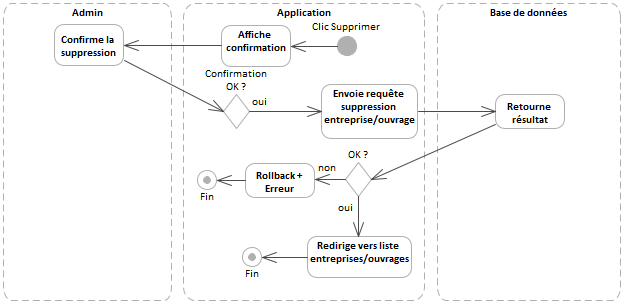
Lorsque l’admin appuie sur le bouton «  Ajouter », celui-ci doit entrer les infos de l’entreprise/ouvrage et valider. Si les données entrées sont valides, alors la requête est envoyée à la base de données qui ajoute l’entreprise/ouvrage. Si tout c’est bien passé, l’admin est redirigé vers la liste des entreprises/ouvrages.

#### Modifie une entreprise/ouvrage



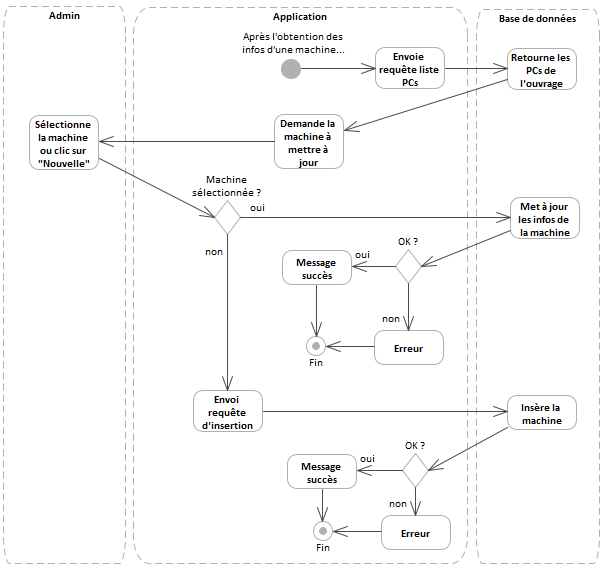
Lorsque l’admin appuie sur le bouton « Modifier », une requête est envoyée à la base de données afin de récupérer les infos de l’entreprise/ouvrage. L’utilisateur modifie les infos de l’entreprise/ouvrage et valide. Si les données entrées sont valides, alors la requête est envoyée à la base de données qui ajoute l’entreprise/ouvrage. Si tout c’est bien passé, l’admin est redirigé vers la liste des entreprises/ouvrages.

#### Supprime une entreprise/ouvrage



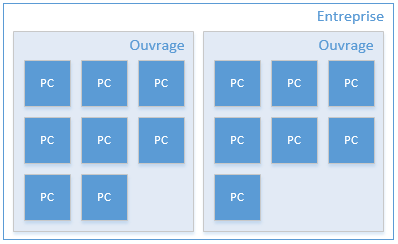
Lorsque l’admin appuie sur le bouton « Supprimer », une fenêtre de confirmation est affichée. Si l’admin appuie sur « OK », on envoie la requête de suppression au serveur. Si elle s’est correctement déroulée, on affiche la liste des entreprises/ouvrages. Sinon on fait un « rollback » de la requête et on affiche une erreur.

### Sauvegarde des informations d’une machine StreamX dans la base de données



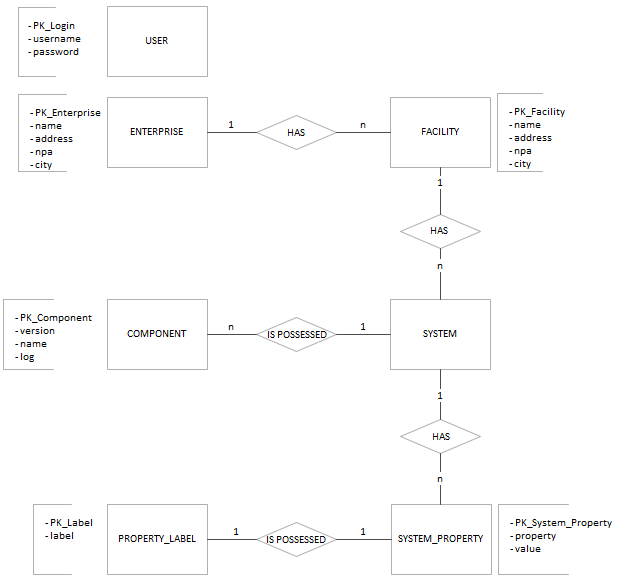
Après l’obtention des infos d’une machine, on récupère tous les PCs de l’ouvrage. Puis, on demande au client quelle machine il veut mettre à jour ou s’il souhaite en ajouter une nouvelle. Dans le premier cas on envoie une requête « INSERT » à la base de données et dans le deuxième cas, une requête « UPDATE ».

## Schéma global de base de données



La base de données est composée d’entreprises. Celles-ci contiennent elles-mêmes des ouvrages. Les PC StreamX sont placés dans ces ouvrages.

## Modèle de Conception de Données (MCD)



Un PC est modélisé sous deux tables, à savoir : « System » et « System\_Property ». La raison à cela est qu’il est facile d’imaginer l’ajout d’autres propriétés pour un PC. Cette solution a été choisie afin de rendre cet ajout plus simple et de ne pas avoir à changer la structure de la base de données.

Une entreprise (Enterprise) a plusieurs ouvrages (Facility) mais un ouvrage ne correspond qu’à une seule entreprise.

Un ouvrage a plusieurs PC (System) mais un PC n’appartient qu’à un seul ouvrage.

Un PC a plusieurs composants (Component) mais un composant n’est possédé que par un seul PC.

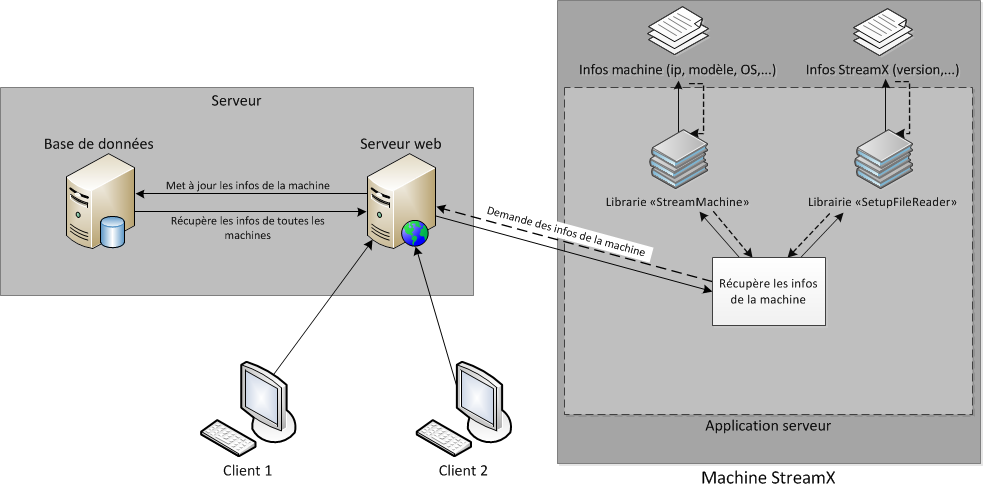
Un PC a plusieurs propriétés (System\_Property) mais une propriété ne caractérise qu’un seul PC.

Une propriété ne possède qu’un label et un label n’est possédé que par une seule propriété.

Enfin, la table « User » permet le stockage des données des utilisateurs.

# Conception

## Architecture du système



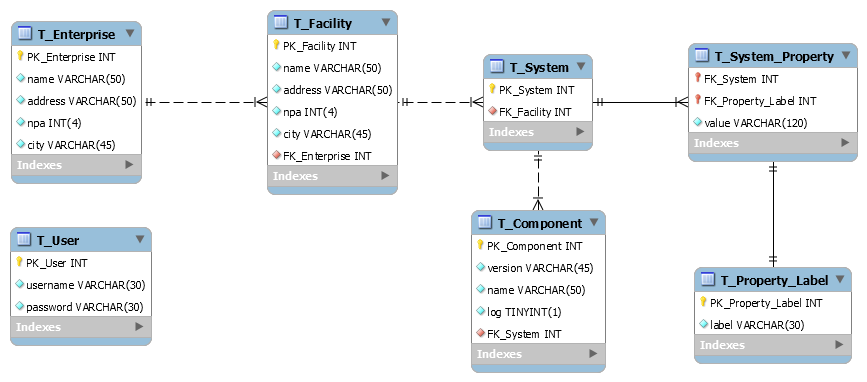
L’application cliente est composée d’un serveur web ainsi que d’une base de données.

La base de données, conformément au MCD, s’occupe de stocker les entreprises, les ouvrages ainsi que les machines StreamX. Ces données seront mises à disposition de l’application afin de les afficher.

Le serveur web héberge l’application cliente StreamScan. Il récupère les données d’une machine StreamX, en contactant l’application serveur StreamScan disposée sur cette dernière, et les stocke dans la base de données.

L’application serveur utilise plusieurs librairies afin de récupérer les informations de la machine et de les transmettre au client.

## Modèle Logique de Données (MLD)



Remarque : Dans la table « T\_System\_Property », on définit la « FK\_System » et la « FK\_Property\_Label » en tant que PK afin qu’un « PC » puisse avoir plusieurs propriétés mais qu’une seule de chaque type.

### Longueur des types

Enterprise, Facility

Name, address Longueur adéquate pour un nom d’entreprise/ouvrage ou une adresse

Npa Un NPA ne dépasse jamais 4 caractères

City Définit en prenant le nom de ville le plus long en France

System Property

Value La valeur pouvant être un texte, une longueur conséquente a été prise

Component

Version Longueur adéquate pour une version, ne sachant pas encore le nombre de digit la composant

Name Longueur générale

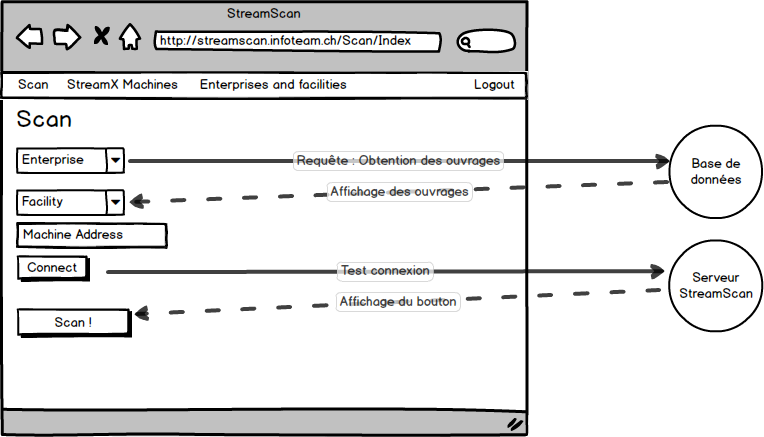
Log Ne peut accueillir qu’un entier de longueur 1 (= booléen)

Property Label

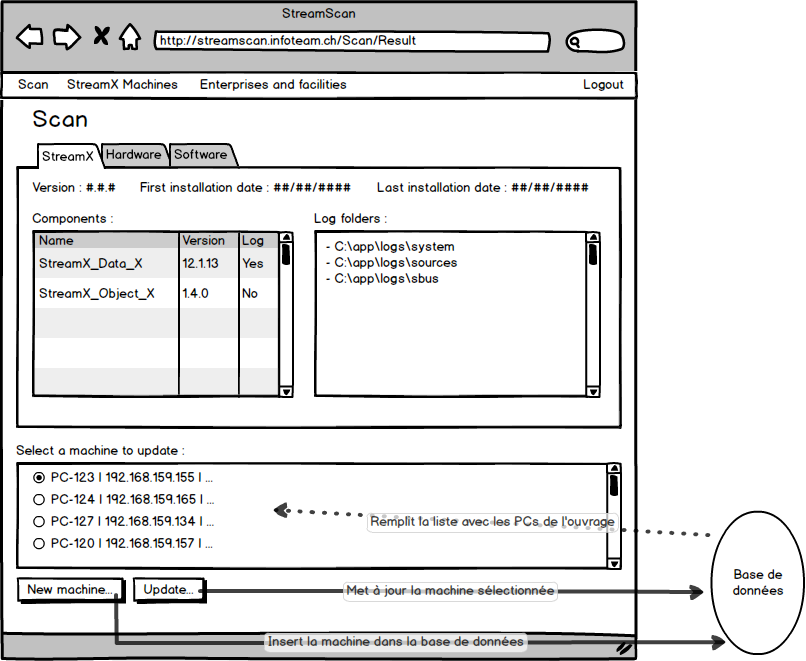
label Longueur adéquate pour un label

## Maquettes

### Scan

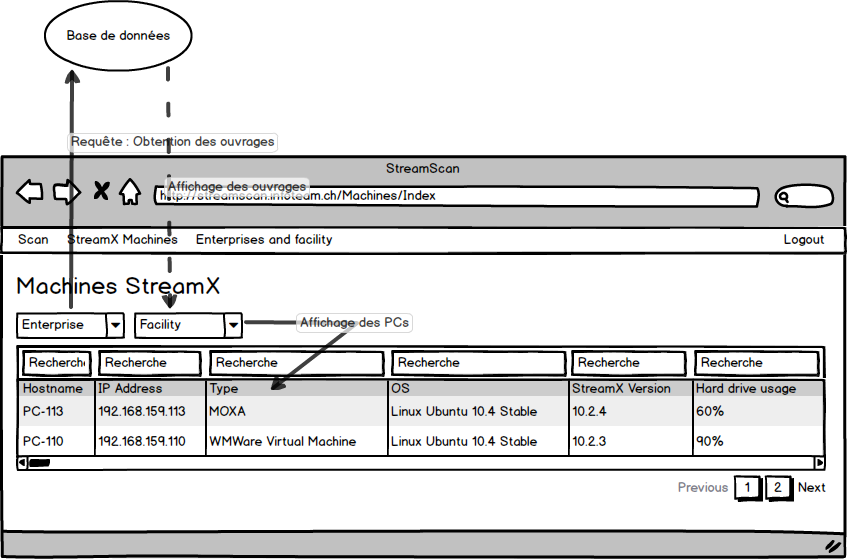


La combobox de l’entreprise, une fois sélectionnée, fait apparaître le formulaire de connexion à la machine. On sélectionne l’ouvrage et on saisit l’adresse de la machine. Le bouton « Connect » connecte l’utilisateur à la machine serveur puis affiche le bouton de scan. Tout ce procédé est fait en Ajax. Le scan affiche la fenêtre modale suivante :



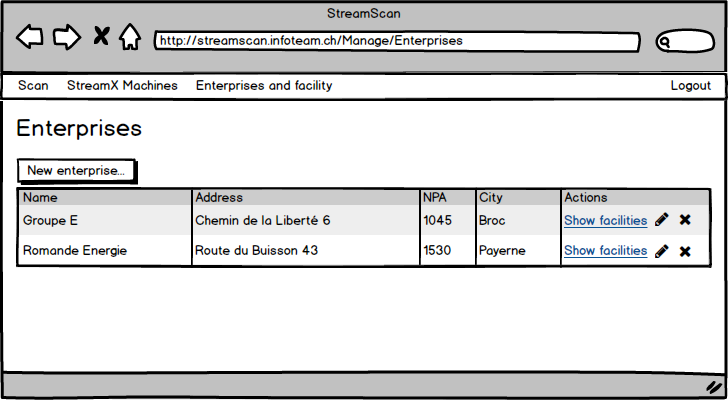
La partie du haut permet de visualiser le résultat de la requête. Celle du bas permet de mettre à jour les infos d’une machine dans la base de données ou d’en insérer une nouvelle.

### StreamX Machines

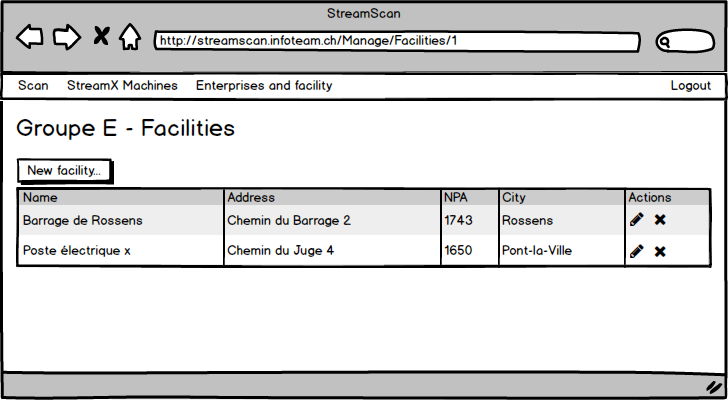


En sélectionnant l’entreprise, on fait apparaître la liste de toutes les machines de celle-ci. Ensuite, en sélectionnant l’ouvrage, on restreint le tableau aux machines de l’ouvrage seulement.

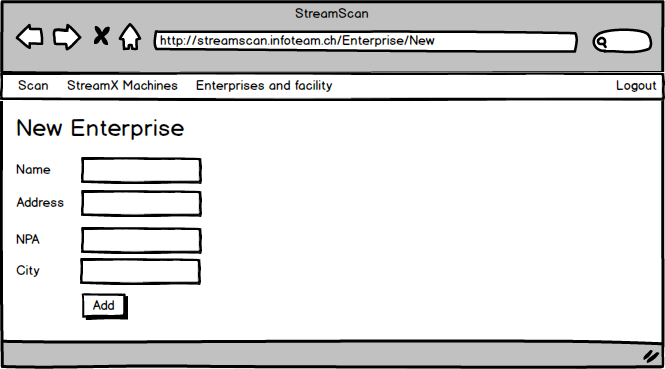
### Enterprises



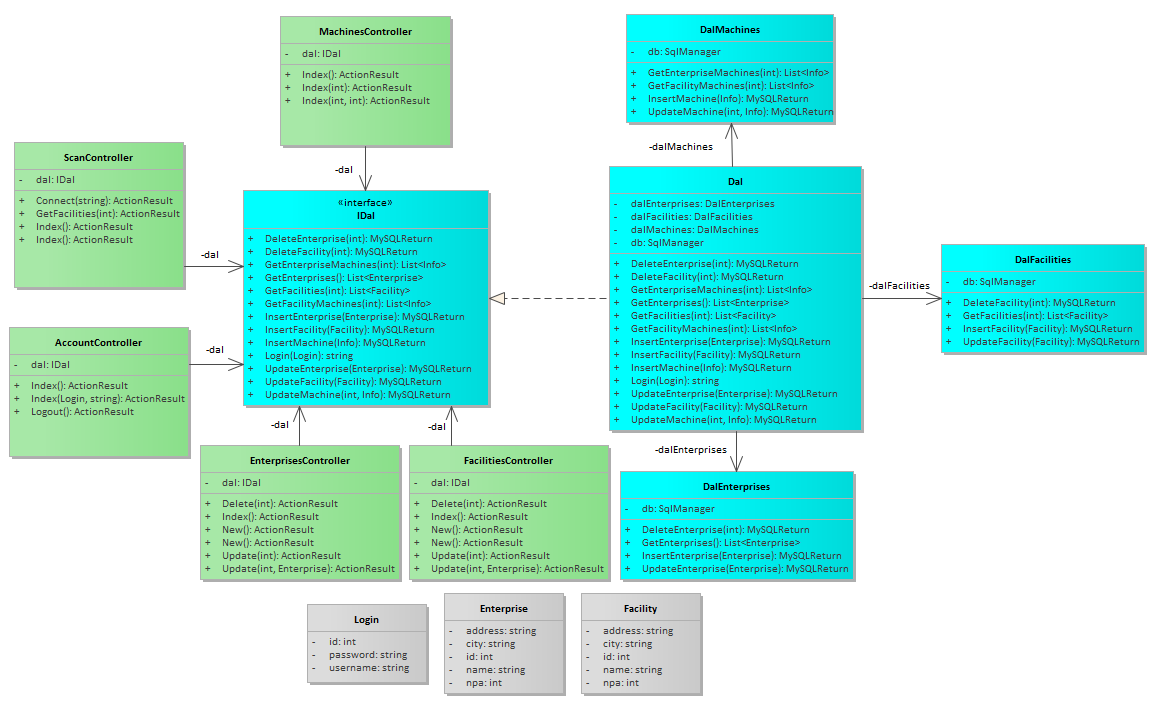
### Facilities



### New enterprise/facility



## Diagramme de classes

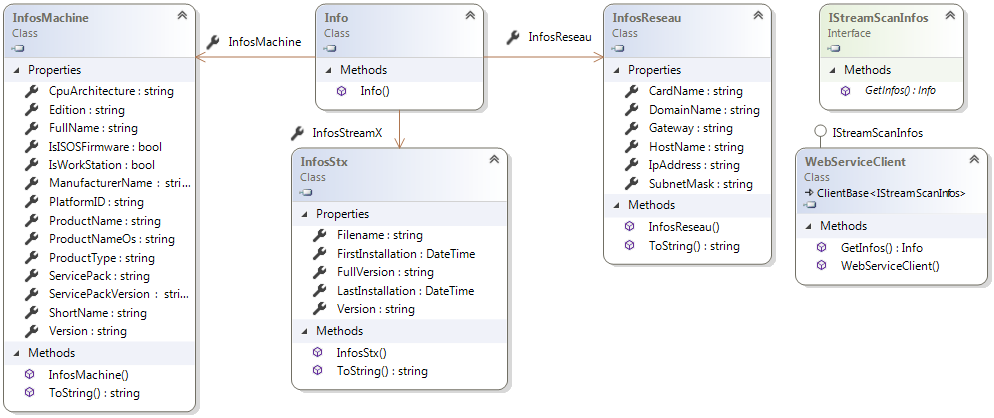


Les « controller » (vert) appellent les vues HTML et font la liaison avec le modèle.

Le modèle (bleu) permet l’accès à la base de données. Les classes roses (équivalent de JavaBean) font également parties du modèle.

Les noms de classes SqlManager et MySqlReturn sont issus de la librairie QueriesManager permettant l’exécution de requêtes SQL (cf. : Chapitre consacré).

La classe Info vient de la librairie StreamScanCommon (cf. : « Librairies utilisées ») permettant le stockage des infos lors de l’envoi depuis le serveur. Voici le digramme de celui-ci :



## Planification des tests

Les tests fonctionnels se font selon les pages suivantes. Les tests spéciaux (cf. : Chapitre consacré) sont mis en italique.

### Login

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire de connexion | Login avec identifiants incorrects | Message d’erreur |  |
| Formulaire de connexion | Login avec le nom d’utilisateur/mot de passe vide | Message d’erreur en dessous de l’élément vide |  |
| Formulaire de connexion | Login avec des identifiants corrects | Message de succès + redirection |  |

### Scan

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Sélection de l’entreprise | Sélection de l’entreprise | Affichage du formulaire de connexion à la machine |  |
| Sélection de l’entreprise | Sélection d’aucune entreprise | Affichage erreur |  |
| Sélection de l’ouvrage | Sélection de l’ouvrage | Aucune erreur |  |
| Sélection de l’ouvrage | Sélection d’aucun ouvrage | Affichage erreur |  |
| Formulaire de connexion à la machine | Connexion avec une adresse incorrecte | Message d’erreur en dessous du champ |  |
| Formulaire de connexion à la machine | Connexion avec une adresse correcte | Affichage du bouton de scan |  |
| Formulaire de scan | Récupération des infos si la connexion a été perdue | Message d’erreur |  |
| Formulaire de scan | Récupération des infos sur un serveur existant | Affichage des infos avec le formulaire de mise à jour de la base de données |  |
| Formulaire de mise à jour de la DB | Clic du bouton de mise à jour si l’élément n’a pas été modifié depuis | Met à jour l’élément sélectionné |  |
| *Formulaire de mise à jour de la DB* | *Clic du bouton de mise à jour si l’élément a été modifié depuis (cf. : « Tests spéciaux – C »)* | *Affichage message d’erreur et rechargement de la liste* |  |
| *Formulaire d’ajout à la DB* | *Clic du bouton d’ajout si l’ouvrage n’existe plus (cf. : « Tests spéciaux – A »)* | *Affichage d’un message d’erreur et rechargement de la page* |  |
| *Formulaire d’ajout à la DB* | *Ajouts simultanés (cf. : « Tests spéciaux – E »)* | *Affichage au deuxième utilisateur* |  |
| Formulaire d’ajout à la DB | Clic du bouton d’ajout | Ajout de la machine dans la base de données |  |

### StreamX Machines

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Sélection de l’entreprise | Sélection de l’entreprise | Affichage des ouvrages + liste toutes les machines de l’entreprise |  |
| Sélection de l’ouvrage | Sélection de l’ouvrage | Affichage des machines de l’ouvrage |  |

### Enterprises

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Bouton « New enterprise » | Clic du bouton « New Enterprise » | Affichage du formulaire d’ajout |  |
| Bouton « Update » | Clic du bouton « Update » | Affichage du formulaire de mise à jour |  |
| Lien « Show facilities » | Clic du lien « Show facilities » | Affichage des ouvrages |  |
| Bouton « Delete » | Clic du bouton « Delete » | Affichage d’une popup de confirmation |  |
| Formulaire de confirmation | Clic « Cancel » | Disparition de la popup |  |
| *Formulaire de confirmation* | *Clic « Delete » si déjà supprimé (cf. : « Tests spéciaux – D »)* | *Affichage message d’erreur* |  |
| Formulaire de confirmation | Clic « Delete » | Suppression de l’entreprise (et de ses ouvrages + machines) |  |

### Facilities

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Bouton « New facility » | Clic du bouton « New facility » | Affichage du formulaire d’ajout |  |
| Bouton « Update » | Clic du bouton « Update » | Affichage du formulaire de mise à jour |  |
| Bouton « Delete » | Clic du bouton « Delete » | Affichage d’une popup de confirmation |  |
| Formulaire de confirmation | Clic « Cancel » | Disparition de la popup |  |
| *Formulaire de confirmation* | *Clic « Delete » si déjà supprimé (cf. : « Tests spéciaux – D »)* | *Affichage message d’erreur* |  |
| Formulaire de confirmation | Clic « Delete » | Suppression de l’ouvrage (et de ses machines) |  |

### New enterprise

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire d’ajout | Ajout avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire d’ajout | Ajout avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| Formulaire d’ajout | Ajout de l’entreprise | Message de succès + redirection vers la liste |  |

### New facility

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire d’ajout | Ajout avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire d’ajout | Ajout avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| *Formulaire d’ajout* | *Ajout dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Message d’erreur* |  |
| Formulaire d’ajout | Ajout de l’ouvrage | Message de succès + redirection vers la liste |  |

### Update enterprise

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire de mise à jour | Mise à jour avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| *Formulaire de mise à jour* | *Mise à jour dont les données d’origine diffèrent de celles de la DB (cf. : « Tests spéciaux – C »)* | *Message d’erreur* |  |
| Formulaire de mise à jour | Mise à jour de l’entreprise | Message de succès + redirection vers la liste |  |

### Update facility

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire de mise à jour | Mise à jour avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| *Formulaire de mise à jour* | *Mise à jour dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Message d’erreur* |  |
| *Formulaire de mise à jour* | *Mise à jour dont les données d’origine diffèrent de celles de la DB (cf. : « Tests spéciaux – C »)* | *Message d’erreur* |  |
| Formulaire de mise à jour | Mise à jour de l’ouvrage | Message de succès + redirection vers la liste |  |

## Tests spéciaux

Étant dans un contexte multi-utilisateurs, les opérations dans la base de données sont à gérer correctement (c.-à-d. verrouillage de tables, transactions,…).

#### Requête contenant une clé étrangère inexistante

Erreur SQL (#1452) survenant dans les cas suivant :

* Insertion d’une machine dans un ouvrage supprimé
* Insertion/modification d’un ouvrage dans une entreprise supprimée

Si l’on prend le premier cas :

Pendant qu’un utilisateur effectue un scan, l’ouvrage dans lequel il opérait est supprimé.

#### Elément supprimé lors d’une mise à jour de celui-ci

Par exemple, si un ouvrage est supprimé par un utilisateur pendant qu’un autre le met à jour. Test si :

* Mise à jour de l’élément supprimé
* Mise à jour de l’élément existant

#### Perte de mise à jour

Ici, deux mises à jour se font au même moment. Les données mises à jour par l’utilisateur 1 seront perdues mais les deux utilisateurs recevront un message de succès.

Pour pallier à ce problème, la requête UPDATE effectuée ne vérifiera pas que par la clé primaire mais par toutes les propriétés de la table. Les requêtes sont faites dans une transaction et si une requête échoue, le « rollback » est effectué. Par exemple :

Source image :

Document de formation transactions (module 223 – EMF)

[UPDATE](http://cpanel.emf-informatique.ch/cpsess1812177458/3rdparty/phpMyAdmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/update.html) T\_System\_Property [SET](http://cpanel.emf-informatique.ch/cpsess1812177458/3rdparty/phpMyAdmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/set.html) value = "" WHERE value = "x86" AND PK\_System\_Property = 4 AND FK\_System = 1 AND FK\_Property\_Label = "5"

On peut voir qu’on spécifie toutes les propriétés de la table.

Le test suivant est appliqué à toutes les requêtes UPDATE de l’application :

* Modification d’une entrée modifiée
* Modification d’une entrée non-modifiée

#### Suppression d’un élément déjà supprimé

Si, par exemple, on souhaite supprimer un ouvrage qui a déjà été supprimé par un autre utilisateur depuis qu’on a affiché la liste, il faut afficher un message d’erreur à l’utilisateur.

On effectue le test suivant :

* Suppression d’un élément supprimé
* Suppression d’un élément existant

Le test survient dans les cas suivant :

* Suppression d’un ouvrage
* Suppression d’une entreprise

#### Insertion des propriétés d’une machine StreamX

Comme la partie de la base de données concernant les machines a été faite selon un modèle optimisé, l’insertion d’une nouvelle machine se fait sous deux tables, à savoir T\_System et T\_System\_Property. La table T\_System devra être verrouillée après l’ajout de l’entrée car on récupère la clé lors de l’ajout dans la table T\_System\_Property.

## Concept d‘implémentation

L’implémentation est faite en **ASP.NET** et hébergée sur un serveur **IIS**.

### Design

Afin d’obtenir une application responsive et attractive sans fournir d’effort et de manière rapide, le framework CSS **Bootstrap** est utilisé.

En plus de celui-ci, le framework **FluentBootstrap** est aussi employé. Ce dernier permet de générer des éléments HTML avec Bootstrap en écrivant du code *C#* à la place du HTML classique. Il reprend le même principe que les helpers HTML disponibles en ASP.NET.

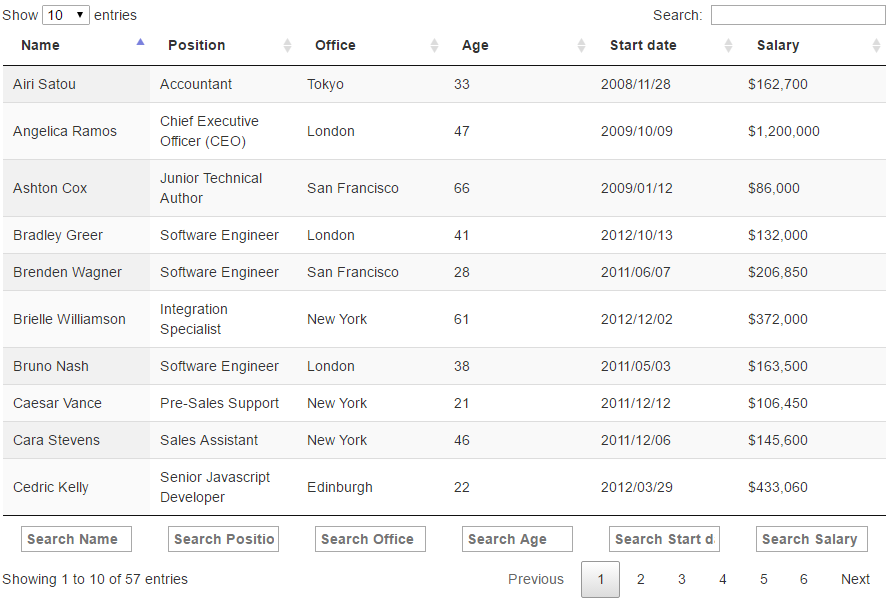
Par exemple, le code HTML d’une barre de navigation (gauche) peut se simplifier par le code C# (droite) suivant :



Source : [fluentBootstrap.com](http://fluentbootstrap.com/)

### Tableaux

Les tableaux listant les machines, les entreprise et les ouvrages seront réalisés avec la librairie « DataTables ». Cette dernière permet d’obtenir des tableaux avec des tris, des champs de recherche, une pagination automatique,… Voici un exemple sur lequel le tableau des machines sera basé :



Source : [datables.net](https://datatables.net/examples/api/multi_filter.html)

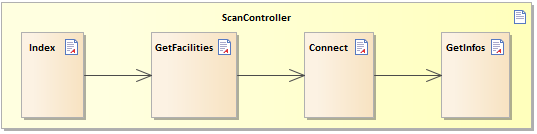
### Outil de backups

Afin de garder une trace des versions du projet, un dépôt Github a été mis en place. Ainsi, tout « bug » survenant pourra être debuggé en comparant avec une version antérieure du projet.

## Tests technologiques

### Récupération des infos StreamScan

Ce test technologique permet d’obtenir les infos depuis le serveur StreamScan et les afficher, le tout en Ajax.



Index

La page Index est appelée en premier. On y affiche la combobox des entreprises précédemment récupérées.

List<Enterprise> enterprises = dal.GetEnterprises();

Dictionary<string, string> cmxEnterprises = new Dictionary<string, string>();

cmxEnterprises.Add("Select an enterprise...", "-1");

foreach (Enterprise enterprise in enterprises)

{

cmxEnterprises.Add(enterprise.Name, "" + enterprise.Id);

}

return View(cmxEnterprises)

Si la valeur de la combobox change, on appelle l’action GetFacilities en se servant de l’event Jquery suivant :

$('#enterprisesSelect').on('change', function () {

$("#enterprisesForm").submit();

})



GetFacilities

L’action GetFacilities récupère les ouvrages de l’entreprise sélectionnée, les ajoute à la combobox et affiche le formulaire de connexion à la machine. Etant fait en Ajax, la validation classique ne fonctionne pas. On contrôle donc manuellement et on lance une exception si les infos ne sont pas valides.

if (enterprise == -1)

throw new Exception("Please select an enterprise");

List<Facility> facilities = dal.GetFacilities(enterprise);

Dictionary<string, string> cmxFacilities = new Dictionary<string, string>();

cmxFacilities.Add("Select a facility...", "-1");

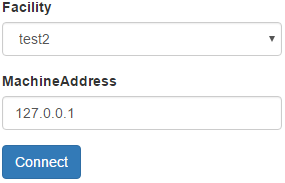
foreach (Facility facility in facilities)

{

cmxFacilities.Add(facility.Name, "" + facility.Id);

}

return PartialView("ScanConnect", new ScanViewModel { Facilities = cmxFacilities });



Connect

En cliquant sur le bouton « Connect », on appelle l’action Connect.

if (facility == -1)

throw new Exception("Please select a facility");

if (MachineAddress == "")

throw new Exception("Please enter an address");

Session["facilityId"] = facility;

bool ok = ClientWCF.CheckStatus(String.Format("http://{0}:{1}/{2}", MachineAddress, ClientWCF.defaultPort, ClientWCF.defaultServiceName));

if (!ok)

throw new Exception("No server has been found to this address");

ClientWCF wcf = new ClientWCF();

wcf.InitClient(MachineAddress);

Session["wcf"] = wcf;

return PartialView("ScanInfosButton");

Celle-ci teste que le serveur existe sur la machine spécifiée et s’y connecte si c’est le cas. Enfin, le bouton de scan est affiché.

L’ID de l’ouvrage et le ClientWCF sont stockés dans la session pour une utilisation lors de la récupération des infos.



Le bouton appelle l’action GetInfos qui envoie la requête au serveur. Les infos sont ensuite affichées.

ClientWCF wcf = (ClientWCF)Session["wcf"];

if (wcf == null)

throw new Exception("The connection has ended. Please reconnect to the server");

Object infos = wcf.SendMessage("GetInfos");

if (infos.GetType() == typeof(string))

return PartialView("ScanInfos", new InfosViewModel { Error = (string)infos });

int facility = (int)Session["facilityId"];

Session["infos"] = (Info)infos;

return PartialView("ScanInfos", new InfosViewModel { Infos = (Info)infos, Machines = dal.GetFacilityMachines(facility) });

On récupère le ClientWCF depuis la session. S’il est null, on affiche un message d’erreur. On envoie ensuite la requête au web service. Si un string est retourné, alors on l’affiche en tant que message d’erreur. Sinon, on récupère l’ID de l’ouvrage afin de pouvoir récupérer ses machines et on stocke les infos dans la session afin de les récupérer lors de la mise à jour ou de l’insertion. Enfin, on affiche la vue.

#### Implémentation du Ajax

La partie Ajax est principalement identique sur chaque page. Prenons celle de la page Connect :

@using (Ajax.BeginForm("Connect", null, new AjaxOptions

{

HttpMethod = "GET",

InsertionMode = InsertionMode.Replace,

UpdateTargetId = "infosButton",

OnFailure = "ConError",

OnSuccess = "ConSuccess",

LoadingElementId = "loading"

}, new { @id = "connectForm" }))

{

using (var form = @Html.Bootstrap().Form().Begin())

{

@form.Select("facility", "Facility", Model.Facilities)

@form.EditorFor(m => m.MachineAddress)

@form.Submit("Connect").EnsureFormGroup(false)

<img id="loading" src="~/Content/Images/ajax-loader.gif"

style="display:none;" />

@form.ValidationSummary().EnsureFormGroup(false)

}

}

<div id="infosButton"></div>

<script type="text/javascript">

function ConError(data) {

var html = data.responseText;

var div = document.createElement('div');

div.innerHTML = html;

div = div.firstChild;

document.querySelector("#connectForm .text-danger").textContent =

div.nextSibling.textContent;

document.querySelector("#infosButton").textContent = "";

}

function ConSuccess() {

document.querySelector("#connectForm .text-danger").textContent = "";

}

</script>

La première partie contient les options d’Ajax. On y spécifie :

HttpMethod Le type de requête (GET/POST)

InsertionMode Le mode d’insertion (remplacement, insertion au début, insertion à la fin)

UpdateTargetId L’ID de l’élément qu’on met à jour par la requête

OnFailure La méthode JS appelée lors d’un échec

OnSuccess La méthode JS appelée lors d’un succès

LoadingElementId L’ID de l’élément affiché pendant l’exécution de la requête (un GIF de chargement)

La deuxième partie concerne le formulaire de connexion, créé avec la librairie FluentBootstrap.

La troisième est le DIV qui contiendra le résultat de la requête.

La dernière contient les méthodes JS utilisées.

La méthode d’erreur récupère le message de l’exception et l‘affiche dans en rouge. Il vide le DIV de résultat également.

La méthode de succès retire juste le message d’erreur de son conteneur.

## Librairie utilisées

### Librairies StreamScan

Les librairies suivantes ont été développées :

* StreamScanCommon Contient les classes communes aux applications serveur et clientes
* StreamScanConnect Permet la connexion au serveur et l’obtention des infos
* QueriesManager Permet l’exécution de requêtes MySQL de manière simplifiée

### Librairies ASP.NET

* FluentBootstrap Génère une structure HTML avec Bootstrap à partir de code C#

(cf. : Concept d’implémentation - Design)

### Librairies JS

* DataTables Permet de créer des tableaux avec tris, champs de recherche, pagination

(cf. : Concept d’implémentation - Tableaux)

* Jquery Simplifie l’utilisation de JavaScript
* Jquery.validate Ajoute un contrôle des formulaires côté client
* Jquery.unobtrustive.ajax Permet l’utilisation de requête Ajax en C#

### Librairies CSS

* Bootstrap Permet la gestion du « responsive » et propose un design attractif

## Utilisation de la librairie QueriesManager

La librairie « QueriesManager » utilise le connecteur MySQL afin de communiquer avec une base de données de ce type. Elle a été créée afin de minimiser le code et de le faciliter.

### SELECT

Prenons l’exemple où l’on souhaite récupérer la liste des entreprises. Le code utilisé est le suivant :

MySqlReturn sqlR = null;

sqlR = db.ExecuteQuery(CEnterprises.GET\_ENTERPRISES);

if (sqlR.ErrorMessage != "")

throw new Exception(sqlR.ErrorMessage);

List<Enterprise> enterprises = new List<Enterprise>();

//On parcours les lignes retournées et on construit une liste de Enterprise

foreach (List<string> line in sqlR.Data)

{

enterprises.Add(new Enterprise

{

Id = Int32.Parse(line[0]),

Name = line[1],

Address = line[2],

Npa = Int32.Parse(line[3]),

City = line[4]

});

}

return enterprises;

Ici, on donne la requête SQL (contenue dans une constante) à la méthode ExecuteQuery qui l’exécute. Celle-ci retourne un objet MySqlReturn qui est composé :

* D’un booléen d’état (= requêtes INSERT/UPDATE/DELETE)
* D’une List contenant le résultat de la requête (= seulement utilisé en SELECT)
* D’un string d’erreur si une erreur est survenue

On vérifie ensuite que la requête se soit bien passée, sinon on lance une exception.

On parcourt alors la List retournée afin de formaté le tout sous une List d’Enterprise.

Remarque : Si la clause WHERE est utilisée, on se sert d’un Dictionary des paramètres selon l’utilisation de la requête INSERT (point suivant)

### INSERT (UPDATE)

Une requête INSERT (ou UPDATE) fonctionne de la même manière que le SELECT sauf qu’on se sert de la surcharge de la méthode ExecuteQuery qui prend un Dictionary<string, Object>. Par exemple :

Dictionary<string, Object> parameters = new Dictionary<string, Object>;

parameters.Add("@username", "admin");

parameters.Add("@pwd", "admin123");

string sql = "INSERT INTO T\_User (username, password) VALUES (@username, @pwd)";

sqlR = db.ExecuteQuery(sql, parameters);

if (sqlR.ErrorMessage != "")

throw new Exception(sqlR.ErrorMessage);

return sqlR.IsOk;

Contrairement à la requête INSERT, on contrôle l’exécution de la requête avec le booléen IsOk.

### DELETE

La requête DELETE suit le même fonctionnement que les précédentes. Cependant, notez que les requêtes DELETE sans clause WHERE sont bloquées par défaut. Pour l’activer, il faut changer le booléen En\_DeleteAll à true :

SqlManager.En\_DeleteAll = true;

# Implémentation

## DAL

La partie DAL (Data Access Layer) permet l’accès à la base de données.

### Requêtes SQL

Afin d’améliorer la lisibilité du code, les requêtes SQL sont stockées dans des classes de constantes :

public const string INSERT\_MACHINE = "INSERT INTO T\_System (FK\_Facility) " +

"VALUES (@facility)";

L’appel est donc plus clair :



### Récupération des machines StreamX d’un ouvrage

public Dictionary<int, Info> GetFacilityMachines(int facility)

{

Dictionary<string, Object> parameters = new Dictionary<string, Object>();

parameters.Add("@facility", facility);

MySqlReturn sqlR = db.ExecuteQuery(CMachines.GET\_FACILITY\_MACHINES, parameters);

if (sqlR.ErrorMessage != "")

throw new Exception(sqlR.ErrorMessage);

if (!sqlR.IsOk)

return new Dictionary<int, Info>();

//Dictionary ayant pour clé la PK\_System et comme valeur un objet Info

//représentant les propriétés de la machine

Dictionary<int, Info> machines = new Dictionary<int, Info>();

//On parcours les infos retournées afin de construire une liste d'Info

foreach (List<string> line in sqlR.Data)

{

Info machine = new Info();

machine.InfosMachine = new InfosMachine();

machine.InfosReseau = new InfosReseau();

machine.InfosStreamX = new InfosStx();

int pk\_system;

int sys\_property;

//On véfifie que les PK soient des INT

if (!Int32.TryParse(line[0], out pk\_system))

throw new Exception($"A database property type is not correct. " +

$"Attempted type : INT32 but got {line[0].GetType()}(Value:{line[0]}). " +

$"Please contact the administrator.");

if (!Int32.TryParse(line[1], out sys\_property))

throw new Exception($"A database property type is not correct. " +

$"Attempted type : INT32 but got {line[1].GetType()}(Value:{line[1]}). " +

$"Please contact the administrator.");

//Si le dictionary contient déjà la machine, alors on la récupère

if (machines.ContainsKey(pk\_system))

{

machine = machines[pk\_system];

}

//On applique la propriété à l'objet Info

machine = HMachines.SetMachineProperty(machine, sys\_property, line[2]);

machines[pk\_system] = machine;

}

return machines;

}

En premier lieu, la requête récupérant les machines de l’ouvrage est lancée. Si tout est OK, on construit un Dictionary composé de la clé de la machine et d’un objet Info (représentant les propriétés d’une machine).

On parcourt ensuite les données retournées afin de construire un objet Info. Si la machine est déjà présente dans la structure, alors on la récupère. Notez le contrôle effectué quant à la conversion string en int pour les clés.

Enfin, on utilise une méthode SetMachineProperty d’une classe Helper afin d’ajouter à l’objet Info donné, la propriété donnée et on met à jour la machine dans le Dictionary. La méthode SetMachineProperty fonctionne ainsi :

switch (idProperty)

{

case CSystemProperties.STREAMX\_VERSION:

machine.InfosStreamX.Version = value;

break;

case CSystemProperties.STREAMX\_FILENAME:

machine.InfosStreamX.Filename = value;

break;

...

}

return machine;

On assigne la valeur de la propriété à la machine donnée en fonction de l’ID de la propriété. Comme on peut le voir, les ID sont stockés sont forme de constante.

### Insertion d’une nouvelle machine StreamX

public MySqlReturn InsertMachine(int facility, Info machine)

{

//On démarre la transaction

db.ExecuteQuery("SET autocommit = 0");

//On verrouille en écriture la table "T\_System" afin de pouvoir

//récupérer plus tard la PK de l'entrée insérée

db.ExecuteQuery(CMachines.LOCK\_SYSTEM);

//On insert la machine dans la table "T\_System"

Dictionary<string, Object> parameters = new Dictionary<string, Object>();

parameters.Add("@facility", facility);

MySqlReturn sqlR = db.ExecuteQuery(CMachines.INSERT\_MACHINE, parameters);

if (!sqlR.IsOk)

{

//On annule la transaction

db.ExecuteQuery("ROLLBACK");

if (sqlR.ErrorMessage == "")

sqlR.ErrorMessage = "Can't insert the machine into the table \"T\_System\"";

return sqlR;

}

//On récupère la dernière PK insérée

sqlR = db.ExecuteQuery(CMachines.GET\_LAST\_INSERT\_ID);

if (!sqlR.IsOk)

{

//On annule la transaction

db.ExecuteQuery("ROLLBACK");

if (sqlR.ErrorMessage == "")

sqlR.ErrorMessage = "Can't get the last inserted ID";

return sqlR;

}

string systemId = sqlR.Data[0][0];

//On insert chaque propriété de la machine dans la table "T\_System\_Property"

Dictionary<int, Object> properties = HMachines.GetMachineProperties(machine);

foreach (int key in properties.Keys)

{

string property = "" + properties[key];

parameters = new Dictionary<string, object>();

parameters.Add("@systemId", systemId);

parameters.Add("@propertyId", key);

parameters.Add("@value", property);

sqlR = db.ExecuteQuery(CMachines.INSERT\_MACHINE\_PROPERTIES, parameters);

if (!sqlR.IsOk)

{

db.ExecuteQuery("ROLLBACK");

if (sqlR.ErrorMessage == "")

sqlR.ErrorMessage = "An error occured during adding the properties";

return sqlR;

}

}

db.ExecuteQuery("COMMIT");

return new MySqlReturn { IsOk = true };

}

Comme on effectue une série de requête, une transaction est utilisée afin de garantir l’exécution de l’entièreté des requêtes. Voici le déroulement des requêtes :

1. Démarrage de la transaction
2. Verrouillage en écriture de la table « T\_System », afin de pouvoir récupérer la bonne clé plus tard
3. Insertion de la machine dans la table « T\_System »
4. Insertion de chaque propriété dans la table « T\_System\_Property »
5. Si aucune tout s’est bien passé durant la transaction, on la « commit »

A noter que les tables sont déverrouillées automatiquement si un « rollback » ou un « commit » est appelé.

Logiquement, la table « T\_System » pourrait être déverrouillé après la récupération de la dernière clé insérée. Le problème est que la requête « UNLOCK TABLES » effectue un « commit » implicite (cf. : [dev.mysql.com](https://dev.mysql.com/doc/refman/5.7/en/lock-tables-and-transactions.html)) et que donc elle ne peut pas être appelée durant la transaction.

# Tests

### Login

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire de connexion | Login avec identifiants incorrects | Message d’erreur | **OK** |
| Formulaire de connexion | Login avec le nom d’utilisateur/mot de passe vide | Message d’erreur en dessous de l’élément vide | **OK** |
| Formulaire de connexion | Login avec des identifiants corrects | Message de succès + redirection | **OK** |

### Scan

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Sélection de l’entreprise | Sélection de l’entreprise | Affichage du formulaire de connexion à la machine | **OK** |
| Sélection de l’entreprise | Sélection d’aucune entreprise | Affichage erreur | **OK** |
| Sélection de l’ouvrage | Sélection de l’ouvrage | Aucune erreur | **OK** |
| Sélection de l’ouvrage | Sélection d’aucun ouvrage | Affichage erreur | **OK** |
| Formulaire de connexion à la machine | Connexion avec une adresse incorrecte | Message d’erreur en dessous du champ | **OK** |
| Formulaire de connexion à la machine | Connexion avec une adresse correcte | Affichage du bouton de scan | **OK** |
| Formulaire de scan | Récupération des infos si la connexion a été perdue | Message d’erreur | **OK** |
| Formulaire de scan | Récupération des infos sur un serveur existant | Affichage des infos avec le formulaire de mise à jour de la base de données | **OK** |
| Formulaire de mise à jour de la DB | Clic du bouton de mise à jour si l’élément n’a pas été modifié depuis | Met à jour l’élément sélectionné | **OK** |
| *Formulaire de mise à jour de la DB* | *Clic du bouton de mise à jour si l’élément a été modifié depuis (cf. : « Tests spéciaux – C »)* | *Affichage message d’erreur et rechargement de la liste* |  |
| *Formulaire d’ajout à la DB* | *Clic du bouton d’ajout si l’ouvrage n’existe plus (cf. : « Tests spéciaux – A »)* | *Affichage d’un message d’erreur et rechargement de la page* |  |
| *Formulaire d’ajout à la DB* | *Ajouts simultanés (cf. : « Tests spéciaux – E »)* | *Affichage au deuxième utilisateur* |  |
| Formulaire d’ajout à la DB | Clic du bouton d’ajout | Ajout de la machine dans la base de données | **OK** |

### StreamX Machines

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Sélection de l’entreprise | Sélection de l’entreprise | Affichage des ouvrages + liste toutes les machines de l’entreprise | **OK** |
| Sélection de l’ouvrage | Sélection de l’ouvrage | Affichage des machines de l’ouvrage | **OK** |
| Recherche | Recherche d’un élément | Affichage des machines correspondantes | **OK** |

### Enterprises

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Bouton « New enterprise » | Clic du bouton « New Enterprise » | Affichage du formulaire d’ajout | **OK** |
| Bouton « Update » | Clic du bouton « Update » | Met à jour la machine |  |
| Lien « Show facilities » | Clic du lien « Show facilities » | Affichage des ouvrages |  |
| Bouton « Delete » | Clic du bouton « Delete » | Affichage d’une popup de confirmation |  |
| Formulaire de confirmation | Clic « Cancel » | Disparition de la popup |  |
| *Formulaire de confirmation* | *Clic « Delete » si déjà supprimé (cf. : « Tests spéciaux – D »)* | *Affichage message d’erreur* |  |
| Formulaire de confirmation | Clic « Delete » | Suppression de l’entreprise (et de ses ouvrages + machines) |  |

### Facilities

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Bouton « New facility » | Clic du bouton « New facility » | Affichage du formulaire d’ajout |  |
| Bouton « Update » | Clic du bouton « Update » | Affichage du formulaire de mise à jour |  |
| Bouton « Delete » | Clic du bouton « Delete » | Affichage d’une popup de confirmation |  |
| Formulaire de confirmation | Clic « Cancel » | Disparition de la popup |  |
| *Formulaire de confirmation* | *Clic « Delete » si déjà supprimé (cf. : « Tests spéciaux – D »)* | *Affichage message d’erreur* |  |
| Formulaire de confirmation | Clic « Delete » | Suppression de l’ouvrage (et de ses machines) |  |

### New enterprise

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire d’ajout | Ajout avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire d’ajout | Ajout avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| Formulaire d’ajout | Ajout de l’entreprise | Message de succès + redirection vers la liste |  |

### New facility

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire d’ajout | Ajout avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire d’ajout | Ajout avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| *Formulaire d’ajout* | *Ajout dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Message d’erreur* |  |
| Formulaire d’ajout | Ajout de l’ouvrage | Message de succès + redirection vers la liste |  |

### Update enterprise

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire de mise à jour | Mise à jour avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| *Formulaire de mise à jour* | *Mise à jour dont les données d’origine diffèrent de celles de la DB (cf. : « Tests spéciaux – C »)* | *Message d’erreur* |  |
| Formulaire de mise à jour | Mise à jour de l’entreprise | Message de succès + redirection vers la liste |  |

### Update facility

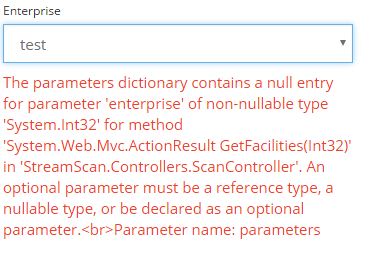
|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire de mise à jour | Mise à jour avec des champs vides | Message d’erreur en dessous de l’élément vide |  |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Message d’erreur en dessous du NPA |  |
| *Formulaire de mise à jour* | *Mise à jour dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Message d’erreur* |  |
| *Formulaire de mise à jour* | *Mise à jour dont les données d’origine diffèrent de celles de la DB (cf. : « Tests spéciaux – C »)* | *Message d’erreur* |  |
| Formulaire de mise à jour | Mise à jour de l’ouvrage | Message de succès + redirection vers la liste |  |

# Problèmes rencontrés

## Version de la librairie StreamScanCommon différente

La librairie « StreamScanCommon », utilisée pour stocker les classes utilisées par le client et le serveur, se doit d’être **exactement** la même entre le serveur et le client. En effet, l’objet Info y est stocké et il est utilisé lors de la récupération des infos (= sérialisation). L’exception suivante est obtenue :

System.ServiceModel.Dispatcher.DataContractSerializerOperationFormatter...



# Bibliographie: liste des sources et références

Conseil: Cette partie peut contenir une liste des sources et éléments littéraires utilisés.

# Glossaire trier par ordre alphabetique

|  |  |
| --- | --- |
| Terme | Signification |
| StreamX | Passerelle de communication développée par Infoteam |
|  |  |
|  |  |

# Signatures

Je soussigné déclare que les informations contenues dans ce rapport de travail pratique individuel rendu ce jour le 13.06.17 dans le cadre de la procédure de qualification de mon CFC d’informaticien, ne sont pas plagiées. Toutes les informations de sources extérieures ainsi que les informations fournies par des tiers durant le déroulement du travail sont consignées.

L’apprenti/-e doit signer la documentation avant la remise afin de témoigner de la justesse des informations qui y figurent.

|  |  |  |
| --- | --- | --- |
| Date | Nom | Signature |
|  |  |  |

# Annexes

Code imprimé, protocole de séances, etc.: spécifier complètement et annexer à la documentation.

Listings de scripts et programmes. Le travail fourni par la candidate ou le candidat doit être entièrement documenté. Omettre le code généré automatiquement s’il n’est pas nécessaire à la compréhension.

Dans le cas où des manuels ont été créés, vous pouvez les ajouter ici en tant qu’annexe.

## Première visite des experts

Les points suivants ont étés notés suites à la première visite des experts :

* Garder une trace dans le journal d’un changement, remarque d’un expert
* Le journal ne doit pas dépasser 120 heures (8h/jour)
* Bien noter s’il y a un changement dans le cahier des charges
* Journal : Précision à l’heure (pas plus précis)

## Deuxième visite des experts

Les points suivants ont été retenus :

Rapport de TPI

* Ne pas oublier le code en annexes
* Orthographe
* Lisibilité du texte et des graphiques

Présentation

* Intonation, vitesse, contacts visuels, gestes
* Maximum 20 minutes
* Imprimer le websummary, format A4