******

**TPI 2017**

*Travail professionnel individuel*

*Informaticien CFC*

StreamScan

Rapport de projet

Table des matières

Résumé du rapport du TPI 3

1 Introduction 4

1.1 StreamX 4

1.2 StreamScan 4

1.3 Analyse de la situation initiale 5

1.4 Analyse de l’état désiré 5

1.5 Cahier des charges 5

2 Personnes de contact 6

3 Organisation du projet 6

3.1 Réalisation des diagrammes 6

3.2 Réalisation des maquettes 6

3.3 Outil de backup 6

4 Analyse 7

4.1 Objectifs 7

4.2 Rentabilité 7

4.3 Fonctionnalités 7

4.4 Planification des étapes du développement 7

4.5 Diagramme d’utilisation 8

4.6 Diagrammes d’activité 9

4.7 Schéma global de base de données 16

4.8 Modèle de Conception de Données (MCD) 17

5 Conception 18

5.1 Architecture du système 18

5.2 Modèle Logique de Données (MLD) 19

5.3 Maquettes 20

5.4 Diagramme de classes 24

5.5 Planification des tests 25

5.6 Tests spéciaux 30

5.7 Concept d‘implémentation 31

5.8 Tests technologiques 33

5.9 Librairie utilisées 36

5.10 Utilisation de la librairie QueriesManager 36

6 Implémentation 38

6.1 Diagramme de classes 38

6.2 DAL 39

6.3 Controllers 44

6.4 Vues (Views) 45

6.5 ViewModels 46

6.6 Pages d’erreur 47

6.7 Problèmes rencontrés 48

6.8 Tests 49

7 Conclusion 54

7.1 Conclusion personnelle 54

7.2 Etat du projet 54

7.3 Améliorations possibles 54

8 Remerciements 55

9 Bibliographie: liste des sources et références 55

10 Glossaire 56

11 Signatures 56

12 Annexes 56

# 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.

Situation initiale

Infoteam développe un produit, StreamX, qui est installé sur des machines qui sont vendues aux clients. Afin de connaître le parc de machines d’un client et d’agir en conséquence, un outil de scan est nécessaire.

StreamScan est une application permettant l’obtention et l’enregistrement de diverses données matérielles et logicielles de machines StreamX depuis une application serveur lancée sur la machine dont on souhaite extraire les informations. Les informations obtenues sont ensuite placées dans une base de données selon l’entreprise et l’ouvrage. Ces derniers sont spécifiés par l’utilisateur avant la récupération des informations d’une machine.

Mise en œuvre

En premier lieu, le cahier des charges a été analysé. De ce dernier, une planification détaillée a été établie en trois étapes : analyse, conception et implémentation.

La phase d’analyse a permis de définir les fonctionnalités et le déroulement de celle-ci via la réalisation de diagrammes (utilisation et activité). Le diagramme MCD a également été modélisé afin de décider de la structure de la base de données. Cette phase permet de construire des bases solides sur lesquelles on se base ensuite pour réaliser l’application.

La conception va dans la continuité de l’analyse et la développe. Le fonctionnement global de l’application est montré avec le schéma de l’architecture, on y a définit la structure de l’application via un diagramme de classes et la présentation de l’application a été illustrée à l’aide de maquettes. De plus, les tests fonctionnels à effectuer ont été établis. Une attention particulière a été prise en ce qui concerne les tests dits « spéciaux » ou « multi-utilisateurs ». En effet, l’application étant sous la forme d’un site web, plusieurs utilisateurs peuvent y être connectés simultanément et donc des erreurs peuvent être obtenues en fonction de l’agissement de ceux-ci. Les technologies utilisées dans l’application sont expliquées et des tests technologiques (proof of concept) en rapport ont été réalisés.

Enfin, pour l’implémentation, le diagramme de classes a été généré afin de montrer les différences avec celui de la conception et de montrer graphiquement la structure finale de l’application. Les différents composants de l’application ont été expliqués par des exemples concrets et appropriés afin de comprendre le fonctionnement global de l’application et de cerner certaines subtilités de ASP.NET. Les tests fonctionnels ont été testés et les problèmes rencontrés développés. Un manuel de l’utilisateur expliquant les fonctionnalités de l’application a été rédigé.

Résultat

L’application finie comprend toutes les fonctionnalités spécifiées dans le cahier des charges. Chacune d’entre elles a été testée et les cas spéciaux traités. La récupération des données est complète mais des détails restent à ajouter. Par exemple, les protocoles de communications ne sont pas explicitement affichés mais correspondent aux dossiers de logs. On peut donc dire que c’est négligeable en l’état. Il a été pris note des améliorations possibles comme par exemple la suppression d’une machine StreamX ou encore la date de mise à jour d’une machine dans la base de données.

# Introduction

## StreamX[[1]](#footnote-1)

Le produit StreamX, développé par Infoteam, est une solution de conduite et de gestion de processus d’acquisition de données temps réels.

Il est installé sur plusieurs types de machines (MOXA, Beckhoff,…), sous différents OS (Windows XP, Windows 7, Linux, ISOS,…) et est configuré selon l’utilisation de client.

L’installateur de StreamX est mis à jour régulièrement et mis à disposition des clients sur la plate-forme d’échange d’Infoteam pour que ceux-ci puissent bénéficier de toutes les nouveautés disponibles. Les clients peuvent également ajouter des composants à l’application ou réajuster sa configuration.

Il est donc compliqué de garder un état à jour du parc des machines hébergeant StreamX. C’est là qu’intervient l’application « StreamScan ».

## StreamScan

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

L’application **StreamScan** est utilisée afin d’obtenir les informations des machines de manière centralisée 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

Les informations suivantes vont pouvoir être obtenues.

* 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 dossiers de Log créés

## Analyse de la situation initiale

En l’état actuel, aucun moyen ne permet de connaître ces informations de manière centralisée.

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 (adresse 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 va permettre d’obtenir les informations d’une machine StreamX grâce à son adresse IP/Hostname et à les stocker dans une base de données MySQL.

Celles-ci sont ensuite listées selon l’entreprise et l’ouvrage auxquels appartient la machine. Un ouvrage est un bâtiment équipé et en production comme un barrage, un poste électrique ou encore une centrale de chauffage à distance.

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 adresse IP
* Sauvegarde des informations récoltées dans une base de données MySQL selon l’ouvrage et l’entreprise auxquels appartient la machine
* 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.

# Personnes de contact

|  |  |
| --- | --- |
| Nom | Rôle |
| M. Nicolas Richoz | Premier expert |
| M. Vincent Gremaud | Deuxième expert |
| M. Gérald Schaller | Expert de validation |
| M. Pierre-Alain Mettraux | Formateur en entreprise |
| Mme. Laurence Dubied Bielmann | Supérieure professionnelle |

# Organisation du projet

## Réalisation des diagrammes

La majeure partie des diagrammes sont réalisés à l’aide du programme « Enterprise Architect ». Celui-ci est très complet et propose un design attractif.

De plus, il offre les fonctionnalités de « Forward Engineering » et « Reverse Engineering » qui sont très utiles et permettent de gagner du temps.

L’outil de la suite Microsoft Office, « Visio », sera également utilisé pour la réalisation de certains diagrammes.

## Réalisation des maquettes

Les maquettes sont réalisées avec le logiciel « Balsamiq Morckups 3 » qui propose une interface intuitive avec de nombreux composants graphiques.

## Outil de backup

Afin de garder une trace des versions du projet, un dépôt GitHub a été mis en place. Ainsi, si une « erreur fantôme » apparaît, l’application pourra être comparée avec une version antérieure afin de trouver l’origine de l’erreur grâce aux différences entre les deux versions.

Un dépôt est également créé pour la documentation. De plus elle est aussi placée sur un disque réseau à Infoteam.

Ceci permet également de faciliter le partage entre plusieurs PC.

# Analyse

## Objectifs

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

* Pouvoir obtenir les informations d’une machine StreamX
* Stocker ces informations dans la base de données
* Récupérer l’ensemble des informations des machines et les afficher à l’écran
* 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 fonctionnel avant la fin du stage.

## Fonctionnalités

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

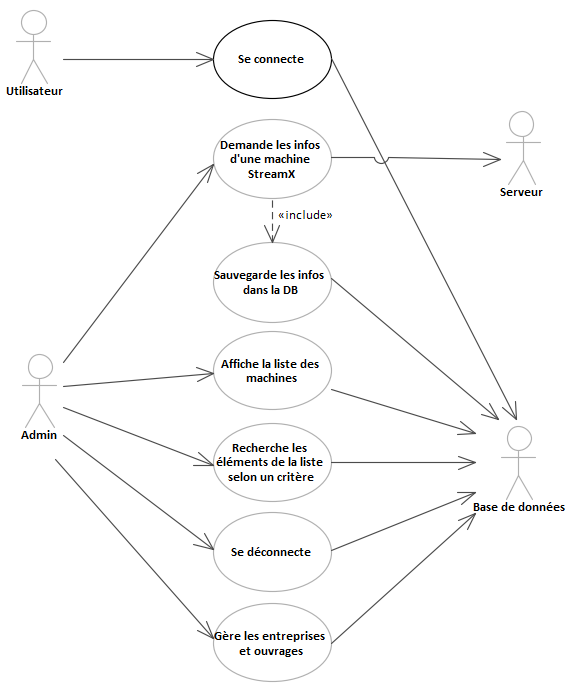
* Obtention des informations d’une machine StreamX (scan)
* Mettre à jour ou insérer les informations dans la base de données à partir d’un scan
* Voir la liste des machines StreamX à partir de la base de données en fonction de l’entreprise et de l’ouvrage
* Gérer les entreprises et les ouvrages

## 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 informations d’une machine StreamX (scan) et 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 StreamX à partir de la base de données
7. Tests fonctionnels de l’application

## Diagramme d’utilisation



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 informations 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

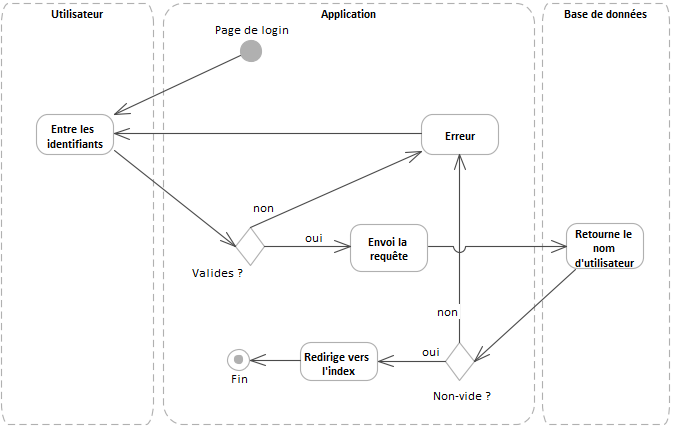
Déconnecte l’admin de l’application

Gère les entreprises et ouvrages

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

## Diagrammes d’activité

### Se connecte

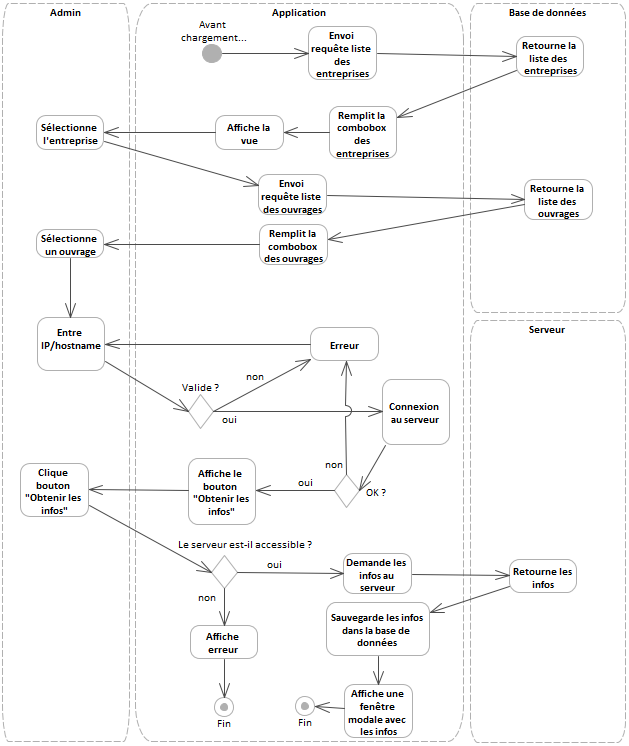


Suite au chargement de la page, l’utilisateur saisit ses identifiants.

S’ils sont invalides, on affiche une erreur. Dans le cas contraire, une requête à la base de données est effectuée 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 informations d’une machine StreamX



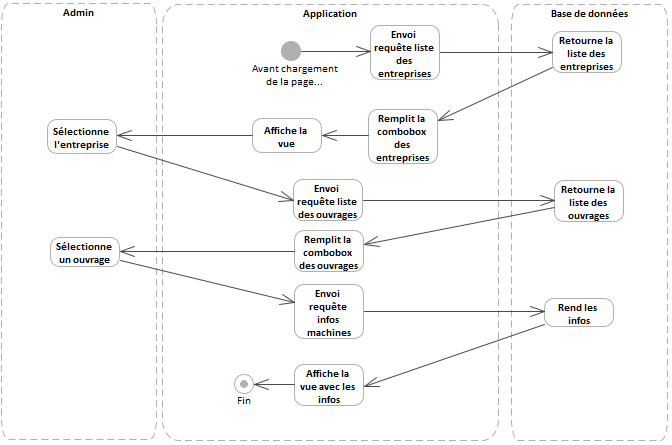
Premièrement, la liste des entreprises est obtenue à partir de la base de données. Elle est ensuite appliquée à la combobox des entreprises et on affiche la vue.

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 informations ».

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 informations sont ensuite sauvegardées dans la base de données puis on affiche une fenêtre modale avec les informations.

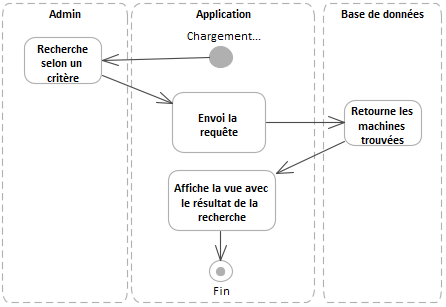
### Affiche la liste des machines



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



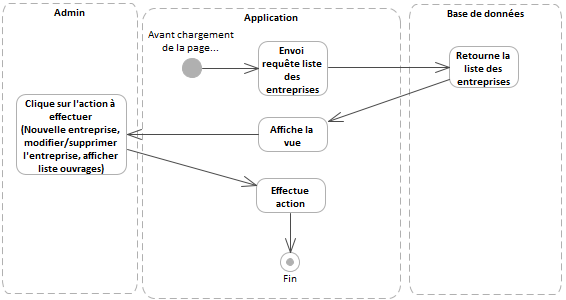
Suite au chargement de la liste des machines, le client peut rechercher un critère.

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

Il n’a pas été jugé nécessaire de réaliser ce diagramme.

### Gestion des entreprises et des ouvrages

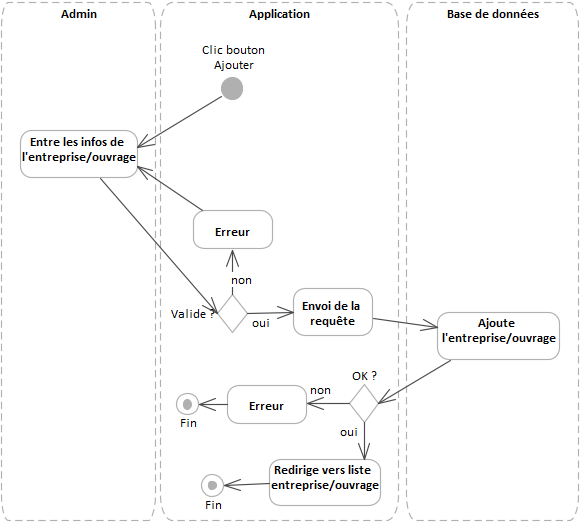


Premièrement, on récupère la liste des entreprises et on affiche la vue. Le client peut ensuite effectuer plusieurs actions :

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

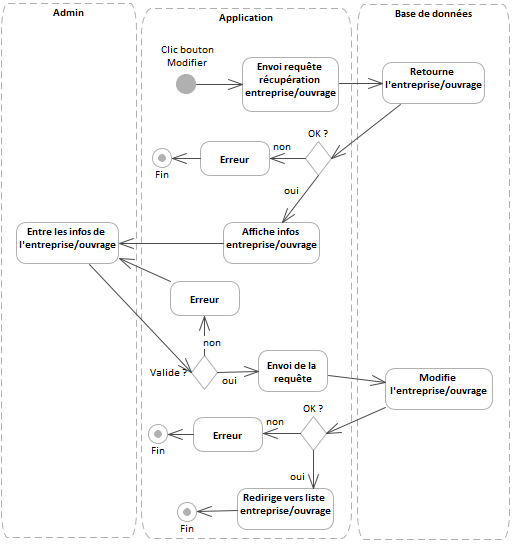
Note : La dernière action affiche une vue similaire à celle-ci, c’est pourquoi aucun diagramme n’y est consacré.

#### Ajoute une entreprise/ouvrage



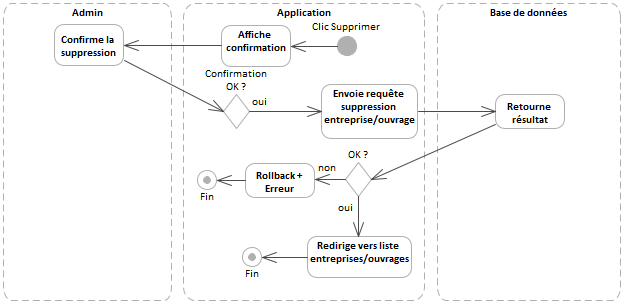
Lorsque l’admin appuie sur le bouton «  Ajouter », celui-ci doit entrer les informations de l’entreprise/ouvrage et valider. Si les données entrées sont valides, alors la requête est envoyée à la base de données qui ajoute l’entreprise/ouvrage. Si tout c’est bien passé, l’admin est redirigé vers la liste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

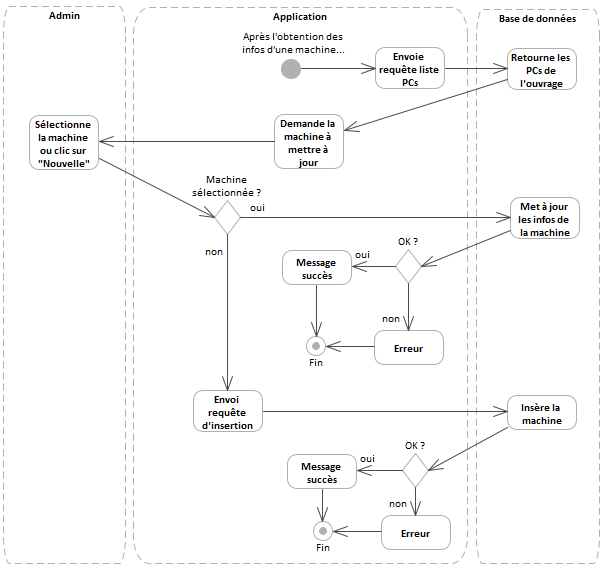


Lorsque l’admin appuie sur le bouton « Supprimer », une fenêtre de confirmation est affichée.

Si l’admin appuie sur « OK », on envoie la requête de suppression au serveur.

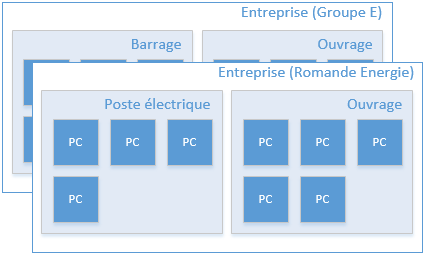
Si elle s’est correctement déroulée, on affiche la liste des entreprises/ouvrages. Sinon on fait un « rollback » de la requête et on affiche une erreur.

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



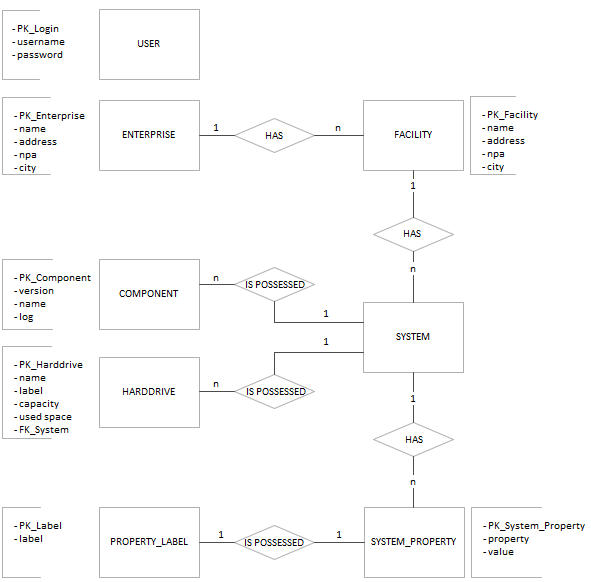
Après l’obtention des informations d’une machine, on récupère tous les PCs de l’ouvrage. Puis, on demande au client quelle machine il veut mettre à jour ou s’il souhaite en ajouter une nouvelle. Dans le premier cas on envoie une requête « INSERT » à la base de données et dans le deuxième cas, une requête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

## Schéma global de base de données

La base de données est composée de plusieurs entreprises.

Celles-ci contiennent elles-mêmes des ouvrages, comme des barrages ou des postes électriques. 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 disques durs (Hard drive) mais un disque dur n’est possédé que par un seul PC.

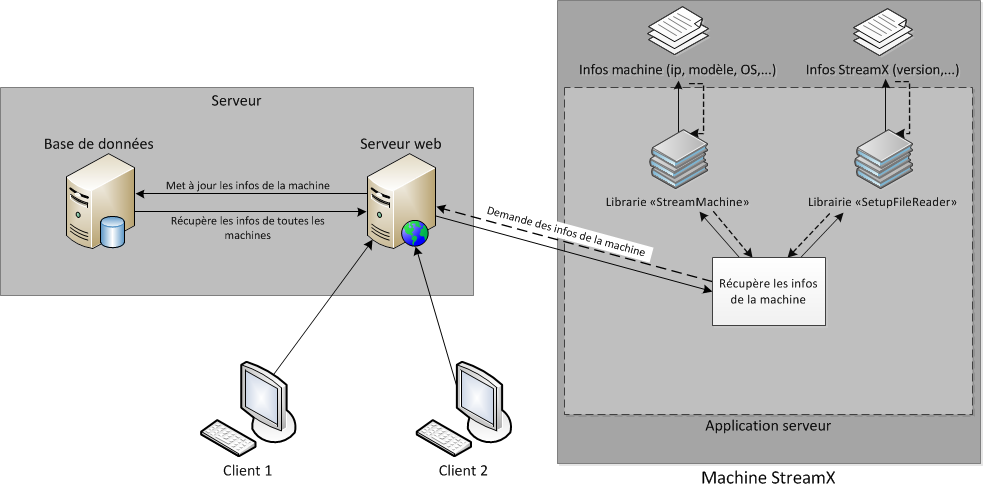
Un PC a plusieurs propriétés (System\_Property) mais une propriété ne caractérise qu’un seul PC.

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



Architecture complète de l’application. A noter que le TPI ne concerne que la partie cliente (gauche)

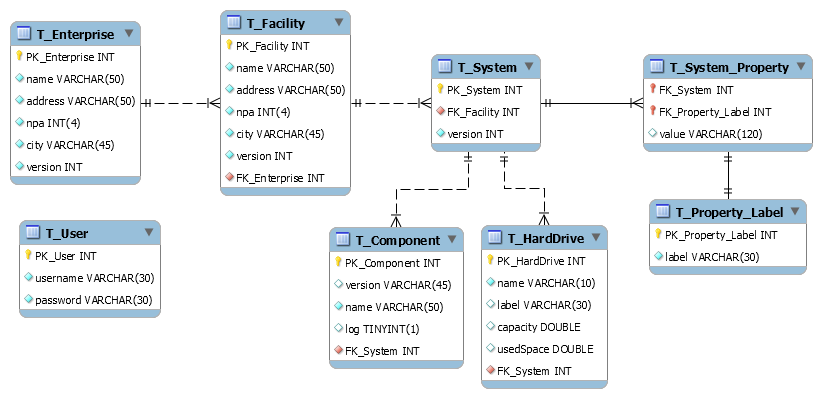
L’application cliente est composée d’un serveur web ainsi que d’une base de données.

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. Par exemple :

|  |  |  |
| --- | --- | --- |
| FK\_System | Label | Value |
| 1 | StreamX\_Version | 6.1 |
| 1 | StreamX\_FileName | StreamX RT 6\_2 2017\_04\_26\_nodoc.exe |

A noter que l’exemple est simplifié pour qu’il soit plus parlant. En effet, la colonne « Label » est en réalité composée d’entiers (FK) correspondants chacun à une clé de la table « T\_Property\_Label ».

### 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)

HardDrive

Name Longueur adéquate pour un nom de disque dur : sur Windows, le nom n’est composés que de 3 caractères (ex. : « C:\ »)

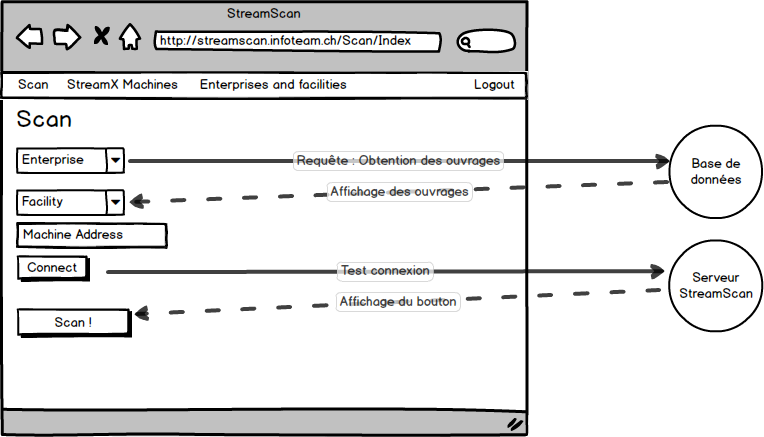
Label Longueur adéquate pour une description de disque dur

Property Label

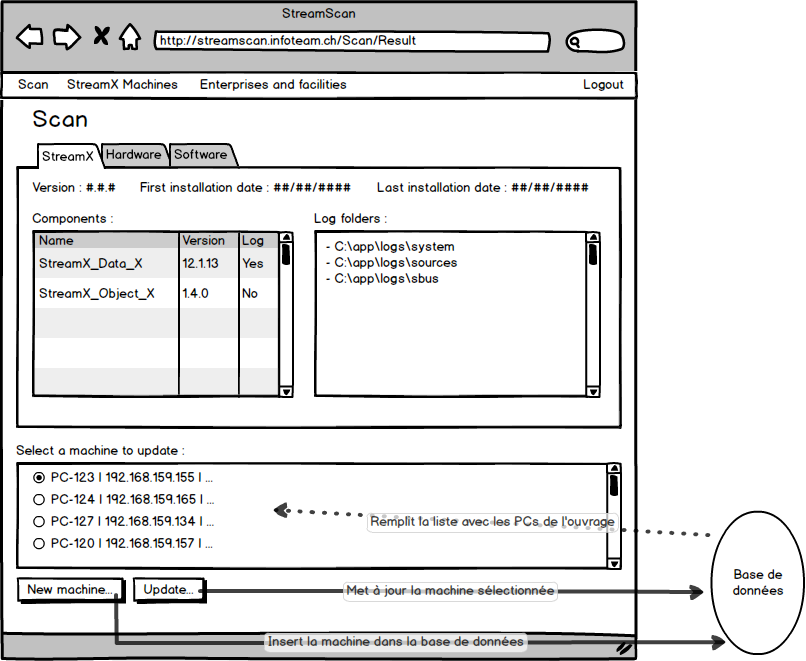
Label Longueur adéquate pour un label

## Maquettes

### Scan

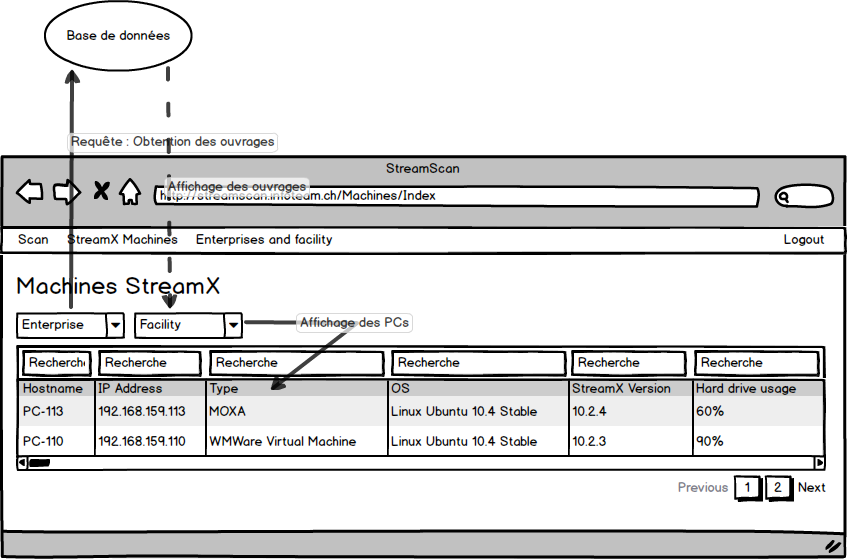


La combobox de l’entreprise, une fois sélectionnée, fait apparaître le formulaire de connexion à la machine. On sélectionne l’ouvrage et on saisit l’adresse de la machine. Le bouton « Connect » connecte l’utilisateur à la machine serveur puis affiche le bouton de scan. Tout ce procédé est fait en Ajax.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



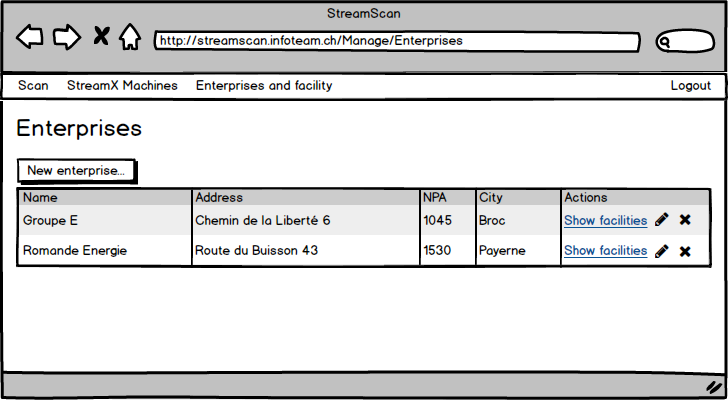
La partie du haut permet de visualiser le résultat de la requête. Celle du bas permet de mettre à jour les informations d’une machine dans la base de données ou d’en insérer une nouvelle.

### StreamX Machines

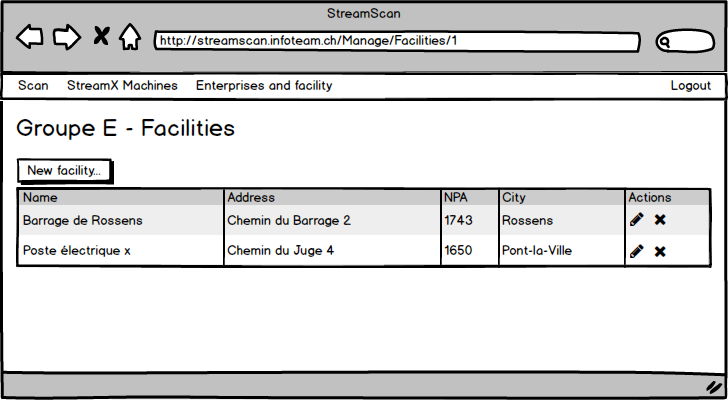


En sélectionnant l’entreprise, on fait apparaître la liste de toutes les machines de celle-ci. Ensuite, en sélectionnant l’ouvrage, on restreint le tableau aux machines de l’ouvrage seulement.

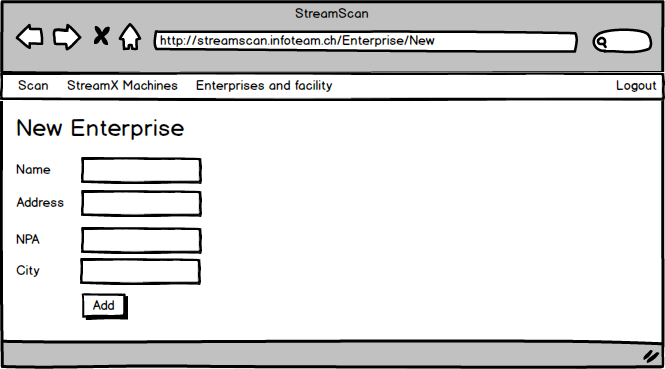
### Enterprises



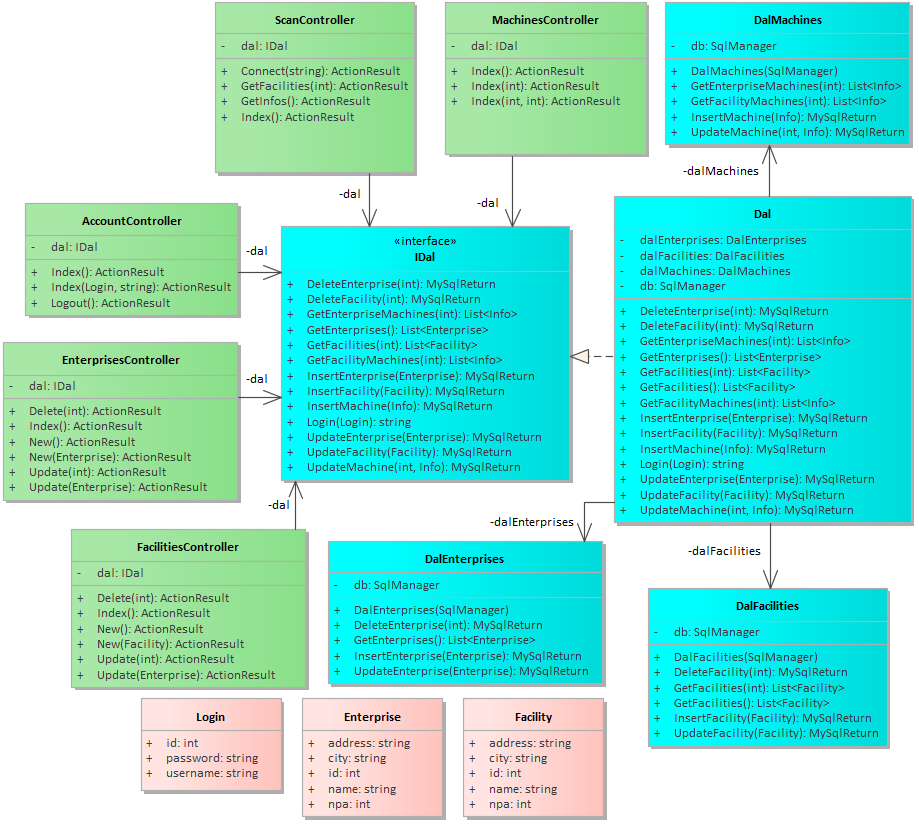
### Facilities



### New Enterprise/Facility



## Diagramme de classes



L’application est réalisée selon le modèle MVC.

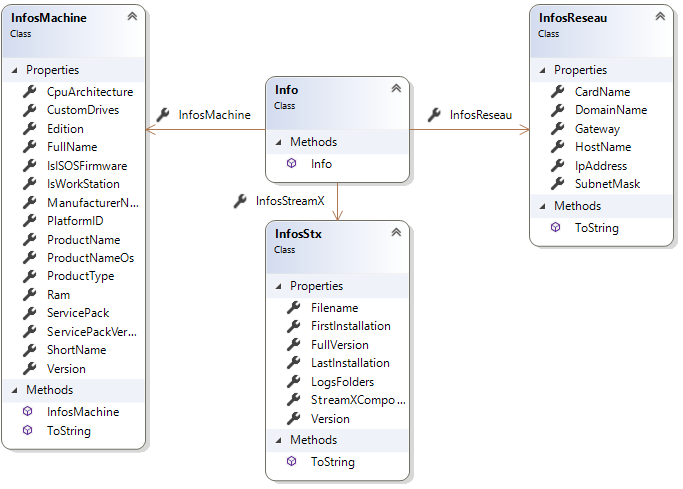
Les « Controller » (vert) appellent les vues HTML et font la liaison avec le modèle. On appelle les méthodes du Controller des actions. Chaque action correspond généralement à une vue, sauf dans le cas d’une requête POST. Par exemple, dans le Controller AccountController la première méthode Index affiche la page de connexion et la seconde agit en POST afin de vérifier les identifiants dans la base de données.

Le modèle (bleu) permet l’accès à la base de données. La Dal (Data Access Layer) est divisée en sous-Dal afin de faciliter la recherche et la lisibilité des méthodes. De plus, cela permet aux Controllers de n’avoir qu’une seule référence vers la Dal.

Les classes roses (équivalent de JavaBean) font également parties du modèle. Elles permettent le stockage des données lors de leur récupération dans la base de données. Cela simplifie les envois aux Controllers.

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

La classe Info vient de la librairie StreamScanCommon (cf. : « Librairies utilisées ») permettant le stockage des informations 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 « Tests spéciaux ») sont mis en italique.

### Login

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire de connexion | Login avec identifiants incorrects | Affichage d’un message d’erreur |  |
| Formulaire de connexion | Login avec le nom d’utilisateur/mot de passe vide | Affichage d’un message d’erreur en dessous de l’élément vide |  |
| Formulaire de connexion | Login avec des identifiants corrects | Affichage d’un message de succès, puis 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 d’un message d’erreur |  |
| Sélection de l’ouvrage | Sélection de l’ouvrage | Aucune erreur affichée |  |
| Sélection de l’ouvrage | Sélection d’aucun ouvrage | Affichage d’un message d’erreur |  |
| Formulaire de connexion à la machine | Connexion avec une adresse incorrecte | Affichage d’un 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 informations si la connexion a été perdue | Affichage d’un message d’erreur |  |
| Formulaire de scan | Récupération des informations sur un serveur existant | Affichage des informations 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 | Mise à jour de 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 d’un 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 | 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 |  |
| Recherche | Recherche d’un élément | Affichage des machines correspondantes |  |

### 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 d’un 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 | Affichage d’un message d’erreur en dessous de l’élément vide |  |
| Formulaire d’ajout | Ajout avec un NPA invalide | Affichage d’un message d’erreur en dessous du NPA |  |
| Formulaire d’ajout | Ajout de l’entreprise | Affiche d’un message de succès + redirection vers la liste |  |

### New facility

|  |  |  |  |
| --- | --- | --- | --- |
| Élément testé | Description du test | Attente | Résultat |
| Formulaire d’ajout | Ajout avec des champs vides | Affichage d’un message d’erreur en dessous de l’élément vide |  |
| Formulaire d’ajout | Ajout avec un NPA invalide | Affichage d’un message d’erreur en dessous du NPA |  |
| *Formulaire d’ajout* | *Ajout dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Affichage d’un message d’erreur* |  |
| Formulaire d’ajout | Ajout de l’ouvrage | Affichage d’un 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 | Affichage d’un message d’erreur en dessous de l’élément vide |  |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Affichage d’un message d’erreur en dessous du NPA |  |
| *Formulaire de mise à jour* | *Mise à jour d’une entreprise inexistante (cf. : « Tests spéciaux – B »)* | *Affichage d’un 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 »)* | *Affichage d’un message d’erreur* |  |
| Formulaire de mise à jour | Mise à jour de l’entreprise | Affichage d’un 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 | Affichage d’un message d’erreur en dessous de l’élément vide |  |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Affichage d’un message d’erreur en dessous du NPA |  |
| *Formulaire de mise à jour* | *Mise à jour dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Affichage d’un message d’erreur* |  |
| *Formulaire de mise à jour* | *Mise à jour d’un ouvrage inexistant (cf. : « Tests spéciaux – B »)* | *Affichage d’un 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 »)* | *Affichage d’un message d’erreur* |  |
| Formulaire de mise à jour | Mise à jour de l’ouvrage | Affichage d’un 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

Un utilisateur affiche le formulaire de mise à jour. Pendant ce temps, un second utilisateur supprime l’élément en question. Lorsque le premier utilisateur cliquera sur le bouton de mise à jour, l’application tentera de mettre à jour un élément inexistant. Le cas doit être traité et donc vérifier d’un message d’erreur est affiché.

#### Perte de mise à jour[[2]](#footnote-2)

1

Ici, deux mises à jour se font au même moment. Les données mises à jour par l’utilisateur 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 également la propriété « version » qui est incrémentée à chaque mise à jour d’un élément. Si celle-ci est différente que lorsqu’on a récupéré l’élément, ce dernier ne sera pas mis à jour.

[UPDATE](http://cpanel.emf-informatique.ch/cpsess1812177458/3rdparty/phpMyAdmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/update.html) T\_Enterprise [SET](http://cpanel.emf-informatique.ch/cpsess1812177458/3rdparty/phpMyAdmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/set.html) name = "" WHERE FK\_Enterprise = "1" AND version = "1"

#### 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.

Le test survient dans les cas suivant :

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

## Concept d‘implémentation

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

### Connexion à l’application serveur

L’application serveur met à disposition les informations d’une machine via un web service. Pour s’y connecter, l’application console de test, réalisée avant ce TPI, a été reprise et modifié sous la forme d’une librairie : **StreamScanConnect**. On peut vérifier si l’application serveur est disponible, s’y connecter et utiliser la méthode « SendMessage » afin qu’elle envoie la requête de récupération des informations.

On vérifie la connexion en envoyant une HttpWebRequest :

bool status = false;

try

{

var myRequest = (HttpWebRequest)WebRequest.Create(url);

var response = (HttpWebResponse)myRequest.GetResponse();

status = true;

}

catch (Exception ex)

{

status = false;

}

On définit ensuite la liaison et l’adresse du serveur :

BasicHttpBinding myBinding = new BasicHttpBinding();

EndpointAddress myEndpoint =

new EndpointAddress($"http://{ip}:{port}/{serviceName}");

ChannelFactory<IStreamScanInfos> myChannelFactory = new

ChannelFactory<IStreamScanInfos>(myBinding, myEndpoint);

client = myChannelFactory.CreateChannel();

On utilise enfin la méthode « SendMessage("GetInfos") » pour envoyer la commande :

client.GetInfos();

### 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.

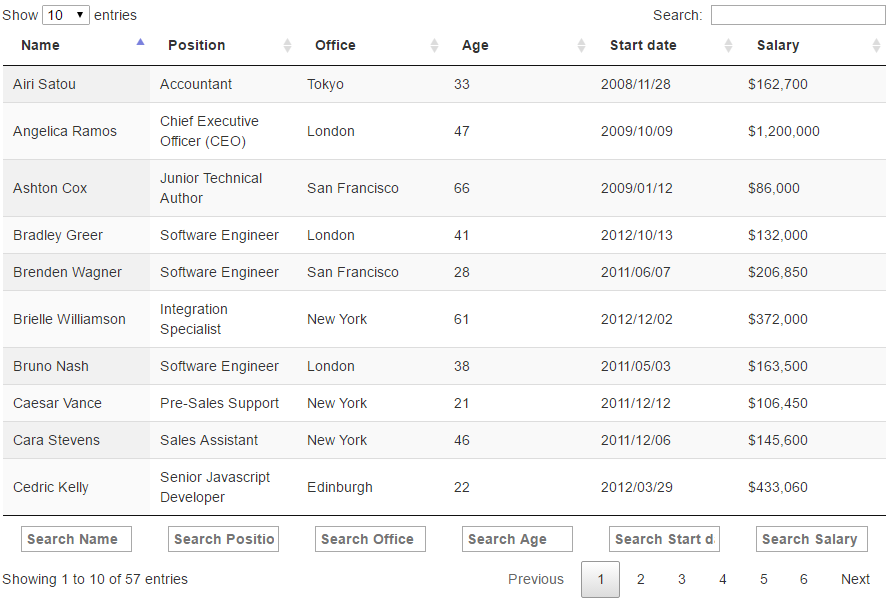
Par exemple, le code HTML d’une barre de navigation (gauche) peut se simplifier par le code C# (droite) suivant :



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

### Tableaux

Les tableaux listant les machines, les entreprise et les ouvrages sont 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 :

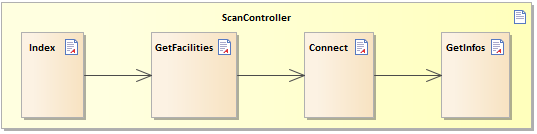


Source : [datables.net](https://datatables.net/examples/api/multi_filter.html)

## Tests technologiques

### Récupération des informations StreamScan

Ce test technologique permet d’obtenir les informations 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);

}

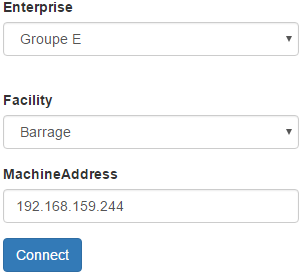
return View(cmxEnterprises)

Si la valeur de la combobox change, on appelle l’action GetFacilities en se servant de l’évènement 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 informations 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");

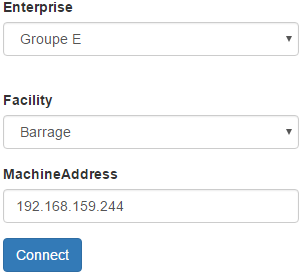
foreach (Facility facility in facilities)

{

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 informations 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 informations 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 en rouge. Il vide également le conteneur où est inséré le résultat en temps normal.

La méthode de succès retire le message d’erreur précédent (si une erreur est survenue) de son conteneur.

## Librairie utilisées

### Librairies StreamScan

Les librairies suivantes ont été développées :

* StreamScanCommon Contient les classes communes aux applications serveur et

cliente

* 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 Crée 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#
* jquery.mCustomScrollbar Offre des barres de scroll customisées attractives

### 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 formater 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.

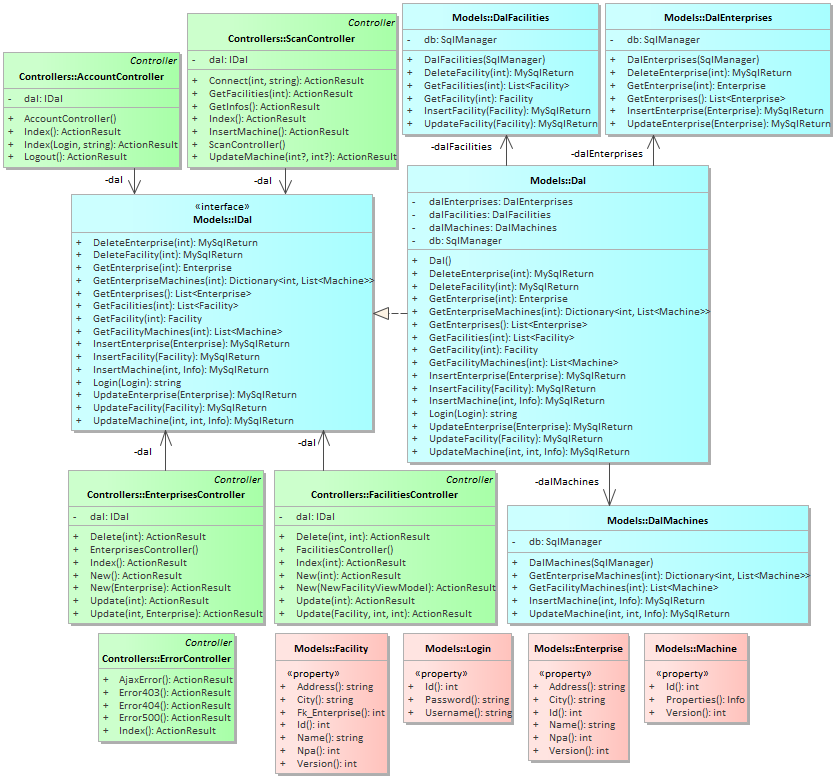
### DELETE

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;

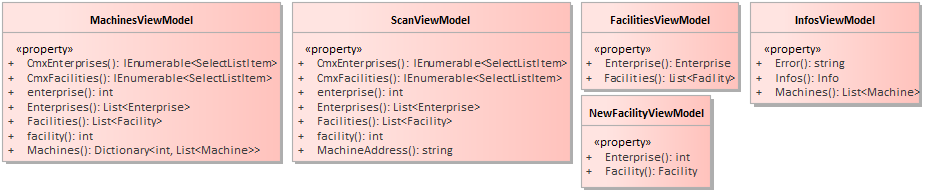
# Implémentation

## Diagramme de classes



Une première différence observable avec le diagramme de la conception est que dans ce dernier, on avait parfois trois fois l’action Index, ce qui n’est pas possible en ASP.NET (maximum 2 fois). Ensuite, on peut noter la présence du Controller ErrorController et du modèle Machine. Globalement, il n’y a pas de grandes différences.

### ViewModels



## DAL

La partie DAL (Data Access Layer) englobe les classes qui permettent l’accès à la base de données. C’est dans celles-ci qu’on effectue les différentes requêtes SQL.

### 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

Cet exemple a été pris afin d’illustrer la récupération de données (= SELECT) car cette méthode est la plus complexe (transaction, nombreuses requêtes sur plusieurs tables).

public List<Machine> GetFacilityMachines(int facility)

{

//On récupère les propriétés de la machine

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 List<Machine>();

//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;

}

Dictionary<int, int> versions = new Dictionary<int, int>();

foreach (int key in machines.Keys)

{

//On récupère la version de la machine

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

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

sqlR = db.ExecuteQuery(CMachines.GET\_MACHINE\_VERSION, parameters);

int version;

if (!Int32.TryParse(sqlR.Data[0][0], out version))

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

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

$"(Value:{sqlR.Data[0][0]}). " +

$"Please contact the administrator.");

versions[key] = version;

//On récupère les composants de la machine

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

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

sqlR = db.ExecuteQuery(CMachines.GET\_COMPONENTS, parameters);

if (sqlR.ErrorMessage != "")

{

throw new Exception(sqlR.ErrorMessage);

}

List<StreamXComponent> components = new List<StreamXComponent>();

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

{

StreamXComponent component = new StreamXComponent();

component.Version = line[0];

component.Name = line[1];

component.Log = (line[2] == "0") ? false : true;

components.Add(component);

}

machines[key].InfosStreamX.StreamXComponents = components;

//On récupère les disques durs de la machine

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

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

sqlR = db.ExecuteQuery(CMachines.GET\_HARDDRIVES, parameters);

if (sqlR.ErrorMessage != "")

{

throw new Exception(sqlR.ErrorMessage);

}

List<CustomDrive> drives = new List<CustomDrive>();

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

{

CustomDrive drive = new CustomDrive();

drive.Name = line[0];

drive.VolumeLabel = line[1];

drive.TotalSize = Double.Parse(line[2]);

drive.UsedSpace = Double.Parse(line[3]);

drives.Add(drive);

}

machines[key].InfosStreamX.StreamXComponents = components;

machines[key].InfosMachine.CustomDrives = drives;

}

//On construit une liste de Machine à partir du Dictionary

List<Machine> machinesList = new List<Machine>();

foreach (int key in machines.Keys)

{

machinesList.Add(new Machine { Id = key, Properties = machines[key], Version = versions[key] });

}

return machinesList;

}

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.

Puis, on utilise la méthode SetMachineProperty afin d’appliquer la propriété à l’objet Info (cf. : Chapitre « Helpers »).

Après, on récupère les composants StreamX ainsi que les disques durs et on les assigne aux objets Info du Dictionary.

Enfin, on construit une liste de Machine à partie du Dictionary.

### Insertion d’une nouvelle machine StreamX

Cet exemple a été pris pour illustrer l’insertion et la suppression de données à cause de sa complexité et car on y effectue des requêtes INSERT et DELETE.

public MySqlReturn InsertMachine(int facility, Info machine)

{

//On démarre la transaction

Transaction transaction = db.BeginTransaction();

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

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

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

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

if (!sqlR.IsOk)

{

//On annule la transaction

transaction.Rollback();

if (sqlR.ErrorMessage == "")

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

return sqlR;

}

int systemId = sqlR.LastInsertedId;

//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 = transaction.ExecuteQuery(CMachines.INSERT\_MACHINE\_PROPERTIES, parameters);

if (!sqlR.IsOk)

{

transaction.Rollback();

if (sqlR.ErrorMessage == "")

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

return sqlR;

}

}

//On insert les composents StreamX

foreach (StreamXComponent component in machine.InfosStreamX.StreamXComponents)

{

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

parameters.Add("@version", component.Version);

parameters.Add("@name", component.Name);

parameters.Add("@log", 0);

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

sqlR = transaction.ExecuteQuery(CMachines.INSERT\_COMPONENT, parameters);

if (!sqlR.IsOk)

{

transaction.Rollback();

if (sqlR.ErrorMessage == "")

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

return sqlR;

}

}

//On insert les disques durs

foreach (CustomDrive drive in machine.InfosMachine.CustomDrives)

{

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

parameters.Add("@name", drive.Name);

parameters.Add("@label", drive.VolumeLabel);

parameters.Add("@capacity", drive.TotalSize);

parameters.Add("@usedSpace", drive.TotalSize - drive.TotalFreeSpace);

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

sqlR = transaction.ExecuteQuery(CMachines.INSERT\_HARDDRIVE, parameters);

if (!sqlR.IsOk)

{

transaction.Rollback();

if (sqlR.ErrorMessage == "")

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

return sqlR;

}

}

transaction.Commit();

transaction.EndTransaction();

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. Insertion de la machine dans la table « T\_System »
3. Récupération de la dernière PK insérée
4. Insertion de chaque propriété dans la table « T\_System\_Property »
5. Insertion des composants StreamX
6. Insertion des disques durs
7. Si tout s’est bien passé durant la transaction, on la « commit »

### Helpers

Comme la table « T\_System\_Property » n’est qu’une association clé - valeur, il doit être possible d’identifier à quelle propriété de l’objet Info appartient la donnée retournée depuis la base. Pour ce faire, on fait appel à une classe « Helper » qui contient différentes méthodes utiles à la classe DalMachine :

* SetMachineProperty(Info machine, int idProperty, string value)

On utilise cette méthode afin d’ajouter à l’objet Info donné, la propriété donnée. La méthode 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.

* GetMachineProperties(Info machine)

Cette méthode convertit un objet Info en Dictionary<int, Object>. Elle est utilisée lors de l’insertion d’une machine dans la base de données :

Dictionary<int, Object> properties = new Dictionary<int, object>

{

{ CSystemProperties.STREAMX\_VERSION, machine.InfosStreamX.Version },

{ CSystemProperties.STREAMX\_FILENAME, machine.InfosStreamX.Filename },

...

};

return properties;

## Controllers

Les Controllers servent de liaison entre la Dal et les vues. En effet, lorsqu’on souhaite afficher une vue on appelle l’action du Controller liée. Généralement, un appel vers la Dal y est effectué et on agit en fonction du résultat (affichage de la vue, redirection, affichage d’un message d’erreur/succès). Par exemple, prenons l’action Index du Controller FacilitiesController :

public ActionResult Index(int enterprise)

{

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

Enterprise enterpriseObj = null;

try

{

enterpriseObj = dal.GetEnterprise(enterprise);

}

catch (ArgumentOutOfRangeException)

{

TempData["Error"] = "The enterprise you request doesn't exist anymore.";

return Redirect("/Enterprises/Index");

}

return View(new FacilitiesViewModel {

Enterprise = enterpriseObj,

Facilities = facilities

});

}

Ici, nous récupérons les ouvrages de l’entreprise dont l’ID est obtenu via les paramètres de l’URL. Ensuite, nous tentons d’obtenir l’entreprise elle-même à l’aide de l’ID.

Si l’exception ArgumentOutOfRangeException se produit, nous plaçons un message d’erreur dans le « TempData » afin de l’afficher après la redirection.

Dans le cas contraire, nous retournons la vue en lui passant le ViewModel.

## Vues (Views)

Les vues sont les différentes pages du site web. Elles sont appelées depuis un Controller. Le code écrit est en HTML mais on peut le mélanger à du C# afin d’afficher des informations serveur. Prenons comme exemple la page de connexion :

@model StreamScan.Models.Login

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

ViewBag.Title = "Login";

}

<script>

//Sélectionne le bon menu dans la barre de navigation

document.querySelectorAll(".nv-login").forEach(function (el) {

el.className += " active";

})

</script>

@if (User.Identity.IsAuthenticated)

{

<p>You are already authenticated.<p>

}

else

{

using (var form = Html.Bootstrap().Form().AddCss(Css.ColMd6, Css.Row).Begin())

{

@form.InputFor(m => m.Username).AddLabelCss("required")

@Html.ValidationMessageFor(m => m.Username)

@form.InputFor(m => m.Password, FormInputType.Password).AddLabelCss("required")

@Html.ValidationMessageFor(m => m.Password)

@form.Submit("Login");

}

}

Le symbole « @ » est utilisé afin de dire à l’application qu’il s’agit de code C#.

En première ligne, on peut voir le modèle de la page. Ce dernier est un objet passé depuis le Controller. On peut alors l’utiliser afin d’afficher les informations contenues.

Le Layout représente la mise en page de chaque page du site.

Le ViewBag est un autre moyen, comme le modèle, pour passer des informations. Ici, on définit le titre de la page qui sera utilisé dans le Layout.

Ensuite, on test si l’utilisateur est authentifié. Si ce n’est pas le cas, on affiche le formulaire de connexion.

Le formulaire en question est réalisé avec du code C# et non pas du HTML. On utilise ici le framework **FluentBootstrap** qui permet de générer du code HTML avec Bootstrap intégré.

### Layout

Le Layout permet d’avoir la même mise en page pour chaque page de l’application. En effet, le Layout est défini sur chaque page afin que ces dernières ne contiennent que le contenu de la page. On retrouve donc dans le Layout les éléments comme :

* Le head
* La barre de navigation
* Le footer

On y gère, par exemple, les menus affichés en fonction de l’authentification :

<ul class="nav navbar-nav navbar-left">

@if (Request.IsAuthenticated)

{

<li class="nv-scan">@Html.ActionLink("Scan", "Index", "Scan")</li>

<li class="nv-machines">

@Html.ActionLink("StreamX Machines", "Index", "Machines")

</li>

<li class="nv-entfac">

@Html.ActionLink("Enterprises and facilities", "Index", "Enterprises")

</li>

}

</ul>

<ul class="nav navbar-nav navbar-right">

@if (!Request.IsAuthenticated)

{

<li class="nv-login">@Html.ActionLink("Login", "Index", "Account")</li>

}

else

{

<li class="nav-item dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown"

role="button" aria-haspopup="true" aria-expanded="false">

@User.Identity.Name <span class="caret"></span>

</a>

<ul class="dropdown-menu">

<li> @Html.ActionLink("Logout", "Logout", "Account")</li>

</ul>

</li>

}

</ul>

Html.ActionLink génère un lien vers une certaine action spécifiée.

On définit à quel endroit sera intégrée la page avec :

@RenderBody()

#### TempData

Le TempData, comme le Model ou le ViewBag, permet de stocker des données. L’avantage de celui-ci est qu’il garde en mémoire ses variables après un Redirect. Il est utilisé dans le Layout afin d’afficher des messages de succès ou d’erreur après une redirection.

@if (@TempData["Message"] != null)

{

@Html.Bootstrap().Alert(AlertState.Success, "" + TempData["Message"])

}

@if (@TempData["Error"] != null)

{

@Html.Bootstrap().Alert(AlertState.Danger, "" + TempData["Error"])

}

## ViewModels

Les ViewModels permettent de stocker des informations depuis un Controller afin de les transmettre à la vue. De la même manière qu’on utilise un model dans la vue, le ViewModel permet de stocker, en plus d’un model, d’autres informations. Par exemple, le ViewModel FacilitiesViewModel est utilisé sur la page Index du Controller Facilities.

En effet, on y affiche la liste des ouvrages. Cependant, nous avons également besoin d’avoir l’entreprise qui les contient pour afficher son nom ainsi que connaître son ID pour le transmettre aux pages de mise à jour ou d’ajout. Ces dernières en ont besoin pour savoir dans quelle entreprise on ajoute/met à jour l’ouvrage.

public class FacilitiesViewModel

{

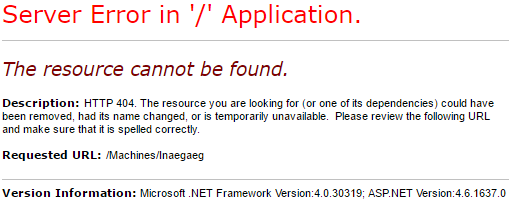
public Enterprise Enterprise { get; set; }

public List<Facility> Facilities { get; set; }

}

## Pages d’erreur

Afin de ne pas avoir des pages d’erreur comme ci-dessous, des pages d’erreur personnalisées ont été mises en place.



La redirection en cas d’erreur ou d’exception se fait en ajoutant le code suivant dans le Web.config :

<customErrors defaultRedirect="/Error" mode="On" >

<error statusCode="403" redirect="/Error/Error403" />

<error statusCode="404" redirect="/Error/Error404" />

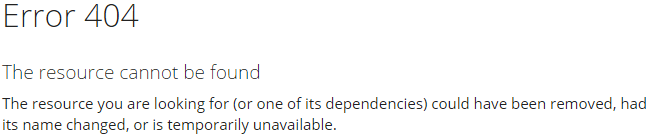
<error statusCode="500" redirect="/Error/Error500" />

</customErrors>

On redirige sur une page d’erreur en fonction du code d’erreur ou sur celle par défaut si l’erreur est inconnue.

Les exceptions sont également gérées manuellement dans certains Controllers, ce qui permet de transmettre le message de l’exception à la page d’erreur via le TempData avant la redirection.

Voici ce qu’on obtient pour une erreur 404 :



## Problèmes rencontrés

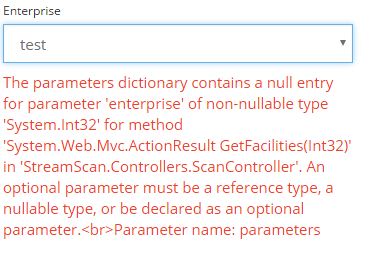
Très peu de problèmes sérieux ont été rencontrés. Ceux expliqués ci-dessous illustrent des erreurs « d’oubli » résolues entre 30 minutes et 1 heure.

### 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 informations (= sérialisation). L’exception suivante est obtenue :

System.ServiceModel.Dispatcher.DataContractSerializerOperationFormatter...

### Paramètres de la route non-optionnels



L’erreur ci-dessus apparaît lorsqu’on donne une valeur NULL à une action qui attend un Int32. Par exemple :

public ActionResult New(int enterprise)

{

return View(new NewFacilityViewModel { Enterprise = enterprise });

}

L’action ci-dessus requiert un entier en paramètre. Si aucun paramètre n’est transmis (donc valeur NULL), on obtient cette exception.

Ici, le problème est que lors de tests, les paramètres, définit à la base comme optionnel, de la route par défaut ont été définit comme requit et en tant que string vide.

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{enterprise}/{facility}",

defaults: new { controller = "Machines",

action = "Index",

enterprise = UrlParameter.Optional,

facility = UrlParameter.Optional }

);

Le code ci-dessus a été changé en :

defaults: new {

controller = "Machines",

action = "Index",

enterprise = "",

facility = "" }

L’action requiert un entier mais on définit un string dans la route, alors NULL est donné.

## 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 informations si la connexion a été perdue | Message d’erreur | **OK** |
| Formulaire de scan | Récupération des informations sur un serveur existant | Affichage des informations 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* | **OK** |
| *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* | **OK** |
| 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 » | Affichage du formulaire de mise à jour | **OK** |
| Lien « Show facilities » | Clic du lien « Show facilities » | Affichage des ouvrages | **OK** |
| Bouton « Delete » | Clic du bouton « Delete » | Affichage d’une popup de confirmation | **OK** |
| Formulaire de confirmation | Clic « Cancel » | Disparition de la popup | **OK** |
| *Formulaire de confirmation* | *Clic « Delete » si déjà supprimé (cf. : « Tests spéciaux – D »)* | *Affichage message d’erreur* | ***OK*** |
| Formulaire de confirmation | Clic « Delete » | Suppression de l’entreprise (et de ses ouvrages + machines) | **OK** |

### Facilities

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

### 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 | **OK** |
| Formulaire d’ajout | Ajout avec un NPA invalide | Message d’erreur en dessous du NPA | **OK** |
| Formulaire d’ajout | Ajout de l’entreprise | Message de succès + redirection vers la liste | **OK** |

### 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 | **OK** |
| Formulaire d’ajout | Ajout avec un NPA invalide | Message d’erreur en dessous du NPA | **OK** |
| *Formulaire d’ajout* | *Ajout dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Message d’erreur* | ***OK*** |
| Formulaire d’ajout | Ajout de l’ouvrage | Message de succès + redirection vers la liste | **OK** |

### 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 | **OK** |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Message d’erreur en dessous du NPA | **OK** |
| *Formulaire de mise à jour* | *Mise à jour d’une entreprise inexistante (cf. : « Tests spéciaux – B »)* | *Affichage d’un message d’erreur* | ***OK*** |
| *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* | ***OK*** |
| Formulaire de mise à jour | Mise à jour de l’entreprise | Message de succès + redirection vers la liste | **OK** |

### 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 | **OK** |
| Formulaire de mise à jour | Mise à jour avec un NPA invalide | Message d’erreur en dessous du NPA | **OK** |
| *Formulaire de mise à jour* | *Mise à jour dans une entreprise inexistante (cf. : « Tests spéciaux – A »)* | *Message d’erreur* | ***OK*** |
| *Formulaire de mise à jour* | *Mise à jour d’un ouvrage inexistant (cf. : « Tests spéciaux – B »)* | *Affichage d’un message d’erreur* | ***OK*** |
| *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* | ***OK*** |
| Formulaire de mise à jour | Mise à jour de l’ouvrage | Message de succès + redirection vers la liste | **OK** |

### Validation des tests fonctionnels

|  |  |  |
| --- | --- | --- |
| **Date** | **Nom** | **Signature** |
| 09.06.2017 | Laurent Chassot |  |

# Conclusion

## Conclusion personnelle

La réalisation de ce TPI fût très intéressante. J’ai eu beaucoup de plaisir à réaliser cette application car, en plus de ma passion pour le web, cela m’a permis d’améliorer mes connaissances en ASP.NET, que je ne connaissais pas quelques mois auparavant. Le résultat obtenu répond à mes attentes et j’en suis très content, même si j’aurais souhaité améliorer certains détails de l’application. De plus, je suis content de savoir que l’application sera utilisée par Infoteam.

Le planning à globalement été respecté. Les phases d’analyse et de conception ont cependant pris beaucoup plus de temps que prévu dans la planification.

J’ai apprécié de passer par des phases d’analyse et de conception car celles-ci n’étaient pas effectuées complètement lors des autres projets que j’ai eus durant mon stage. Il était intéressant de réaliser les différents schémas (MCD, MLD, maquettes, diagrammes d’activités,…) et les autres étapes car en premier lieu on ne se rend pas compte de l’importance qu’ils ont mais en prenant le temps de les faire, on réalise toute leur utilité car on se base sur eux et le temps « perdu » à les faire est récupéré après.

Je pense également que réaliser ce projet m’a aussi appris à ne pas négliger ou sous-estimer certaines tâches de l’application comme la modélisation de la base de données ou de prendre le temps de chercher une meilleure solution à un problème.

## Etat du projet

L’application finie comprend toutes les fonctionnalités spécifiées dans le cahier des charges. Chacune d’entre elles a été testée et les cas spéciaux traités. La récupération des données est complète mais des détails restent à ajouter.

Une attention à l’égard du design peut également être prise car même si il est selon moi attractif, la mise en page pourrait être revue. Avoir par exemple, les formulaires avec plusieurs champs par ligne, de façon à casser la structure des formulaires et de diminuer l’espacement à droite car tout est aligné à gauche.

## Améliorations possibles

Les objectifs de base du cahier des charges ont pu être réalisés. On peut cependant définir des améliorations à l’application.

### Base de données

* Ajout de la date de mise à jour d’une machine dans la base de données
* Ajout d’un historique des modifications par machine dans la base de données
* Mots de passes hachés dans la base de données

### Application

* Ajout des protocoles de communication d’une machine : ceux-ci sont listés parmi les dossiers de logs mais ne sont pas explicitement obtenus ni sauvegardés. De ce fait, cette fonctionnalité est négligeable mais il serait intéressant et plus propre de la mettre en place
* Ajout d’une page détaillée par machine : pour l’instant les informations sont toutes disponibles sous la forme d’un tableau « scrollable » horizontalement mais il serait intéressant d’avoir une page spécifique pour chaque machine.
* Ajout du choix du protocole de communication pour le scan : le serveur implémente les connexions WCF (web service) et TCP. Le client est actuellement limité par une communication WCF uniquement.
* Suppression d’une machine : de base on considère qu’une machine ne doit pas être supprimée. Cependant, il est possible que les informations d’une machine ne correspondent plus à une machine existante au bout d’un certain temps. C’est pourquoi il serait intéressant de pouvoir la supprimer.
* Amélioration du design, spécialement au niveau du positionnement des éléments sur la page afin de réduire l’espacement à droite

# Remerciements

Je voudrais tout d’abord remercier mes deux experts, M. Nicolas Richoz et M.Vincent Gremaud, d’avoir accepté d’évaluer mon projet, d’y consacrer du temps et d’avoir émis des remarques pertinentes lors des visites.

Je remercie également M. Pierre-Alain Mettraux pour le temps consacré au projet et pour ses remarques lors des visites.

Enfin, un grand merci à M. David Godel pour son soutien et son suivi technique au long du projet et Mme. Laurence Dubied Bielmann pour son suivi sur le rapport et pour ses conseils.

# Bibliographie: liste des sources et références

Les sources sont citées en-dessous des images ou en tant que note de page.

# Glossaire

|  |  |
| --- | --- |
| Terme | Signification |
| Ouvrage | Un ouvrage est un bâtiment équipé et en production (ex. : Poste électrique, barrage, centrale de chauffage à distance,…) qui compose une entreprise |
| Route | Système décomposant un Controller en actions ayant plusieurs paramètres. Utilisé dans l’URL sous la forme « /Controller/Action/[param1]/[param2] » |
| StreamX | Produit développé par Infoteam composé principalement d’une passerelle de communication |
| ViewBag | Système permettant le stockage d’informations afin de les transmettre entre action et vue. Utilisé dans ce projet uniquement pour le stockage du titre de la page. |
| ViewModel | Système permettant le stockage d’informations afin de les transmettre entre action et vue. Permet l’auto-complétion contrairement au ViewBag et est plus organisé |

# 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.

|  |  |  |
| --- | --- | --- |
| **Date** | **Nom** | **Signature** |
| 13.06.2017 | Laurent Chassot |  |

# Annexes

* Organisation du projet
  + Cahier des charges
  + Planning
  + Journal de travail
* Librairies et scripts
* Projet
  + Code source de l’application
  + Tests technologiques
* Diagrammes UML et schémas
* Manuel de l’utilisateur
* Notes des visites des experts
* Documentation (docx)

1. *Source : Cahier des charges - Introduction* [↑](#footnote-ref-1)
2. *Source de l’image : Document de formation - transactions (module 223 – EMF)* [↑](#footnote-ref-2)