

DESCRIPTION DETAILLÉE DES MISSIONS RÉALISÉES

Application de gestion des zones de stockage de GSB

BTS SIO Option SLAM

Bassette Chancereul Gwendoline

Du 07/10/2019 au 06/11/2019

Table des matières

I-	Contexte	2
II-	Présentation du PPE.....	2
1-	Besoin	2
2-	Contraintes	2
3-	Architecture de l'application	3
4-	Description du domaine de gestion	4
5-	Schéma de la base de données.....	6
III-	Description détaillée du PPE.....	9
1-	Consultation des contrôles effectués.....	9
2-	Ajout d'un contrôle effectué	17
3-	Modification d'un contrôle effectué	29
4-	Suppression d'un contrôle effectué	38

I- Contexte

Le laboratoire Galaxy Swiss Bourdin (GSB) est issu de la fusion entre le géant américain Galaxy (spécialisé dans le secteur des maladies virales dont le sida et les hépatites) et le conglomérat européen Swiss Bourdin (travaillant sur des médicaments plus conventionnels), lui-même déjà union de trois petits laboratoires.

En 2009, les deux géants pharmaceutiques ont uni leurs forces pour créer un leader de ce secteur industriel. L'entité Galaxy Swiss Bourdin Europe a établi son siège administratif à Paris.

Le siège social de la multinationale est situé à Philadelphie, Pennsylvanie, aux Etats-Unis.

Le groupe GSB produit et distribue des médicaments dans de nombreux pays.

Des contrôles sont effectués sur les zones de stockage afin de permettre d'assurer :

- La bonne conservation des substances et des médicaments (l'humidité de l'air est par exemple un des facteurs de dégradation des principes actifs des médicaments, par ailleurs beaucoup de principes actifs sont photosensibles et ne doivent pas être exposés directement à la lumière du jour)
- La sécurité du personnel et des marchandises stockées.
- La protection de l'environnement contre des accidents (explosion, incendie, infiltration de produits dangereux dans le sol ...)

L'entreprise GSB dispose de plusieurs zones de stockages pour entreposer les médicaments qu'elle fabrique et les substances utilisées pour les fabriquer (poudre, gaz...). Elle demande à des entreprises spécialisées de réaliser régulièrement différents contrôles au sein de ses zones de stockage.

II- Présentation du PPE

1- Besoin

Actuellement, il n'existe aucun suivi des contrôles réalisés dans les zones de stockage de GSB.

Il faut donc développer une application permettant d'enregistrer les caractéristiques des zones de stockage et des entreprises habilitées à les contrôler. La gestion des contrôles réalisés doit également être prise en charge par l'application.

L'application Windows Dotnet à développer sera accessible à toute personne (salarié et directeur) du service Sécurité, au directeur du service Production et au directeur du service financier pour le suivi des contrôles

L'accès ne sera possible que pour les acteurs concernés et à partir d'un poste de l'entreprise GSB. L'authentification préalable est nécessaire pour accéder à l'application, les fonctionnalités proposées à l'utilisateur dépendront de son rôle.

2- Contraintes

Architecture

L'application respectera l'architecture en couche (GUI, BL, DAL, BO.)

Chaque composant BL, DAL, et BO sera une bibliothèque de classes.

Ergonomie et charte graphique

Aucune charte graphique mais il doit y avoir une uniformité dans les formulaires.

Codage

Les règles de bonnes pratiques de développement utilisées au cours des deux années de BTS pour encadrer le développement d'applications en C# et en faciliter la maintenance doit être respecté.

Les éléments à fournir devront respecter le nommage des fichiers, des classes, des variables, des paramètres, des composants graphiques...

Chaque classe et méthode sera documentée. La qualité de la documentation sera évaluée.

Environnement

Utilisation du framework Dotnet. Le langage à utiliser est le C#.

Les données

La base de données sera gérée par le SGBD Sql Server.

La couche DAL ne contiendra pas de requête SQL mais fera appel à des procédures stockées dans la base de données. La gestion des erreurs sera prise en compte dans les procédures stockées (les erreurs retournées par la procédure stockée doivent être enregistrées dans Sql Server)

Pour chaque enregistrement d'une table on conservera la date de création de l'enregistrement et la dernière date de mise à jour.

Pour chaque intervention sur les données d'une table (ajout, modification, suppression), il est nécessaire de garder la trace de l'intervention (utilisateur responsable de l'intervention, table concerné, opération réalisée, date et heure) dans une table TraceIntervention.

Le nom des procédures stockées devra commencer par sp et sera suivi d'une expression significative. Exemple : splnsClient est une procédure stockée qui permet d'ajouter un enregistrement dans la table client, spqCnsClientParNom est une procédure stockée qui permet de consulter la table client par nom de client.

Le nom des triggers devra commencer par trg et sera suivi d'une expression significative.

Documentation à rendre

- Diagramme de cas d'utilisation
- Un plan de test et un dossier de test pour chaque fonctionnalité
- Dossier technique de l'application (liste des bases de données utiles, MCD des bases de données, comptes applicatifs, description de l'architecture applicative, liste et description des composants (diagramme de classes), liens entre composants, liste des procédures et objectifs...)
- Dossier de mise en production : script de création ou de modification des bases de données, comptes à créer, .exe à copier (nouveau ou à remplacer), procédure de retour arrière, procédure de test de vérification d'installation.
- Documentation utilisateur (mode opératoire utilisateur)

Responsabilités

Vous livrerez un système opérationnel, une base de données exemple avec un compte de test, la documentation spécifique (C.F ci-dessus) permettant un transfert de compétences et un mode opératoire propre à chaque module.

Equipe : 5 personnes

Langages : Application Windows Form en C#, Framework .NET, SQL

Logiciels utilisés : Visual Studio 2017, SQL Server Management Studio 2018

3- Architecture de l'application

L'application doit respecter l'architecture en couche.

Elle est composée de quatre couches communiquant les unes avec les autres :

- **L'interface graphique** (ou GUI). Elle contient les formulaires responsables de l'affichage des données à l'utilisateur. Les données transmises dans l'interface graphique sont transmises à la couche logique métier.
- Les **objets métier** (ou BO) contient une table pour chaque table de la base de données. Elle gère les classes contenant la description de tous les objets métiers manipulés dans l'application.

- La **couche logique métier** (ou BLL) est responsable du traitement, du contrôle des données saisies dans l'interface graphique. Elle dialogue avec la couche d'accès aux données grâce à des objets pour obtenir et modifier les données enregistrées dans une base de données ou dans un fichier.
- La **couche d'accès aux données** (ou DAL) est responsable de la création des objets métiers à partir de la base de données et de la mise à jour des données de la base de données. Chaque classe de la couche données est responsable de la lecture, l'ajout, la modification et de la suppression des données d'une table spécifique de la base de données.

L'architecture en couche présente plusieurs avantages :

- Le travail en équipe est optimisé. Un membre de l'équipe peut travailler sur la couche d'accès aux données, pendant qu'un autre peut travailler sur la couche métier ou sur l'interface graphique.
- La migration d'un environnement graphique à un autre est relativement simple : grâce au découpage en couche, la couche d'accès aux données et la couche métier seront réutilisés, et seule la couche GUI devra être réécrite (passer de Windows à ASP.NET) pour appeler la couche BLL existante.
- Si le support physique de stockage des données change (passage d'une base de données ACCESS à une base de données Oracle) alors seule la couche DAL devra être réécrite.
- L'évolution des traitements (contrôle...), y est simplifiée car seule une couche est à modifier (BLL), la couche de présentation délègue donc la gestion de la partie métier à la couche BLL.

4- Description du domaine de gestion

Les zones de stockage

GSB dispose de plusieurs zones de stockages pour les produits. Les informations caractérisant les zones de stockage sont le nom de la zone, l'emplacement (bâtiment, étage, adresse postale). Chaque zone de stockage n'accueillera qu'une seule catégorie de produits. Il existe plusieurs catégories de produit (exemples : composés sanguins, stupéfiants, gaz...)

Voici un extrait de la liste des catégories de produits :

- Médicament classe I (aérosol)
- Médicament classe II (solution buvable)
- Médicament classe III (poudre)
- Médicament classe IV (cachets)
- Vaccin classe I
- Vaccin classe II
- Vaccin classe III
- Gaz médical
- Composés sanguins
- Produits radiopharmaceutiques
- Stupéfiants
- Poudre pour solution type A
- Poudre pour solution type B
- Poudre pour solution type C

Les types de contrôles et les contrôles

Il existe plusieurs types de contrôles (de poids, d'humidité, de pression, de luminosité, de température, de propreté, de sécurité, d'incendie...)

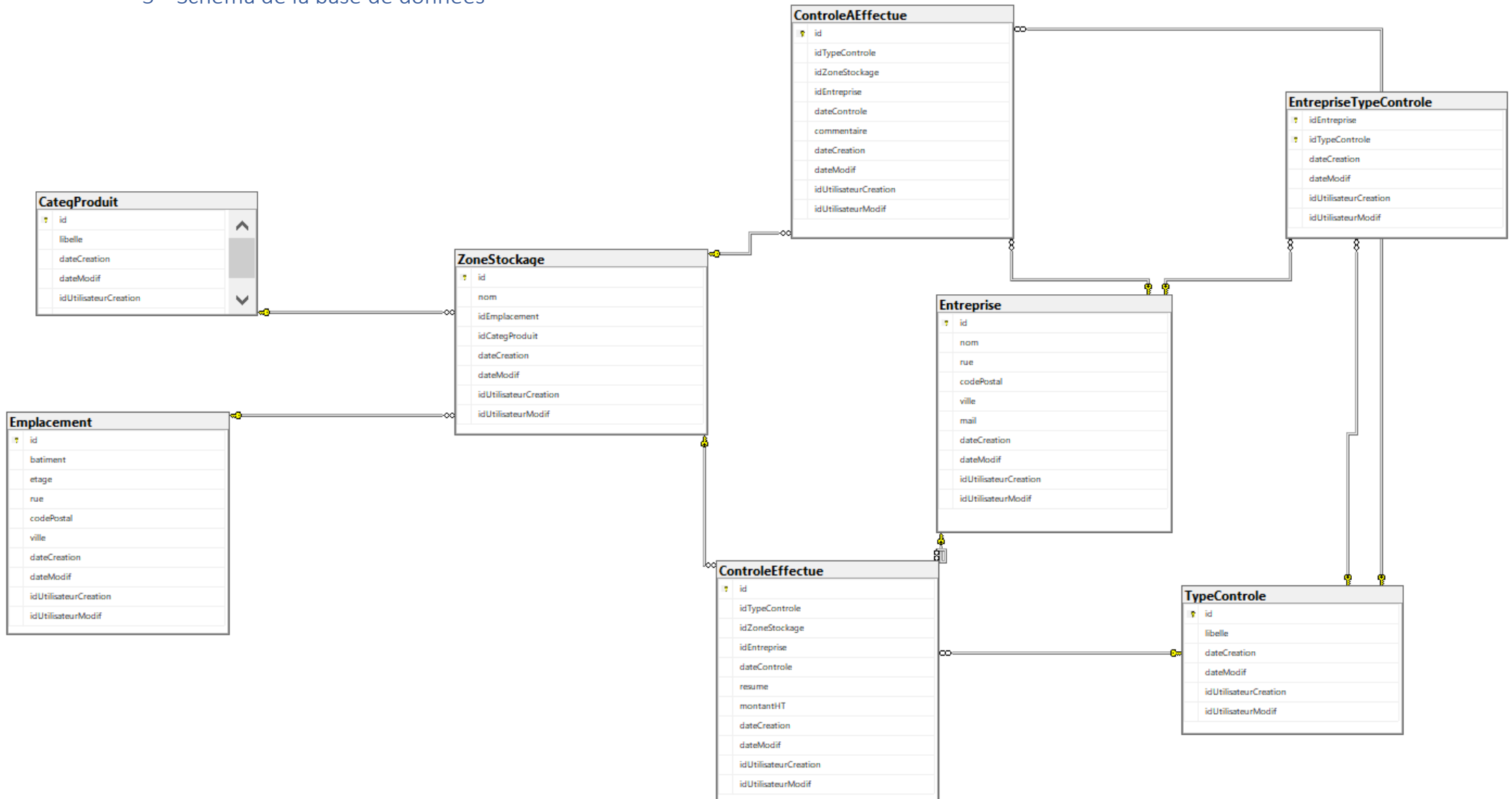
Chaque zone de stockage peut subir plusieurs types de contrôles. Par exemple la zone de stockage située au 4^{ème} étage du bâtiment 1 sise 6 rue des cordeliers à Rougemont doit subir des contrôles d'humidité (celle-ci doit être inférieure à 30%) ; de sécurité (les fenêtres doivent être protégées et les serrures très solides car des produits stupéfiants sont stockés dans cette zone) et d'incendie (état des extincteurs)

Il est donc nécessaire d'enregistrer pour chaque zone de stockage les types de contrôles à réaliser : s'il y a lieu un commentaire pourra être indiqué pour préciser le détail des contrôles à réaliser (exemple : humidité inférieure à 30%). GSB demande régulièrement à des entreprises spécialisées de contrôler ses zones de stockage. Pour chaque contrôle réalisé on enregistrera la zone de stockage concernée, la date du contrôle, un résumé du résultat du contrôle réalisé, le type de contrôle réalisé et le montant HT facturé pour le contrôle.

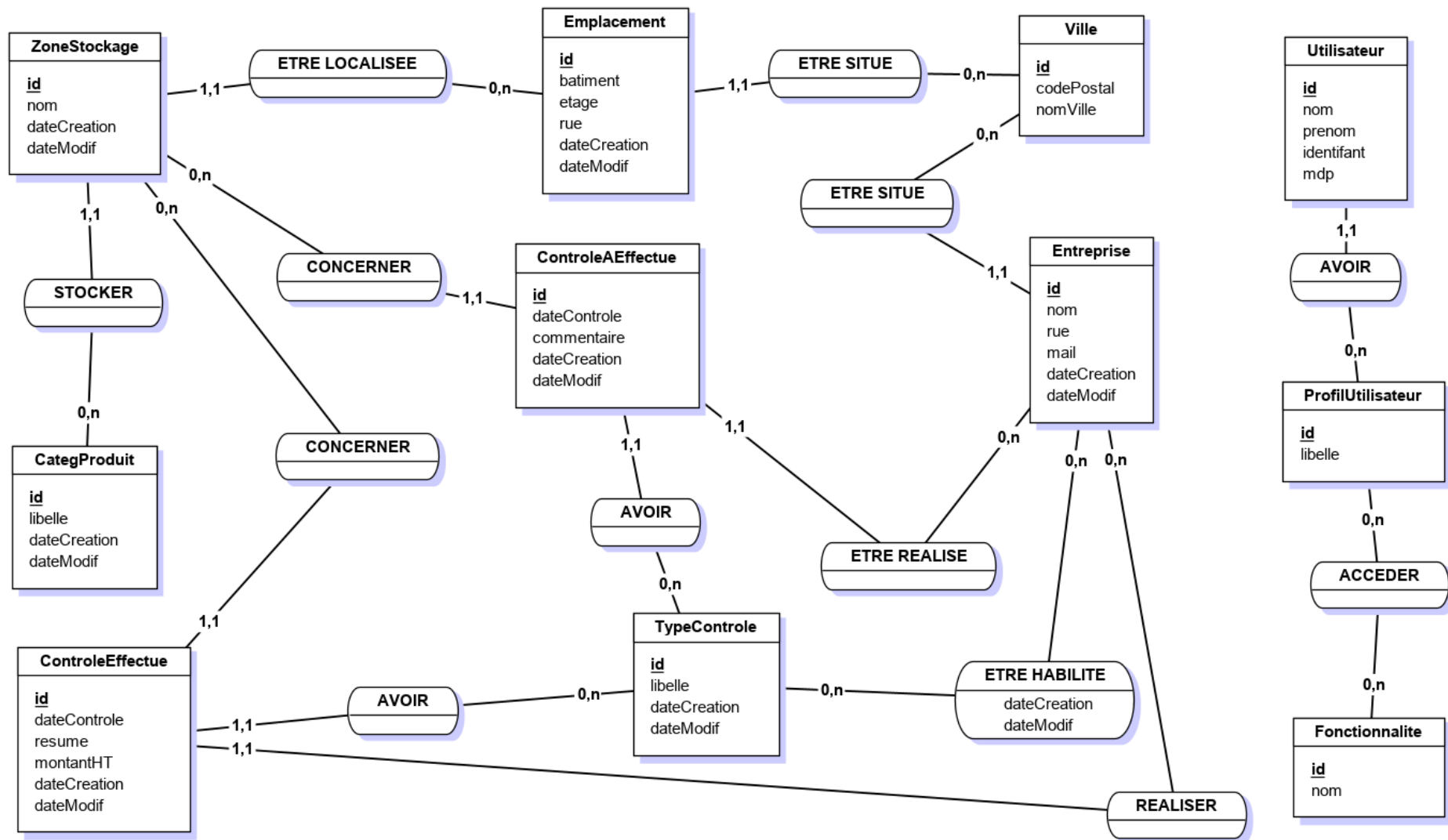
Les entreprises de contrôle

Pour chaque entreprise qui réalise un contrôle au sein de GSB il est nécessaire de connaître son nom, son adresse postale et son adresse mail afin de pouvoir la contacter. GSB souhaite également enregistrer les types de contrôles pour lesquels elle est habilitée.

5- Schéma de la base de données

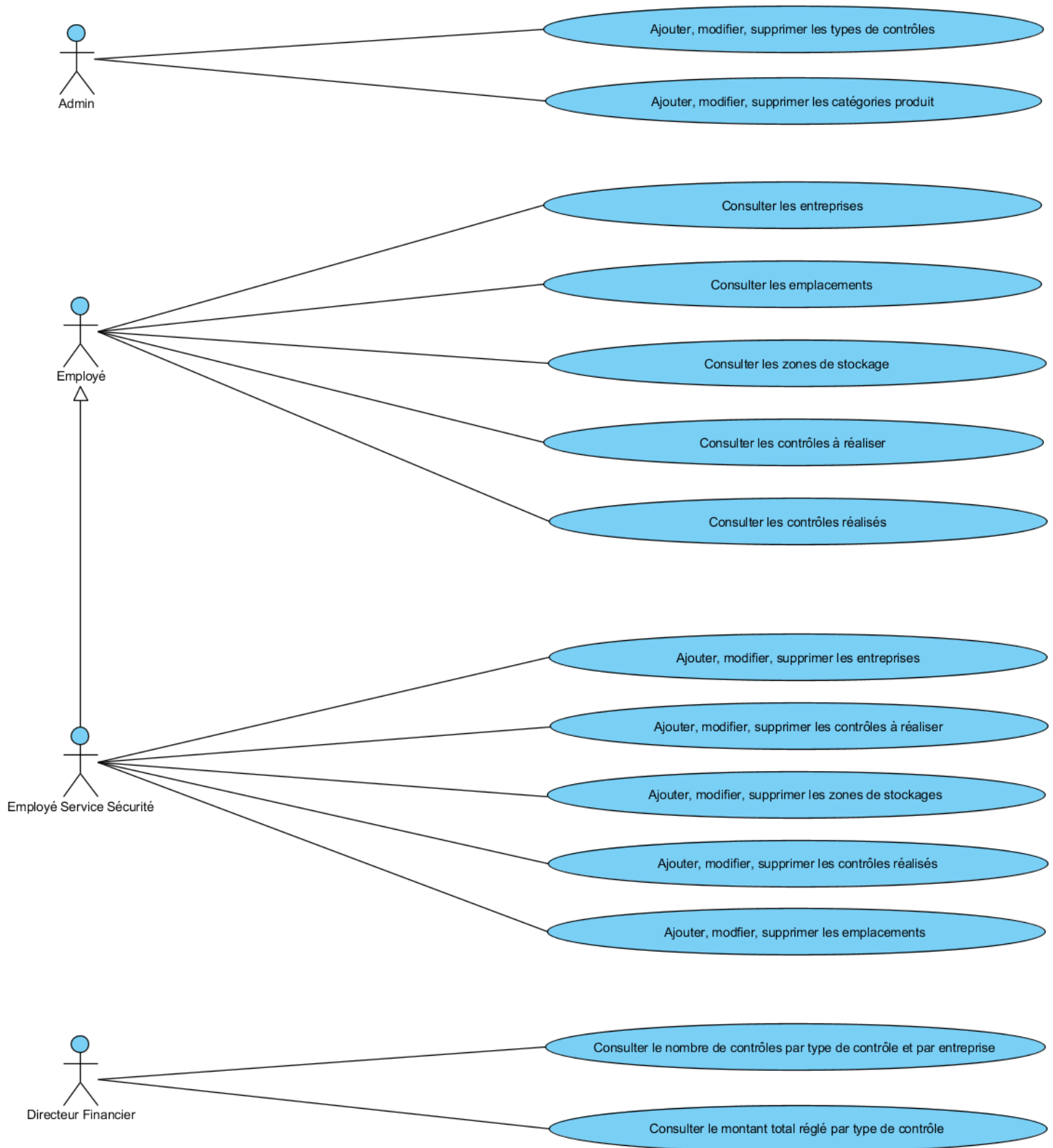


6- Modèle conceptuel des données



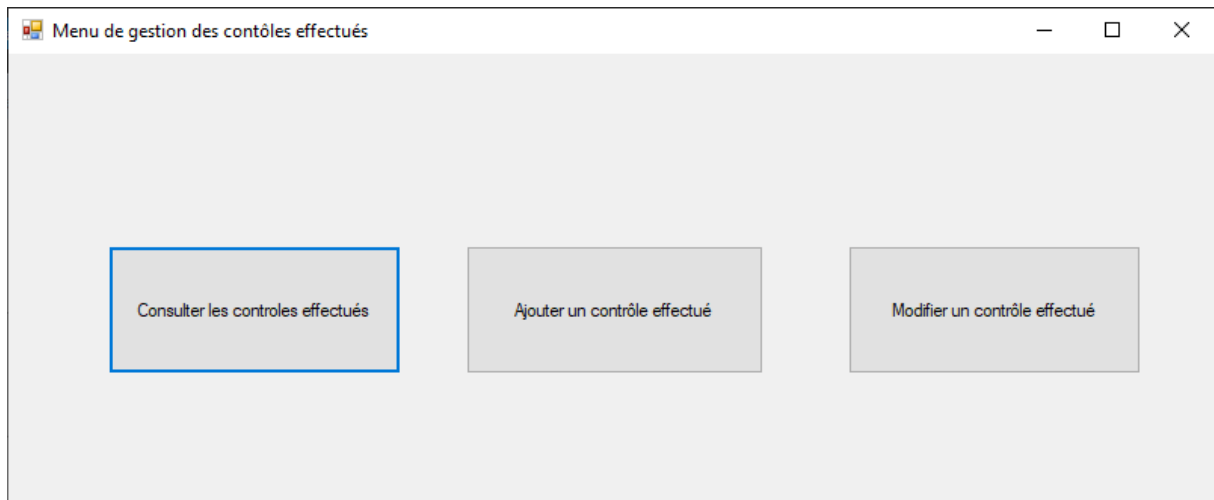
Le schéma de la base de données et le MCD sont simplifiés. Pour des raisons de lisibilité toutes les associations avec l'entité utilisateur n'apparaissent pas. Dans la base de données chaque table contient les champs `idUtilisateurCreat` et `idUtilisateurModif` clés étrangères sur la table `Utilisateur`, ce qui sur le MCD représenterait deux associations liées à l'entité `Utilisateur`.

7- Diagramme de cas d'utilisation



III- Description détaillée du PPE

La gestion des contrôles effectués se fait à l'aide du menu de gestion des contrôles effectués :

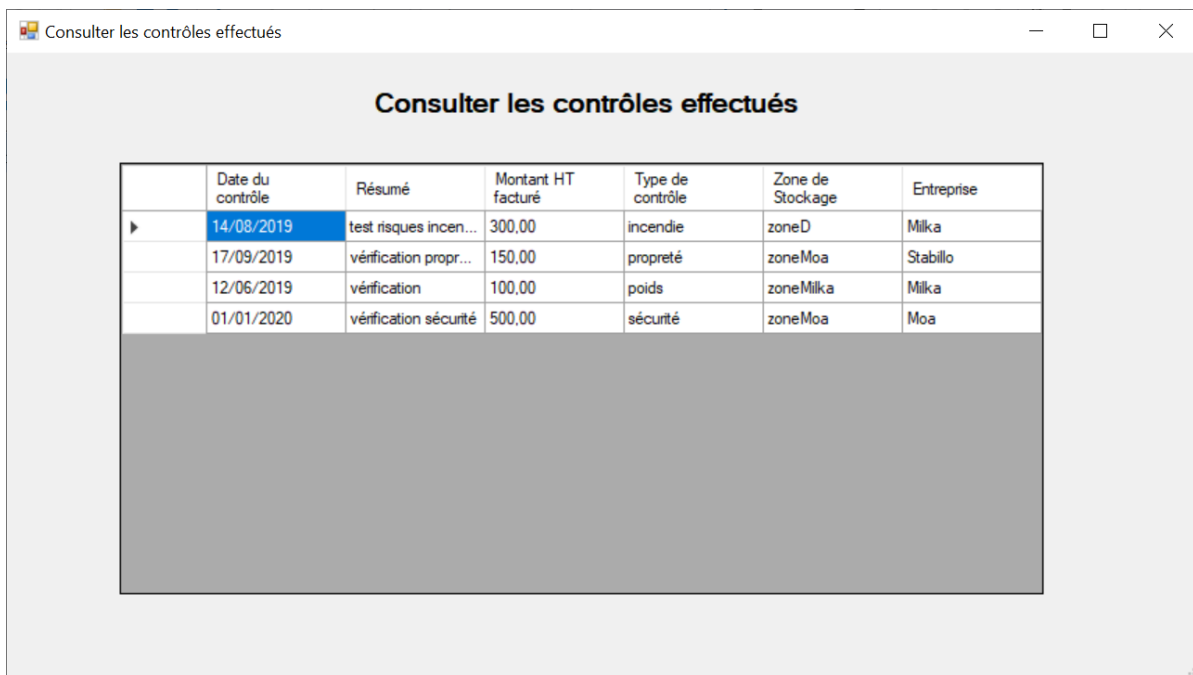


Le bouton « Consulter les contrôles effectués » permet la consultation de tous les contrôles qui ont été réalisés et qui sont enregistrés dans la base de données. Le bouton « Ajouter un contrôle effectué » permet de saisir un contrôle qui a été réalisé. Le bouton « Modifier un contrôle effectué » permet de modifier les informations d'un contrôle.

1- Consultation des contrôles effectués

La consultation de tous les contrôles ayant été réalisés doit être possible. Les données à afficher sont la date du contrôle, le résumé, le montant hors taxes facturé, le type de contrôle, la zone de stockage concernée par le contrôle et l'entreprise ayant réalisé le contrôle.

Voici la fonctionnalité de consultation :



Pour réaliser cette fonctionnalité, j'ai d'abord créé le formulaire dans le la couche GUI qui gère l'interface graphique. Pour afficher les informations, j'ai utilisé un DataGridView.

J'ai créé un nouveau fichier dans la couche BO que j'ai nommé `ControleEffectue` comme la table dans la base de données. Cette classe possède les mêmes caractéristiques que la table `ControleEffectue`. Les colonnes ayant des clés étrangères sur d'autres tables, sont des objets dans la couche BO. Ainsi, la colonne `idTypeControle` présente dans la base de données est `leTypeControle` de type `TypeControle`. Cet attribut sera une référence à un objet de type `TypeContrôle`.

```
private int id;
private TypeControle leTypeControle;
private ZoneStockage laZoneStockage;
private Entreprise lEntreprise;
private DateTime dateControle;
private string resume;
private decimal montantHT;
private DateTime dateCreation;
private DateTime dateModif;
private Utilisateur utilisateurCreat;
private Utilisateur utilisateurModif;
```

*Capture d'écran des attributs de la classe
ControleEffectue*

dbo.ControleEffectue

Colonnes

- id (PK, int, non NULL)
- idTypeControle (FK, int, NULL)
- idZoneStockage (FK, int, NULL)
- idEntreprise (FK, int, NULL)
- dateControle (date, NULL)
- resume (varchar(200), NULL)
- montantHT (numeric(6,2), NULL)
- dateCreation (date, NULL)
- dateModif (date, NULL)
- idUtilisateurCreation (FK, int, NULL)
- idUtilisateurModif (FK, int, NULL)

Capture d'écran de la table ControleEffectue

Pour pouvoir créer les attributs faisant référence à d'autres classes, j'ai dû réaliser les classes `TypeControle`, `ZoneStockage`, `Entreprise` et `Utilisateur` dont voici les attributs :

21 références

```
public class Utilisateur
{
    private int id;
    private string nom;
    private string prenom;
}
```

18 références

```
public class CategProduit
{
    private int id;
    private string libelle;
}
```

```
public class Entreprise
{
    private int idEntreprise;
    private string nomEntreprise;
    private string rue;
    private string codePostale;
    private string ville;
    private string mail;
}
```

28 références

```
public class TypeControle
{
    private int id;
    private string libelle;
    private DateTime dateCreation;
    private DateTime dateModif;
    private Utilisateur utilisateurCreat;
    private Utilisateur utilisateurModif;
}
```

```
public class ZoneStockage
{
    private int id;
    private string nomZone;
    private Emplacement lEmplacement;
    private CategProduit laCateg;
    private DateTime dateCreation;
    private DateTime dateModif;
    private Utilisateur utilisateurCreat;
    private Utilisateur utilisateurModif;
}
```

J'ai ensuite créé les constructeurs, les assesseurs en lecture et en écriture. Voici l'un des constructeurs de la classe `ControleEffectue` :

3 références

```
public ControleEffectue(int id, TypeControle typeControle, ZoneStockage zoneStockage,
    Entreprise entreprise, DateTime dateControle, string resume, decimal montantHT)
{
    this.id = id;
    this.leTypeControle = typeControle;
    this.laZoneStockage = zoneStockage;
    this.lEntreprise = entreprise;
    this.dateControle = dateControle;
    this.resume = resume;
    this.montantHT = montantHT;
}
```

Ci-dessous- les getters et setters de la classe ControleEffectue :

```
1 référence
public int Id { get => id; set => id = value; }
5 références
public TypeControle LeTypeControle { get => leTypeControle; set => leTypeControle = value; }
4 références
public ZoneStockage LaZoneStockage { get => laZoneStockage; set => laZoneStockage = value; }
4 références
public Entreprise LEntreprise { get => lEntreprise; set => lEntreprise = value; }
3 références
public DateTime DateControle { get => dateControle; set => dateControle = value; }
3 références
public string Resume { get => resume; set => resume = value; }
3 références
public decimal MontantHT { get => montantHT; set => montantHT = value; }
0 références
public DateTime DateCreation { get => dateCreation; set => dateCreation = value; }
0 références
public DateTime DateModif { get => dateModif; set => dateModif = value; }
0 références
public Utilisateur UtilisateurCreat { get => utilisateurCreat; set => utilisateurCreat = value; }
0 références
public Utilisateur UtilisateurModif { get => utilisateurModif; set => utilisateurModif = value; }
```

Puis j'ai créé une bibliothèque de classe GesStockageDAL qui contient les classes Connexion et ControleEffectueDAO. La classe Connexion est une classe statique responsable de la création d'un objet de la classe SqlConnection. Voici son contenu :

```
26 références
static public class Connexion
{
    static private SqlConnection objConnex;
    //le constructeur statique (appelé une seule fois)
    // crée un objet instance de la classe SqlConnection
0 références
    static Connexion()
    {
        objConnex = new SqlConnection();
        objConnex.ConnectionString = ConfigurationManager.ConnectionStrings["GSB_Stockage"].ConnectionString;
    }

    // la méthode GetObjConnexion fournit l'objet instance de
    // la classe SqlConnection dans un état "connexion ouverte"
14 références
    public static SqlConnection GetObjConnexion()
    {
        // on ouvre la connexion si elle est fermée
        if (objConnex.State == System.Data.ConnectionState.Closed)
        {
            objConnex.Open();
        }
        // on retourne l'objet responsable de la connexion
        return objConnex;
    }

    // la méthode CloseConnexion met l'objet instance de la classe
    // SqlConnection dans un état "connexion fermée"
11 références
    public static void CloseConnexion()
    {
        // si la connexion est ouverte on la ferme
        if (objConnex != null && objConnex.State != System.Data.ConnectionState.Closed)
        {
            objConnex.Close();
        }
    }
}
```

J'ai ensuite réalisé les instructions suivantes dans la classe ControleEffectueDAO. Cette classe utilise un « design pattern Sigleton », il s'agit d'une technique permettant de créer un seul objet pour une classe. Avec ces instructions on crée un attribut du type de la classe. Ensuite, si l'objet uneInstance n'existe pas on le crée puis on retourne l'objet uneInstance. Voici cette méthode :

```

9 références
public class ControleEffectueDAO
{
    private static ControleEffectueDAO uneInstance;
    //cette méthode crée un objet de la classe ControleEffectueDAO s'il n'en existe pas déjà
    //puis retourne la référence à cet objet
5 références
    public static ControleEffectueDAO GetInstance()
    {
        if (uneInstance == null)
        {
            uneInstance = new ControleEffectueDAO();
        }
        return uneInstance;
    }
    //le constructeur par défaut est privé : il ne sera donc pas possible de créer un objet à
    //l'extérieur de la classe avec l'instruction new
1 référence
    private ControleEffectueDAO()
    {
    }
}

```

Pour pouvoir obtenir tous les contrôles qui ont été effectués qui sont stockés dans la base de données j'ai réalisé la méthode getControlesEffectues(). Cette méthode retourne une collection contenant les contrôles effectués existants dans la table ControleEffectues.

Je commence donc par déclarer mes variables de travail. La collection lesControlesEffect sera la collection retournée par la méthode, maCommand va contenir la requête SQL permettant d'obtenir toutes les caractéristiques de tous les contrôles effectués, monLecteur permettra de récupérer les enregistrements retournés par la requête SQL, les autres variables permettront de créer les objets à ajouter à la collection lesControlesEffect. Voici ces instructions :

```

public List<ControleEffectue> GetControlesEffectues()
{
    //déclaration des variables de travail
    List<ControleEffectue> lesControlesEffect;
    SqlCommand maCommand;
    SqlDataReader monLecteur;
    int idLu;
    TypeControle unTypeControle;
    int idTypeControleLu;
    ZoneStockage uneZoneStockage;
    int idZoneStockageLu;
    Entreprise uneEntreprise;
    int idEntrepriseLu;
    DateTime dateControleLu;
    string resumeLu;
    decimal montantHTLu;
    string libelleZoneStockage;
    string libelleTypeControle;
    string nomEntrepriseLu;
    ControleEffectue unControleEffectue;
}

```

Après avoir déclaré les variables, on récupère l'objet responsable de la connexion à la base de données. La méthode appelé est GetObjConnexion de la classe Connexion présentée précédemment. On crée ensuite la collection lesControlesEffect qui contiendra tous les contrôles effectués qui sont présent dans la base de données. Puis, on crée

ensuite l'objet qui contiendra la requête SQL. On appelle ensuite la procédure stockée spObtenirContrôlesEffectues. Voici les instructions permettant cela :

```
// on récupère l'objet responsable de la connexion à la base
SqlConnection cnx = Connexion.GetObjConnexion();

//on crée la collection lesContrôlesEffect de type List<ContrôleEffectue> qui va contenir
//les caractéristiques des contrôles effectués enregistrés dans la base de données
lesContrôlesEffect = new List<ContrôleEffectue>();

// on crée l'objet de type SqlCommand qui va contenir la requête sql
//permettant d'obtenir toutes les caractéristiques de tous les contrôles effectués
maCommand = new SqlCommand();
maCommand.Connection = cnx;
maCommand.CommandType = System.Data.CommandType.StoredProcedure;
maCommand.CommandText = "spObtenirContrôlesEffectues";
```

La procédure stockée spObtenirContrôlesEffectues fait un select sur la base de données. Cette requête retourne pour chaque ligne :

- l'identifiant du contrôle effectué,
- la date du contrôle,
- l'identifiant de la zone de stockage,
- le résumé du contrôle,
- le nom de la zone de stockage,
- le libellé du type de contrôle,
- l'identifiant du type de contrôle,
- le nom de l'entreprise ayant réalisé le contrôle,
- l'identifiant de l'entreprise
- le montant HT facturé pour le contrôle.

Voici cette procédure stockée :

```
-- =====
-- Author:      <Bassette Gwendoline>
-- Create date: <15/10/2019>
-- Description: <Obtenir tous les contrôles effectués >
-- =====
CREATE PROCEDURE [dbo].[spObtenirContrôlesEffectues]
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    select ControleEffectue.id, dateControle, idZoneStockage, resume , ZoneStockage.nom as nomZoneStockage ,
    TypeControle.libelle as libelleTypeControle, idTypeControle, Entreprise.nom as nomEntreprise, idEntreprise, montantHT
    From ControleEffectue
    Join ZoneStockage on idZoneStockage = ZoneStockage.id
    Join TypeControle on idTypeControle = TypeControle.id
    Join Entreprise on idEntreprise = Entreprise.id

END
GO
```

On exécute ensuite la requête SQL en récupérant dans le dataReader monLecteur les enregistrements. Puis pour chaque enregistrement de monLecteur on crée un objet de type ControleEffectue que l'on ajoute ensuite à la collection lesControlesEffectues. On récupère ainsi pour chaque enregistrement retourné les informations dans les variables déclarées précédemment. Voici ces instructions :

```
// on exécute la requête et on récupère dans un DataReader les enregistrements
monLecteur = maCommand.ExecuteReader();
// pour chaque enregistrement du DataReader on crée un objet de
//ControleEffectue que l'on ajoute dans la collection lesControlesEffect
while (monLecteur.Read())
{
    //on récupère les informations du contrôle effectué

    idLu = (int)monLecteur["id"];
    idTypeControleLu = (int)monLecteur["idTypeControle"];
    libelleTypeControle = (string)monLecteur["libelleTypeControle"];
    idZoneStockageLu = (int)monLecteur["idZoneStockage"];
    libelleZoneStockage = (string)monLecteur["nomZoneStockage"];
    idEntrepriseLu = (int)monLecteur["idEntreprise"];
    dateControleLu = (DateTime)monLecteur["dateControle"];
    nomEntrepriseLu = (string)monLecteur["nomEntreprise"];
    resumeLu = (string)monLecteur["resume"];
    montantHTLu = (decimal)monLecteur["montantHT"];
}
```

Il faut ensuite créer des objets de TypeControle, ZoneStockage et Entreprise car ce sont des objets qui sont attendus par le constructeur de ControleEffectue. On crée ainsi les trois objets à partir des identifiants et des libellés retournés par le select. On peut ensuite créer l'objet de ControleEffectue puis l'ajouter à la collection. Voici les instructions réalisant cela :

```
unTypeControle = new TypeControle(idTypeControleLu, libelleTypeControle);
uneZoneStockage = new ZoneStockage(idZoneStockageLu, libelleZoneStockage);
uneEntreprise = new Entreprise(idEntrepriseLu, nomEntrepriseLu);

//on crée une instance de la classe ControleEffectue
unControleEffectue = new ControleEffectue(idLu, unTypeControle, uneZoneStockage, uneEntreprise,
    dateControleLu, resumeLu, montantHTLu);

// On ajoute l'objet à la collection
lesControlesEffect.Add(unControleEffectue);
```

Lorsque tous les enregistrements ont été ajoutés à la collection, on ferme le DataReader, la connexion et on retourne la collection. Voici ces instructions :

```
}
//on ferme le DataReader
monLecteur.Close();
//on ferme la connexion
Connexion.CloseConnection();
//on retourne la collection
return lesControlesEffect;
}
```

Puis, j'ai créé la classe `ControleEffectueManager` dans la bibliothèque de classe `GesStockageBLL`. J'ai d'abord implémenté le design pattern Singleton dans la classe de la même manière que pour la bibliothèque de classe `GesStockageDAL`. Voici ces méthodes :

12 références

```
public class ControleEffectueManager
{
    private static ControleEffectueManager uneInstance;
    //cette méthode crée un objet de la classe ControleEffectueManager s'il n'en existe pas déjà
    //puis retourne la référence à cet objet
    8 références
    public static ControleEffectueManager GetInstance()
    {
        if (uneInstance == null)
        {
            uneInstance = new ControleEffectueManager();
        }
        return uneInstance;
    }
    //le constructeur par défaut est privé : il ne sera donc pas possible de créer un objet à
    //l'extérieur de la classe avec l'instruction new
    1 référence
    private ControleEffectueManager()
    {
    }
}
```

J'ai ensuite ajouté une méthode `GetControlesEffectues()` dans la couche BLL. Cette méthode appelle la couche DAL pour obtenir et retourner la collection de contrôles effectués. Voici cette méthode :

4 références

```
public List<ControleEffectue> GetControlesEffectues()
{
    //ici on peut appliquer des règles métier
    return ControleEffectueDAO.GetInstance().GetControlesEffectues();
}
```

J'ai ensuite modifié la couche GUI qui contient le formulaire pour appeler la méthode ci-dessus de la couche BLL.

J'ai donc ajouté les instructions suivantes qui créent une collection de type `ControleEffectue`, puis qui appellent la couche BLL pour obtenir la liste des contrôles. On indique ensuite que le `dataSource` du `DataGridView` est la collection `lesControlesEffectues`.

1 référence

```
public FormConsultControleEffectue()
{
    InitializeComponent();

    //On crée une liste qui contiendra les controles effectués
    List<ControleEffectue> lesControlesEffectues;
    lesControlesEffectues = new List<ControleEffectue>();

    // On récupère tous les contrôles dans la BD
    lesControlesEffectues = ControleEffectueManager.GetInstance().GetControlesEffectues();

    //Les informations s'affichent dans le datagridview
    dtgconsultContrEffect.DataSource = null;
    dtgconsultContrEffect.DataSource = lesControlesEffectues;
}
```


J'ai également ajouté des instructions permettant de ne pas afficher les colonnes inutiles et de renommer les intitulés des colonnes. Voici ces instructions :

```
dtgconsultContrEffect.Columns[0].Visible = false;
dtgconsultContrEffect.Columns[1].Visible = false;
dtgconsultContrEffect.Columns[2].Visible = false;
dtgconsultContrEffect.Columns[3].Visible = false;

dtgconsultContrEffect.Columns[4].HeaderText = "Date du contrôle";
dtgconsultContrEffect.Columns[5].HeaderText = "Résumé";
dtgconsultContrEffect.Columns[6].HeaderText = "Montant HT facturé";

dtgconsultContrEffect.Columns[7].Visible = false;
dtgconsultContrEffect.Columns[8].Visible = false;
dtgconsultContrEffect.Columns[9].Visible = false;
dtgconsultContrEffect.Columns[10].Visible = false;

dtgconsultContrEffect.Columns[11].HeaderText = "Type de contrôle";
dtgconsultContrEffect.Columns[12].HeaderText = "Zone de Stockage";
dtgconsultContrEffect.Columns[13].HeaderText = "Entreprise";
```

Sur l'extrait du tableau ci-dessous, on peut remarquer que le tableau comporte des colonnes vides qui correspondent à des attributs de la classe ControleEffectuee de la couche BO. Aussi, les noms des colonnes sont les noms donnés aux attributs dans la couche BO. C'est pourquoi j'ai rendu ces colonnes non visibles et que j'ai renommé les en-têtes des colonnes.

Consulter les contrôles effectués

	DateControle	Resume	MontantHT	DateCreation	DateModif	UtilisateurCreat
►	14/08/2019	test risques incen...	300,00			
	17/09/2019	vérification propr...	150,00			
	12/06/2019	vérification	100,00			
	01/01/2020	vérification sécurité	500,00			

J'ai également ajouté des instructions dans la classe ControleEffectuee de la couche BO qui permettent d'obtenir le libellé du type de contrôle réalisé, le nom de l'entreprise et le nom de la zone de stockage correspondants aux objets LeTypeControle, LaZoneStockage et LEntreprise. Voici ces instructions :

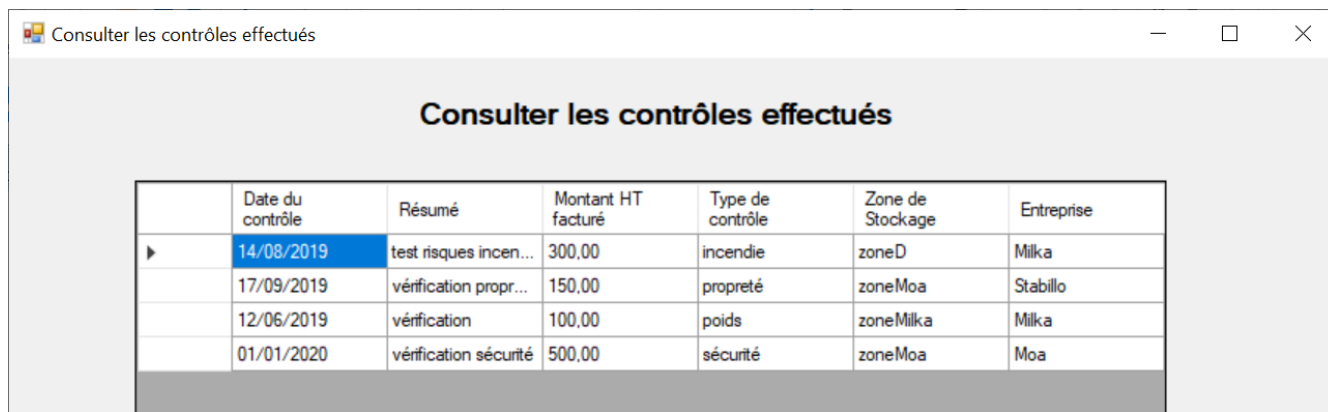
```
0 références
public string Libelle { get => LeTypeControle.Libelle; }
0 références
public string NomZone { get => LaZoneStockage.NomZone; }
0 références
public string NomEntreprise { get => LEntreprise.NomEntreprise; }
```

Sans ces instructions on ne pourrait pas afficher le libellé du type de contrôle réalisé, le nom de l'entreprise et le nom de la zone de stockage. Voici comment serait le tableau :

Consulter les contrôles effectués

	Id	LeTypeControle	LaZoneStockage	LEntreprise	DateControle
►	1	GesStockageBO.TypeControle	GesStockageBO.ZoneStockage	GesStockageBO.Entreprise	14/08/2019
	2	GesStockageBO.TypeControle	GesStockageBO.ZoneStockage	GesStockageBO.Entreprise	17/09/2019
	3	GesStockageBO.TypeControle	GesStockageBO.ZoneStockage	GesStockageBO.Entreprise	12/06/2019
	4	GesStockageBO.TypeControle	GesStockageBO.ZoneStockage	GesStockageBO.Entreprise	01/01/2020

Voici la page qu'obtient l'utilisateur lorsqu'il souhaite consulter les contrôles effectués :



	Date du contrôle	Résumé	Montant HT facturé	Type de contrôle	Zone de Stockage	Entreprise
▶	14/08/2019	test risques incen...	300,00	incendie	zoneD	Milka
	17/09/2019	vérification propr...	150,00	propreté	zoneMoa	Stabillo
	12/06/2019	vérification	100,00	poids	zoneMilka	Milka
	01/01/2020	vérification sécurité	500,00	sécurité	zoneMoa	Moa

2- Ajout d'un contrôle effectué

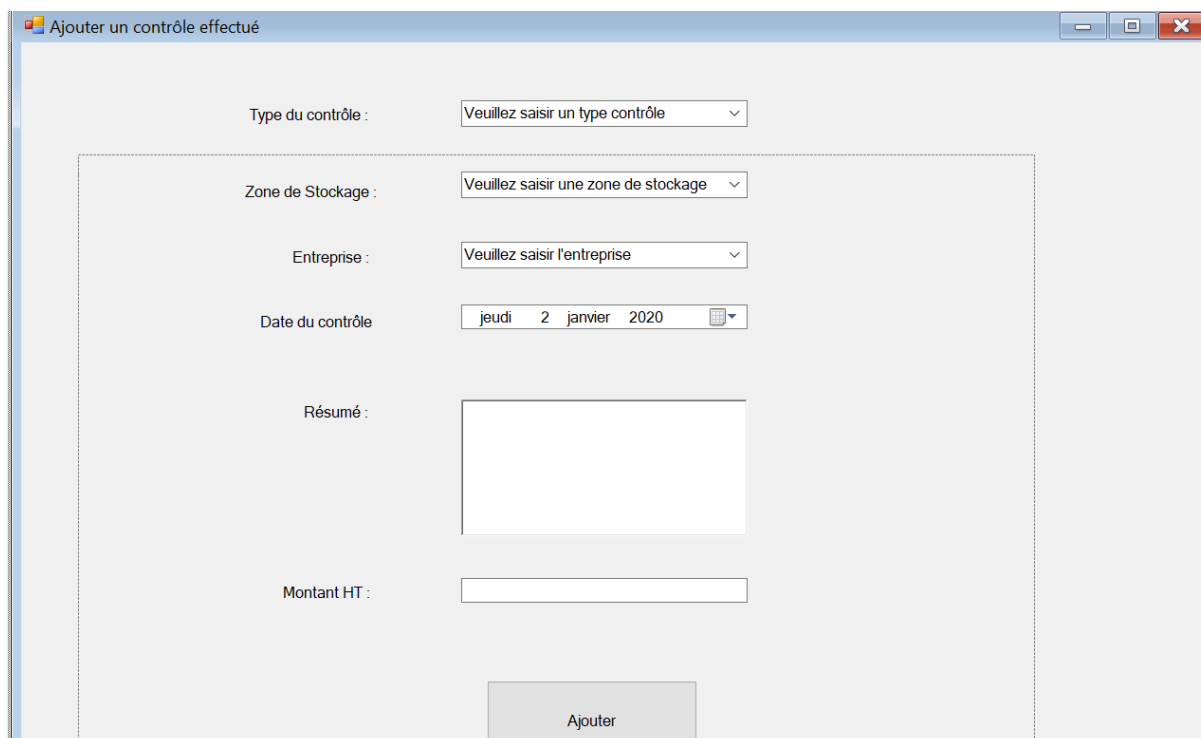
Les contrôles ayant été effectués doivent pouvoir être ajoutés à partir de l'application.

J'ai donc commencé par réaliser le formulaire FormAjoutControleEffectue dans la GUI. Ce formulaire devra contenir :

- une liste déroulante avec tous les types de contrôles possibles
- une liste déroulante avec toutes les zones de stockage
- une liste déroulante avec toutes les entreprises
- un dateTimePicker permettant de choisir une date de contrôle à l'aide d'un calendrier
- une richTextBox pour la saisie du résumé du contrôle
- une textBox pour la saisie du montant hors taxe facturé
- un bouton de validation.

J'ai également ajouté un panel qui englobe tous les éléments sauf la liste déroulante des types de contrôle. De cette façon l'utilisateur choisi le type de contrôle puis les entreprises sont affichées en fonction du type de contrôle. En effet, une entreprise ne réalise que certain type de contrôles il ne faut donc afficher que les entreprises qui réalisent le contrôle sélectionné.

Voici le formulaire :



Type du contrôle :

Zone de Stockage :

Entreprise :

Date du contrôle :

Résumé :

Montant HT :

Ajouter

J'ai ensuite ajouté une nouvelle classe TypeControleDAO dans la bibliothèque de classe GesStockageDAL. J'ai d'abord implémenté le design pattern Singleton dans la classe de la même manière que pour la classe ControleEffectueDAO.

Puis, j'ai créé une méthode GetTypesControles() qui retourne une collection d'objets de type TypeControle. On déclare les variables de travail, on récupère l'objet responsable de la connexion en appelant la méthode GetObjConnexion de la classe Connexion, puis on crée la liste lesTypesControles qui contiendra tous les types de contrôles et qui sera retournée par la méthode. Voici ces instructions :

```
1 référence
public List<TypeControle> GetTypesControle()
{
    //déclaration des variables de travail
    int idLu;
    string libelleLu;
    SqlCommand maCommande;
    maCommande = new SqlCommand();
    SqlDataReader monLecteur;

    //on récupère l'objet responsable de la connexion à la base
    maCommande.Connection = Connexion.GetObjConnexion();

    //on crée la collection lesTypesControles de type list <TypeControle> qui va contenir
    //les caractéristiques des TypeControle enregistrés dans la base de données
    List<TypeControle> lesTypesControles;
    lesTypesControles = new List<TypeControle>();
}
```

On crée ensuite l'objet de type SqlCommand qui contiendra la procédure stockée spObtenirLesTypesDeControle puis on exécute la requête. Voici les instructions réalisant cela :

```
//on crée l'objet de type SqlCommand qui va contenir la requête SQL
//permettant d'obtenir toutes les caractéristiques de tous les types de contrôles
maCommande.CommandType = CommandType.StoredProcedure;
maCommande.CommandText = "spObtenirLesTypesDeControles";
monLecteur = maCommande.ExecuteReader();
```

Voici la procédure stockée spObtenirLesTypesDeControles qui permet d'obtenir l'identifiant et le libellé de tous les types de contrôles présents dans la base de données. Voici cette procédure stockée :

```
-- =====
-- Author:      <Bassette Gwendoline>
-- Create date: <17/10/2019,>
-- Description: <Select de tous les types de contrôle>
-- =====
CREATE PROCEDURE [dbo].[spObtenirLesTypesDeControles]
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT id, libelle
    From TypeControle
END
GO
```

Pour chaque enregistrement retourné par la requête on crée un objet instance de TypeControle qu'on ajoute ensuite à la collection lesTypesControles. On ferme ensuite le dataReader et la connexion avant de retourner la collection. Voici ces instructions :

```
//on exécute la requête et on récupère dans un DataReader
//on crée un objet instance de
//TypeControle que l'on ajoute dans la collection lesTypesControles
while (monLecteur.Read())
{
    // on récupère le type de contrôle

    idLu = (int)monLecteur["id"];
    libelleLu = (string)monLecteur["libelle"];

    // on crée une instance de la classe TypeControle
    TypeControle unTypeControle = new TypeControle(idLu, libelleLu);

    // on ajoute l'instance créée dans la collection
    lesTypesControles.Add(unTypeControle);
}
//on ferme le DataReader
monLecteur.Close();

//on ferme la connexion
Connexion.CloseConnection();

//on retourne la collection
return lesTypesControles;
}
```

J'ai ensuite ajouté une nouvelle classe TypeControleManager dans la bibliothèque de classe GesStockageBLL. J'ai d'abord implémenté le design pattern Singleton dans la classe de la même manière que pour la classe TypeControleDAO. Puis j'ai ajouté une nouvelle méthode GetTypesControle dans laquelle j'appelle la méthode retournant la collection des types de contrôles et je retourne cette collection. Voici cette méthode :

```
3 références
public List<TypeControle> GetTypesControle()
{
    //ici, on peut appliquer des règles métier
    return TypeControleDAO.GetInstance().GetTypesControle();
}
```

J'ai ensuite regardé si des méthodes existaient déjà pour obtenir les informations de la liste déroulante des zones de stockage.

Ainsi j'ai utilisé la méthode GetZonesStock() qui a été réalisée par un autre étudiant. Cette méthode effectue la requête SQL suivante :

```
SELECT ZoneStockage.id, nom, idCategProduit, CategProduit.libelle, idEmplacement, batiment, etage, rue, codePostal,
ville
FROM ZoneStockage
JOIN CategProduit on CategProduit.id=ZoneStockage.idCategProduit
JOIN Emplacement on ZoneStockage.idEmplacement=Emplacement.id
```

Pour chaque enregistrement un objet est créé puis ajouté à la collection de ZoneStockage qui est ensuite retournée.

J'ai aussi utilisé la méthode `getZonesStock` de la classe `ZoneStockageManager` qui appelle la couche DAL afin d'obtenir la collection de zones de stockages. Voici cette méthode :

```
// appel de la couche DAL pour récupérer une collection de zone de stockage
- références
public List<ZoneStockage> GetZonesStock()
{
    // Ici, on peut appliquer des règles métier
    return ZoneStockageDAO.GetInstance().GetZonesStock();
}
```

Il fallait ensuite que je récupère les informations de ma liste déroulante contenant les entreprises. Toutes les entreprises ne doivent pas être affichées car une entreprise n'est habilitée que pour réaliser certains types de contrôles. Par exemple une entreprise peut être habilitée à réaliser des contrôles d'humidité et de sécurité mais pas des contrôles de pression.

J'ai donc réalisé une nouvelle méthode `GetNomEntrepriseParTypeControle` dans la classe `EntrepriseDAO`. Cette méthode permet de retourner une collection avec les noms et identifiant des entreprises qui sont habilitées pour le type de contrôle passé en paramètre. Cette identifiant est récupéré à partir du choix de type contrôle fait dans la liste déroulante.

Ci-dessous je déclare donc mes variables de travail, récupère l'objet responsable de la connexion à la base de données, puis crée la collection qui sera retournée.

```
- références
public List<Entreprise> GetNomEntreprisesParTypeControle(int idTypeContr)
{
    //déclaration des variables de travail
    string nomLu;
    int idLu;
    List<Entreprise> lesEntreprises;

    SqlCommand maCommande;
    maCommande = new SqlCommand();
    SqlDataReader monLecteur;
    //on récupère l'objet responsable de la connexion à la base
    maCommande.Connection = Connexion.GetObjConnexion();

    //on crée la collection lesEntreprises de type list <Entreprise> qui va contenir
    //les caractéristiques des Entreprises enregistrés dans la base de données

    lesEntreprises = new List<Entreprise>();
}
```

Je crée ensuite l'objet qui contiendra la procédure stockée `spObtenirEntrepriseParTypeControle`, sans oublier de déclarer le paramètre `idTypeContr`, puis j'exécute la requête. Voici ces instructions :

```
//on crée l'objet de type SqlCommand qui va contenir la requête SQL
//permettant d'obtenir toutes les caractéristiques de toutes les entreprises
maCommande.CommandType = System.Data.CommandType.StoredProcedure;
maCommande.CommandText = "spObtenirEntrepriseParTypeControle";
maCommande.Parameters.Add("idTypeContr", SqlDbType.Int);
maCommande.Parameters[0].Value = idTypeContr;
monLecteur = maCommande.ExecuteReader();
```

La procédure stockée spObtenirEntrepriseParTypeControle effectue un select afin d'obtenir l'identifiant et le nom des entreprise dont l'idTypeControle est le même que celui passé en paramètre :

```
-- =====
-- Author:      <Bassette Gwendoline>
-- Create date: <01/01/2020>
-- Description: <Obtenir les entreprise par type de contrôle >
-- =====
CREATE PROCEDURE [dbo].[spObtenirEntrepriseParTypeControle] (@idTypeContr int)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    Select id, nom
    from Entreprise
    join EntrepriseTypeControle on idEntreprise = Entreprise.id
    where idTypeControle = @idTypeContr
END
GO
```

Pour chaque enregistrement on crée un objet de type Entreprise qu'on ajoute ensuite à la collection. On ferme ensuite le datareader et la connexion puis on retourne la collection contenant les entreprises. Voici ces instructions :

```
//on execute la requete et on recupere dans un DataReader on cree un objet instance de
//Entreprise que l'on ajoute dans la collection lesEntreprises
while (monLecteur.Read())
{
    // on récupère l'entreprise

    idLu = (int)monLecteur["id"];
    nomLu = (string)monLecteur["nom"];

    // on crée une instance de la classe Entreprise
    Entreprise uneEntreprise = new Entreprise(idLu, nomLu);

    // on ajoute l'instance créée dans la collection
    lesEntreprises.Add(uneEntreprise);
}
//on ferme le DataReader
monLecteur.Close();

//on ferme la connexion
Connexion.CloseConnection();

//on retourne la collection
return lesEntreprises;
```

J'ai ensuite appelé cette méthode dans la classe EntrepriseManager de la couche BLL et retournée la collection des entreprises habilitées pour le type de contrôle sélectionné. Voici cette méthode :

```
- références
public List<Entreprise> GetNomEntreprisesParTypeControle(int idTypeControle)
{
    //ici, on peut appliquer des règles métier
    return EntrepriseDAO.GetInstance().GetNomEntreprisesParTypeControle(idTypeControle);
}
```

J'ai ensuite réalisé la méthode `AjoutControleEffectue` qui fera l'insert du contrôle qui a été saisi par l'utilisateur avec le formulaire.

Dans cette méthode on déclare les variables de travail, puis on récupère l'objet responsable de la connexion à la base de données avant de créer l'objet responsable de la connexion. Les instructions réalisant cela sont les suivantes :

- références

```
public int AjoutControleEffectue(ControleEffectue unControle)
{
    SqlCommand maCommand;
    int idTypeControle;
    int idZoneStockage;
    int idEntreprise;
    DateTime dateControle;
    string resume;
    decimal montant;
    DateTime dateCreation;
    int nbTraite;

    //on récupère l'objet responsable de la connexion à la base de données
    SqlConnection cnx = Connexion.GetObjConnexion();

    // on crée l'objet qui va contenir la requête SQL d'insert qui sera exécuté
    maCommand = new SqlCommand();
    maCommand.Parameters.Clear();
    maCommand.Connection = cnx;
```

On attribut ensuite aux variables les valeurs à partir de l'objet en paramètre passé à la méthode, puis on appelle la procédure stockée en déclarant les paramètres. Voici ces instructions :

```
//on récupère les informations du contrôle effectué
idTypeControle = unControle.LeTypeControle.Id;
idZoneStockage = unControle.LaZoneStockage.Id;
idEntreprise = unControle.LEntreprise.IdEntreprise;
dateControle = unControle.DateControle;
resume = unControle.Resume;
montant = unControle.MontantHT;

dateCreation = DateTime.Now;
maCommand.CommandType = CommandType.StoredProcedure;
//appel de la procédure stockée
maCommand.CommandText = "spAjouteControleEffectue";
//initialisation des paramètres
maCommand.Parameters.Add("idTypeContr", SqlDbType.Int);
maCommand.Parameters[0].Value = idTypeControle;

maCommand.Parameters.Add("idZoneStock", SqlDbType.Int);
maCommand.Parameters[1].Value = idZoneStockage;

maCommand.Parameters.Add("idEntreprise", SqlDbType.Int);
maCommand.Parameters[2].Value = idEntreprise;

maCommand.Parameters.Add("dateContr", SqlDbType.DateTime);
maCommand.Parameters[3].Value = dateControle;

maCommand.Parameters.Add("resume", SqlDbType.VarChar);
maCommand.Parameters[4].Value = resume;

maCommand.Parameters.Add("montantHT", SqlDbType.Decimal);
maCommand.Parameters[5].Value = montant;

maCommand.Parameters.Add("dateCreat", SqlDbType.DateTime);
maCommand.Parameters[6].Value = dateCreation;
```


La procédure stockée `spAjouteControleEffectue` réalise un insert dans la table `ControleEffectue` à partir des informations passées en paramètre :

```
-- =====
-- Author:      <Bassette Chancereul Gwendoline>
-- Create date: <04/11/2019>
-- Description: <Insert d'un controle effectue>
-- =====
CREATE PROCEDURE [dbo].[spAjouteControleEffectue] (@idTypeContr int, @idZoneStock int, @idEntreprise int,
@dateContr date, @resume varchar(200), @montantHT numeric(6,2), @dateCreat date)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    Insert into ControleEffectue values (@idTypeContr, @idZoneStock, @idEntreprise, @dateContr,@resume, @montantHT,
    @dateCreat, null, null, null)
END
GO
```

Enfin, on exécute la requête SQL en récupérant le nombre d'insert traité, on ferme la connexion puis on retourne le nombre d'insert qui ont été traités. En retournant ce nombre cela nous permet d'afficher un message d'erreur à l'utilisateur si l'insert n'a pas été exécuté. Voici ces instructions :

```
// on exécute la requête
nbTraite = maCommand.ExecuteNonQuery();

//on ferme la connexion
Connexion.CloseConnection();
return nbTraite;
}
```

J'ai ensuite ajouté une méthode `CreerControleEffectue` dans la classe `ControleEffectueManager`. Lorsqu'on appelle cette méthode il faut lui passer plusieurs paramètres. Cette méthode vérifie que les valeurs saisies dans le formulaire sont valides et crée une instance de `ControleEffectue` avant d'appeler la couche DAL.

On déclare les variables de travail. Les objets `uneZoneStockage`, `unTypeControle` et `uneEntreprise` permettront de créer l'objet `unControle` utilisé pour l'insert dans la base de données. Voici ces instructions :

```
- références
public int CreerControleEffectue(int unIdTypeContr, int unIdZoneStock, int unIdEntrep,
    DateTime uneDateContr, string unResume, string unMontant, out string msgErreur)
{
    int nombre = 0;
    msgErreur = "";
    ZoneStockage uneZoneStockage;
    TypeControle unTypeControle;
    Entreprise uneEntreprise;
    ControleEffectue unControle;
    decimal unMontantSaisi = 0;
    bool conv;
```


On vérifie qu'un montant a été saisi dans la textBox, si ce n'est pas le cas on prépare le message d'erreur. Si un montant a été saisi on essaie de convertir la valeur saisie en nombre et si ce n'est pas possible on prépare un message d'erreur à l'utilisateur en lui demandant de saisir un nombre. Voici ces instructions :

```
if (unMontant == "" || unMontant == null)
{
    msgErreur = "Veuillez saisir le montant\n";
}
else
{
    conv = decimal.TryParse(unMontant, out unMontantSaisi);
    if (conv == false)
    {
        msgErreur += "Veuillez saisir un nombre";
    }
}
```

On vérifie que le résumé a été saisi et que des valeurs ont été choisies dans les listes déroulantes, si besoin on prépare les messages d'erreurs. On vérifie ensuite que la date de contrôle saisie est inférieure à celle du jour. Voici ces instructions :

```
if (unResume == "" || unResume == null)
{
    msgErreur += "Veuillez saisir un résumé";
}
if (unIdEntrep == -1)
{
    msgErreur += "Veuillez choisir une entreprise";
}
if (unIdTypeContr == -1)
{
    msgErreur += "Veuillez choisir un type de contrôle";
}
if (unIdZoneStock == -1)
{
    msgErreur += "Veuillez choisir une zone de stockage ";
}
if (uneDateContr > DateTime.Now)
{
    msgErreur += "Veuillez saisir une date valide";
}
```

S'il y a un message d'erreur on retourne 0, sinon on crée les objets.

```
if (msgErreur != "")
{
    return 0;
}
else
{
    uneZoneStockage = new ZoneStockage(unIdZoneStock);
    unTypeControle = new TypeControle(unIdTypeContr);
    uneEntreprise = new Entreprise(unIdEntrep);
    unControle = new ControleEffectue(unTypeControle, uneZoneStockage, uneEntreprise,
    uneDateContr, unResume, unMontantSaisi);
```

On essaye ensuite d'appeler la couche DAL qui fait ajouter le contrôle dans la base de données. Si ce n'est pas possible on récupère le message d'erreur puis on retourne le nombre retourné par la méthode `AjoutControleEffectue`. Voici les instructions réalisant cela :

```
try
{
    nombre = ControleEffectueDAO.GetInstance().AjoutControleEffectue(unControle);
}
catch (System.InvalidOperationException err)
{
    //exception généré si problème avec les objets
    msgErreur = "ERREUR Lecture :" + err.Message;
}
catch (Exception err)
{
    //autres exceptions générés
    msgErreur = "ERREUR GRAVE :" + err.Message;
}
return nombre;
}
```

Il est maintenant possible de modifier la couche GUI. J'ai donc appelé la méthode `GetTypesControles` afin d'initialiser la liste déroulante des types de contrôles. Le texte qui est affiché est le libellé du type de contrôle et la valeur est l'identifiant du type de contrôle. Voici cette méthode :

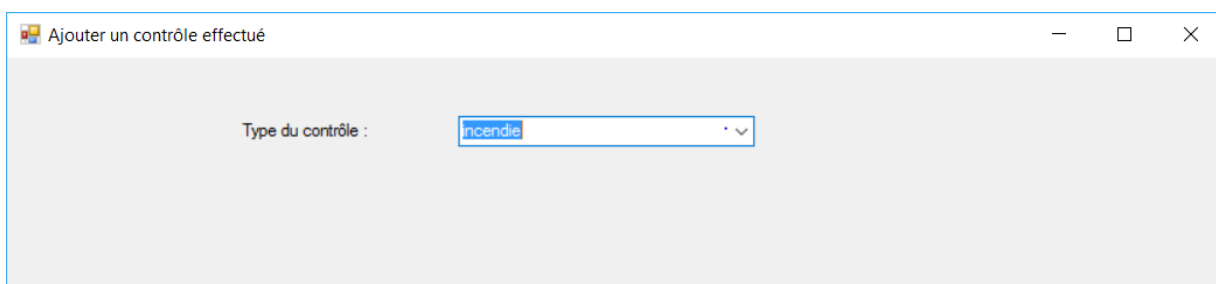
- références

```
public FormAjoutControleEffectue()
{
    InitializeComponent();

    // les valeurs qu'on passe au DisplayMember et au ValueMember sont les nom des attributs
    //dans la classe TypeControle
    cbxTypeControle.DataSource = TypeControleManager.GetInstance().GetTypesControle();

    //On affiche le libellé du type controle
    cbxTypeControle.DisplayMember = "libelle";
    //la valeur de la checkbox est l'id du type controle
    cbxTypeControle.ValueMember = "id";
}
```

Au lancement du formulaire voici la page qui s'affiche :



J'ai ensuite ajouté les instructions exécutées lorsque l'utilisateur sélectionne un type de contrôle. La méthode qui gère cela est `cbxTypeControle_SelectionChangeCommitted()`. Dans cette méthode on rend le panel visible, puis on converti la valeur sélectionnée dans la liste déroulante en nombre avant d'appeler la méthode `GetNomEntrepriseParTypeControle`. Cette méthode présentée précédemment retourne une collection d'entreprises ayant pour `idTypeControle`, l'id du type de contrôle sélectionné dans la liste déroulante. On affiche le nom de l'entreprise dans la liste déroulante et la valeur est l'id de l'entreprise. On appelle ensuite la méthode `GetZonesStock` qui retourne une collection contenant toutes les zones de stockages. On affiche dans la liste déroulante le nom de la zone et la valeur est l'identifiant de la zone de stockage. Les instructions suivantes réalisent cela :

```
private void cbxTypeControle_SelectionChangeCommitted(object sender, EventArgs e)
{
    pnlControleEffectue.Visible = true;
    int idTypeControle;
    int.TryParse(cbxTypeControle.SelectedValue.ToString(), out idTypeControle);
    cbxEntreprise.DataSource = EntrepriseManager.GetInstance().GetNomEntreprisesParTypeControle(idTypeControle)

    // les valeurs qu'on passe au DisplayMember et au ValueMember sont les nom des attributs
    //dans la classe Entreprise

    //On affiche le nom de l'entreprise
    cbxEntreprise.DisplayMember = "nomEntreprise";

    //la valeur de la checkbox est l'id de l'entreprise
    cbxEntreprise.ValueMember = "idEntreprise";

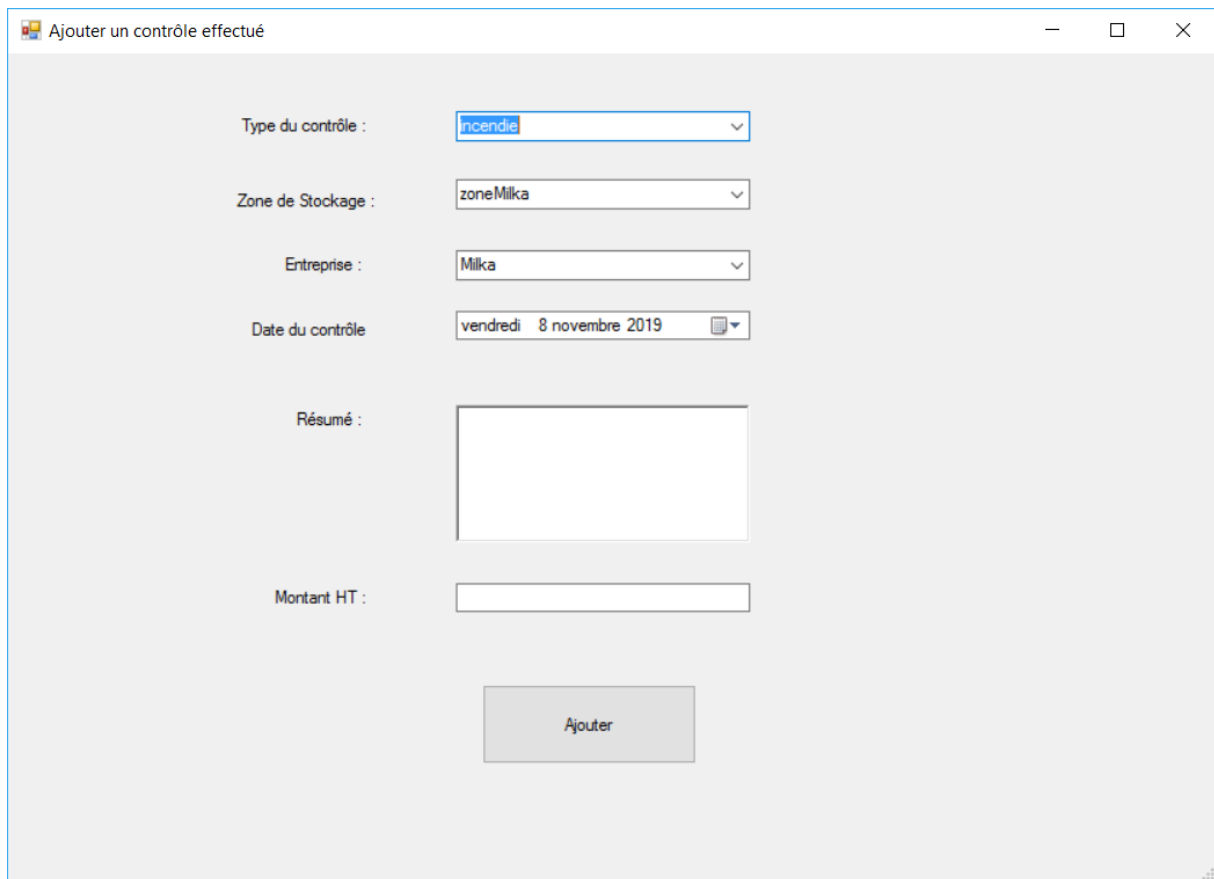
    cbxZoneStockage.DataSource = ZoneStockageManager.GetInstance().GetZonesStock();
    cbxZoneStockage.DisplayMember = "nomZone";
    cbxZoneStockage.ValueMember = "id";
}
```

Enfin on réinitialise la liste déroulante comme dans le constructeur et on sélectionne la valeur qui avait été sélectionné par l'utilisateur.

```
// les valeurs qu'on passe au DisplayMember et au ValueMember sont les nom des attributs
//dans la classe TypeControle
cbxTypeControle.DataSource = TypeControleManager.GetInstance().GetTypesControle();

//On affiche le libellé du type controle
cbxTypeControle.DisplayMember = "libelle";
//la valeur de la checkbox est l'id du type controle
cbxTypeControle.ValueMember = "id";
cbxTypeControle.SelectedValue = idTypeControle;
}
```

Voici le formulaire lorsque le type de contrôle a été sélectionné :



Ajouter un contrôle effectué

Type du contrôle : Incendie

Zone de Stockage : zoneMilka

Entreprise : Milka

Date du contrôle : vendredi 8 novembre 2019

Résumé :

Montant HT :

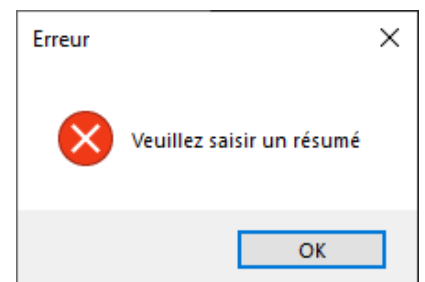
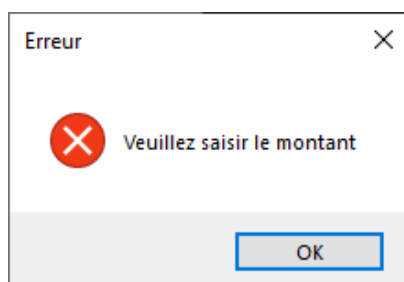
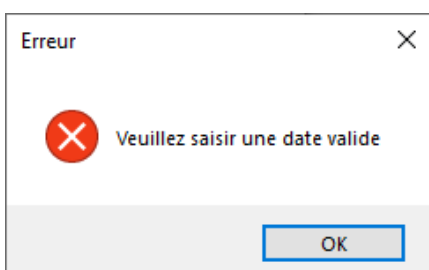
Ajouter

Puis j'ai écrit les instructions exécutées lors du clic sur le bouton Valider. On appelle la méthode `CreerControleEffectue` de la couche BLL qui vérifie que les informations saisies sont valides, puis qui crée un objet qu'il passe à la couche DAL qui fait l'insert dans la base de données. Les informations passées en paramètre sont la valeur du type de contrôle sélectionné, la valeur de la zone de stockage sélectionnée, la valeur de l'entreprise sélectionnée, la date, le résumé et le montant hors taxe saisi. Le message d'erreur est retourné par la méthode `CreerControleEffectue`. S'il y a un message d'erreur on l'affiche à l'aide d'une message box. Voici les instructions qui font cela :

```
private void btnValider_Click(object sender, EventArgs e)
{
    string err = "";
    ControleEffectueManager.GetInstance().CreerControleEffectue((int)cbxTypeControle.SelectedValue,
        (int)cbxZoneStockage.SelectedValue, (int)cbxEntreprise.SelectedValue, dtpDate.Value,
        rtbResume.Text, txtMontantHT.Text, out err);

    if (err != "")
    {
        //On affiche le message d'erreur s'il y en a un
        DialogResult erreur;
        erreur = MessageBox.Show(err, "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Voici les messages d'erreurs pouvant apparaître :

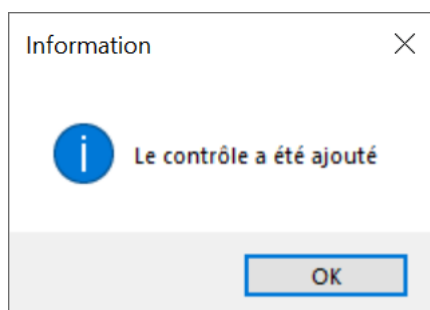


S'il n'y a pas de message d'erreur alors on affiche un message d'information à l'utilisateur pour lui dire que l'ajout a bien été réalisé, on réinitialise ensuite tous les éléments du formulaire. Voici ces instructions :

```
else
{
    //On informe l'utilisateur que le contrôle a été ajouté
    DialogResult info;
    info = MessageBox.Show("Le contrôle a été ajouté", "Information", MessageBoxButtons.OK,
        MessageBoxIcon.Information);

    //On remet tous les élément du formulaire à 0
    cbxEntreprise.SelectedIndex = -1;
    cbxTypeContrôle.SelectedIndex = -1;
    cbxZoneStockage.SelectedIndex = -1;
    txtMontantHT.Text = "";
    rtbResume.Text = "";
    dtpDate.Value = DateTime.Now;
}
}
```

Voici le message qui s'affiche :



3- Modification d'un contrôle effectué

Les utilisateurs doivent également pouvoir modifier un contrôle à partir de l'application. Pour cela j'ai réalisé le formulaire ci-dessous qui est composé d'un dataGridView avec les informations de tous les contrôle comme pour la consultation et d'un formulaire de modification. Lorsque l'utilisateur sélectionne un contrôle dans le tableau le formulaire à droite est prérempli avec les informations du contrôle sélectionné.

	Date du contrôle	Résumé	Montant HT facturé	Type de contrôle	Zone de Stockage	Entreprise
▶	04/11/2019	test sécurité ince...	500,00	incendie	zoneMoa	Mika
	04/11/2019	test température	234,00	propreté	zoneD	3M
	04/11/2019	test risques incen...	344,00	incendie	zoneD	Mika
	05/11/2019	vérification propr...	322,00	propreté	zoneMika	Mika

Type du contrôle :

Zone de Stockage :

Entreprise :

Date du contrôle :

Résumé :

Montant HT :

Pour afficher les informations du contrôle sélectionné, il faut réaliser une méthode dans la couche DAL. J'ai donc ajouté la méthode `GetUnControle` à laquelle on passe en paramètre l'id du contrôle qu'on souhaite obtenir. Cette méthode permet d'obtenir les informations du contrôle qu'on souhaite modifier.

On déclare d'abord les variables de travail :

```
1 référence
public ControleEffectue GetUnControleEffectue(int idControleEffectue
{
    SqlCommand maCommand;
    SqlDataReader monLecteur;

    int idLu = 0;
    TypeControle unTypeControle;
    int idTypeControleLu = 0;
    ZoneStockage uneZoneStockage;
    int idZoneStockageLu = 0;
    Entreprise uneEntreprise;
    int idEntrepriseLu = 0;
    DateTime dateControleLu = DateTime.Now;
    string resumeLu="";
    decimal montantHTLu = 0;
    string libelleZoneStockage="";
    string libelleTypeControle="";
    string nomEntrepriseLu="";
    ControleEffectue unControleEffectue = new ControleEffectue();
```

Puis on récupère l'objet responsable de la connexion à la base de données et on crée l'objet qui va contenir l'appelle de la procédure stockée spObtenirUnControleEffectue en déclarant le paramètre :

```
//on récupère l'objet responsable de la connexion à la base de données
SqlConnection cnx = Connexion.GetObjConnexion();

// on crée l'objet qui va contenir la requête SQL d'insert qui sera exécuté
maCommand = new SqlCommand();
maCommand.Parameters.Clear();
maCommand.Connection = cnx;
maCommand.CommandType = System.Data.CommandType.StoredProcedure;
maCommand.CommandText = "spObtenirUnControleEffectue";

maCommand.Parameters.Add("idControle", SqlDbType.Int);
maCommand.Parameters[0].Value = idControleEffectue;
```

La procédure stockée spObtenirUnControleEffectue réalise un select sur la base de données pour obtenir les informations du contrôle ayant pour identifiant l'idControle passé en paramètre :

```
-- =====
-- Author:      <Bassette Chancereul Gwendoline>
-- Create date: <04/11/2019>
-- Description: <Obtenir un contrôle effectué>
-- =====
CREATE PROCEDURE [dbo].[spObtenirUnControleEffectue] (@idControle int)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT ControleEffectue.id, dateControle, idZoneStockage, resume, ZoneStockage.nom as nomZoneStockage,
    TypeControle.libelle as libelleTypeControle,
    idTypeControle, Entreprise.nom as nomEntreprise, idEntreprise, montantHT
    From ControleEffectue
    Join ZoneStockage on idZoneStockage = ZoneStockage.id
    Join TypeControle on idTypeControle = TypeControle.id
    Join Entreprise on idEntreprise = Entreprise.id
    where ControleEffectue.id = @idControle
END
GO
```

On exécute ensuite la requête et on attribue aux variables la valeur lue avec le datareader :

```
// on exécute la requête et on récupère dans un DataReader les enregistrements
monLecteur = maCommand.ExecuteReader();

while(monLecteur.Read())
{
    //on récupère les informations du contrôle effectué

    idLu = (int)monLecteur["id"];
    idTypeControleLu = (int)monLecteur["idTypeControle"];
    libelleTypeControle = (string)monLecteur["libelleTypeControle"];
    idZoneStockageLu = (int)monLecteur["idZoneStockage"];
    libelleZoneStockage = (string)monLecteur["nomZoneStockage"];
    idEntrepriseLu = (int)monLecteur["idEntreprise"];
    dateControleLu = (DateTime)monLecteur["dateControle"];
    nomEntrepriseLu = (string)monLecteur["nomEntreprise"];
    resumeLu = (string)monLecteur["resume"];
    montantHTLu = (decimal)monLecteur["montantHT"];
```

On crée les objets unTypeControle, uneZoneStockage et uneEntreprise avant de créer l'instance de la classe ControleEffectue. On ferme ensuite la connexion et le datareader avant de retourner l'objet ContrôleEffectue. Voici ces instructions :

```
unTypeControle = new TypeControle(idTypeControleLu, libelleTypeControle);
uneZoneStockage = new ZoneStockage(idZoneStockageLu, libelleZoneStockage);
uneEntreprise = new Entreprise(idEntrepriseLu, nomEntrepriseLu);
//on crée une instance de la classe ControleEffectue
unControleEffectue = new ControleEffectue(idLu, unTypeControle, uneZoneStockage, uneEntreprise,
    dateControleLu, resumeLu, montantHTLu);

}
//on ferme le DataReader
monLecteur.Close();
//on ferme la connexion
Connexion.CloseConnection();
return unControleEffectue;
}
```

J'ai ensuite ajouté une méthode GetUnControleEffectue dans la classe ControleEffectueManager qui appelle la couche DAL. Voici cette méthode :

```
1 référence
public ControleEffectue GetUnControleEffectue(int idControle)
{
    return ControleEffectueDAO.GetInstance().GetUnControleEffectue(idControle);
}
```

J'ai ainsi pu modifier les instructions du formulaire pour l'initialiser avec les informations du contrôle sélectionné. Ainsi on transforme en chaîne de caractère la valeur de la ligne sélectionné qui correspond à l'identifiant du contrôle. On le transforme ensuite en entier puis on appelle la méthode GetUnContrôleEffectue de la couche BLL qui retourne l'instance de ControleEffectue avec les informations du contrôle sélectionné. On initialise le résumé le montant hors taxe et la date du contrôle avec ces valeurs. Voici ces instructions :

```
private void dtgconsultContrEffect_Click(object sender, EventArgs e)
{
    //ControleEffectue leControleEffectue;
    string convert;
    int idSelect;

    //On récupère l'id du controle sélectionné
    convert = ((dtgconsultContrEffect.SelectedRows[0]).Cells[0].Value.ToString());
    int.TryParse(convert, out idSelect);

    //On récupère les informations du client sélectionné
    leControleEffectue = ControleEffectueManager.GetInstance().GetUnControleEffectue(idSelect);

    //Les textbox sont préremplis avec le montant et le résumé présent dans la BD
    txtMontantHT.Text = leControleEffectue.MontantHT.ToString();
    rtbResume.Text = leControleEffectue.Resume.ToString();
    dtpDate.Text = leControleEffectue.DateControle.ToString();
    //On récupère l'id du Controle
    idControleModif = idSelect;
}
```


On appelle ensuite les méthodes permettant d'initialiser les listes déroulantes de la même manière que pour le formulaire d'ajout, puis pour chaque formulaire on sélectionne la valeur correspondant aux informations du contrôle sélectionné :

```
//On remplit la liste déroulante avec les libellés présents dans la BD
cbxTypeControle.DataSource = TypeControleManager.GetInstance().GetTypesControle();
cbxTypeControle.DisplayMember = "libelle";
cbxTypeControle.ValueMember = "id";

cbxZoneStockage.DataSource = ZoneStockageManager.GetInstance().GetZonesStock();
cbxZoneStockage.DisplayMember = "nomZone";
cbxZoneStockage.ValueMember = "id";

cbxEntreprise.DataSource = EntrepriseManager.GetInstance().GetNomEntreprisesParTypeControle(leControleEffectue.LeTypeControle.Id);
cbxEntreprise.DisplayMember = "nomEntreprise";
cbxEntreprise.ValueMember = "idEntreprise";

cbxTypeControle.SelectedValue = leControleEffectue.LeTypeControle.Id;
//cbxEntreprise.SelectedValue = leControleEffectue.LEntreprise.IdEntreprise;
cbxZoneStockage.SelectedValue = leControleEffectue.LaZoneStockage.Id;
```

On obtient ainsi le formulaire ci-dessous, à gauche la ligne en bleu est celle du contrôle sélectionné et à droite le formulaire est pré-rempli avec les informations du contrôle :

Date du contrôle	Résumé	Montant HT facturé	Type de contrôle	Zone de Stockage	Entreprise
04/11/2019	test sécurité ince...	500,00	incendie	zoneMoa	Mika
04/11/2019	test température	234,00	propreté	zoneD	3M
04/11/2019	test risques incen...	344,00	incendie	zoneD	Mika
05/11/2019	vérification propr...	322,00	propreté	zoneMika	Mika

Type du contrôle :

Zone de Stockage :

Entreprise :

Date du contrôle :

Résumé :

Montant HT :

J'ai ensuite réalisé la méthode permettant de mettre à jour les informations dans la base de données. J'ai ajouté la méthode `ModifieUnControleEffectue` qui réalise l'update des informations dans la base de données à partir de l'instance de `ControleEffectue` passé en paramètre.

On déclare les variables puis on récupère l'objet responsable de la connexion à la base de données.

```
public int ModifieUnControleEffectue(ControleEffectue unControle)
{
    SqlCommand maCommand;
    int nbTraite;
    int idContr;
    int idTypeContr;
    int idZoneStock;
    int idEntrepr;
    string resume;
    decimal montantHT;
    DateTime dateContr;
    DateTime dateModification;
    //Utilisateur idUtilisatModif
    //on récupère l'objet responsable de la connexion à la base de données
    SqlConnection cnx = Connexion.GetObjConnexion();
```

On crée ensuite l'objet responsable de la connexion à la base de données et les informations du contrôle qui a été passé à la méthode en paramètre, la date de modification est la date du jour :

```
// on crée l'objet qui va contenir la requête SQL d'insert qui sera exécuté
```

```
maCommand = new SqlCommand();
maCommand.Parameters.Clear();
maCommand.Connection = cnx;

//on récupère les informations du controle
idContr = unControle.Id;
idTypeContr = unControle.LeTypeControle.Id;
idZoneStock = unControle.LaZoneStockage.Id;
idEntrepr = unControle.LEntreprise.IdEntreprise;
resume = unControle.Resume;
montantHT = unControle.MontantHT;
dateContr = unControle.DateControle;
dateModification = DateTime.Now;
```

On appelle la procédure stockée spModifierUnControleEffectue en déclarant les paramètres. On récupère le nombre de lignes traitées lorsqu'on exécute la requête SQL puis on ferme la connexion avant de retourner ce nombre. Voici ces instructions :

```
maCommand.CommandType = System.Data.CommandType.StoredProcedure;
//appel de la procédure stockée
maCommand.CommandText = "spModifierUnControleEffectue";

maCommand.Parameters.Add("idTypeControle", SqlDbType.Int);
maCommand.Parameters[0].Value = idTypeContr;

maCommand.Parameters.Add("idZoneStockage", SqlDbType.Int);
maCommand.Parameters[1].Value = idZoneStock;

maCommand.Parameters.Add("idEntreprise", SqlDbType.Int);
maCommand.Parameters[2].Value = idEntrepr;

maCommand.Parameters.Add("dateControle", SqlDbType.DateTime);
maCommand.Parameters[3].Value = dateContr;

maCommand.Parameters.Add("resume", SqlDbType.VarChar);
maCommand.Parameters[4].Value = resume;

maCommand.Parameters.Add("montantHT", SqlDbType.Decimal);
maCommand.Parameters[5].Value = montantHT;

maCommand.Parameters.Add("dateModif", SqlDbType.DateTime);
maCommand.Parameters[6].Value = dateModification;

maCommand.Parameters.Add("idControle", SqlDbType.Int);
maCommand.Parameters[7].Value = idContr;

// on exécute la requête
nbTraite = maCommand.ExecuteNonQuery();

//on ferme la connexion
Connexion.CloseConnection();
return nbTraite;
}
```

La procédure stockée spModifieUnControleEffectue réalise un update de la table ControleEffectue avec les informations passées en paramètre et dont l'identifiant du contrôle est identique à l'identifiant passé en paramètre :

```
-- =====
-- Author:      <Bassette Chancereul Gwendoline>
-- Create date: <04/11/2019>
-- Description: <Obtenir un contrôle effectué>
-- =====
CREATE PROCEDURE [dbo].[spModifierUnControleEffectue] (@idTypeControle int, @idZoneStockage int,
@idEntreprise int, @dateControle date, @resume varchar(200), @montantHT decimal(6,2), @dateModif date, @idControle int )
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    update ControleEffectue set idTypeControle = @idTypeControle, idZoneStockage = @idZoneStockage,
    idEntreprise = @idEntreprise, dateControle = @dateControle, resume = @resume,
    montantHT = @montantHT, dateModif = @dateModif
    where id = @idControle
END
GO
```

J'ai ensuite réalisé la méthode ModifierUnContrôle dans la couche BLL. On déclare les variables puis on vérifie les informations saisies comme pour l'ajout. Voici ces instructions :

```
1 référence
public int ModifierUnContrôle(int id, int unIdTypeContrôle, int unIdZoneStockage, int unIdEntreprise,
    DateTime uneDateContrôle, string unResume, string unMontantHT, out string msgErr )
{
    int nombre = 0;
    msgErr = "";
    decimal unMontantSaisi = 0;
    bool conv;
    ControleEffectue leControleEffectue;
    TypeContrôle leTypeContrôle;
    ZoneStockage laZoneStockage;
    Entreprise lEntreprise;

    if (unMontantHT == "" || unMontantHT == null)
    {
        msgErr = "Veuillez saisir le montant";
    }
    else
    {
        conv = decimal.TryParse(unMontantHT, out unMontantSaisi);
        if (conv == false)
        {
            msgErr += "\nVeuillez saisir un nombre";
        }
    }

    if (unResume == "" || unResume == null)
    {
        msgErr += "\nVeuillez saisir un résumé";
    }
    if (unIdEntreprise == -1)
    {
        msgErr += "\nVeuillez choisir une entreprise";
    }
    if (unIdTypeContrôle == -1)
    {
        msgErr += "\nVeuillez choisir un type de contrôle";
    }
    if (unIdZoneStockage == -1)
    {
        msgErr += "\nVeuillez choisir une zone de stockage ";
    }
    if (uneDateContrôle > DateTime.Now)
    {
        msgErr += "\nVeuillez saisir une date valide";
    }
    if (msgErr != "")
    {
        return 0;
    }
}
```

S'il y a un message d'erreur on retourne 0, sinon on crée les objets comme pour l'ajout d'un contrôle.

```
if (msgErr != "")
{
    return 0;
}
else
{
    //sinon on crée le controle dans la BD
    leTypeContrôle = new TypeContrôle(unIdTypeContrôle);
    laZoneStockage = new ZoneStockage(unIdZoneStockage);
    lEntreprise = new Entreprise(unIdEntreprise);
    leControleEffectue = new ControleEffectue(id, leTypeContrôle, laZoneStockage, lEntreprise,
        uneDateContrôle, unResume, unMontantSaisi);
}
```

On appelle ensuite la couche DAL pour réaliser l'update. S'il y a une erreur on l'ajoute à la variable contenant tous les messages d'erreur. Voici les instructions réalisant cela :

```
try
{
    nombre = ControleEffectueDAO.GetInstance().ModifieUnControleEffectue(leControleEffectue);
}

catch (System.InvalidOperationException err)
{
    //exception généré si problème avec les objets
    msgErr = "ERREUR Lecture :" + err.Message;
}
catch (Exception err)
{
    //autre exceptions générés
    msgErr = "ERREUR GRAVE :" + err.Message;
}
return nombre;
}
```

Il est maintenant possible de modifier le formulaire de modification pour que les modifications soient prises en compte.

On appelle donc la méthode ModifierUnControle de la couche BLL en passant les informations saisies en paramètre. S'il y a un message d'erreur alors on affiche une message box pour informer l'utilisateur d'une erreur de saisie ou d'une erreur lors de l'update. La méthode faisant cela est la suivante :

```
private void btnValider_Click(object sender, EventArgs e)
{
    string err = "";
    ControleEffectueManager.GetInstance().ModifierUnControle(idControleModif,
        (int)cbxTypeControle.SelectedValue, (int)cbxZoneStockage.SelectedValue,
        (int)cbxEntreprise.SelectedValue, dtpDate.Value, rtbResume.Text, txtMontantHT.Text,
        out err);
    if (err != "")
    {
        DialogResult erreur;
        erreur = MessageBox.Show(err, "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

On met ensuite à jour le dataGridView avec les informations mises à jour en appelant la méthode GetControlesEffectues et en initialisant à nouveau le tableau comme ceci :

```

    ,
    // On récupère tous les controles dans la BD et on met à jour le datagridview
    lesControlesEffectues = ControleEffectueManager.GetInstance().GetControlesEffectues();

    //Les informations s'affichent dans le datagridview
    dtgconsultContrEffect.DataSource = null;
    dtgconsultContrEffect.DataSource = lesControlesEffectues;
    dtgconsultContrEffect.Columns[0].Visible = false;
    dtgconsultContrEffect.Columns[1].Visible = false;
    dtgconsultContrEffect.Columns[2].Visible = false;
    dtgconsultContrEffect.Columns[3].Visible = false;

    dtgconsultContrEffect.Columns[4].HeaderText = "Date du contrôle";
    dtgconsultContrEffect.Columns[5].HeaderText = "Résumé";
    dtgconsultContrEffect.Columns[6].HeaderText = "Montant HT facturé";

    dtgconsultContrEffect.Columns[7].Visible = false;
    dtgconsultContrEffect.Columns[8].Visible = false;
    dtgconsultContrEffect.Columns[9].Visible = false;
    dtgconsultContrEffect.Columns[10].Visible = false;

    dtgconsultContrEffect.Columns[11].HeaderText = "Type de contrôle";
    dtgconsultContrEffect.Columns[12].HeaderText = "Zone de Stockage";
    dtgconsultContrEffect.Columns[13].HeaderText = "Entreprise";
}

```

4- Suppression d'un contrôle effectué

Les informations d'un contrôle doivent pouvoir être supprimées à partir de l'application. Pour cela j'ai mis la fonctionnalité de suppression sur la page de modification. L'utilisateur sélectionne le contrôle dans le tableau et clique sur le bouton « Supprimer » entouré en rouge :

Date du contrôle	Résumé	Montant HT facturé	Type de contrôle	Zone de Stockage	Entreprise
04/11/2019	test sécurité ince...	500,00	incendie	zoneMoa	Mika
04/11/2019	test température	234,00	propreté	zoneD	3M
04/11/2019	test risques incen...	344,00	incendie	zoneD	Mika
05/11/2019	vérification propr...	322,00	propreté	zoneMika	Mika

Type du contrôle :

Zone de Stockage :

Entreprise :

Date du contrôle :

Résumé :

Montant HT :

Pour réaliser la suppression j'ai ajouté une méthode `SupprControleEffectue` qui récupère l'objet responsable de la connexion, crée l'objet qui va contenir la procédure stockée, on appelle la procédure stockée en déclarant le paramètre qui est l'identifiant du contrôle sélectionné. Voici ces instructions :

- références

```
public int SupprControleEffectue (int idControle)
{
    SqlCommand maCommand;
    int nbTraite;

    //on récupère l'objet responsable de la connexion à la base de données
    SqlConnection cnx = Connexion.GetObjConnexion();

    // on crée l'objet qui va contenir la requête SQL de delete qui sera exécutée
    maCommand = new SqlCommand();
    maCommand.Parameters.Clear();
    maCommand.Connection = cnx;
    maCommand.CommandType = System.Data.CommandType.StoredProcedure;

    //appel de la procédure stockée
    maCommand.CommandText = "spSupprimeControleEffectue";

    maCommand.Parameters.Add("idControle", SqlDbType.Int);
    maCommand.Parameters[0].Value = idControle;
```

La procédure stockée `spSupprimeControleEffectue` réalise un delete dans la table `ControleEffectue` pour l'enregistrement dont l'identifiant est identique à celui passé en paramètre :

```
-- =====
-- Author:      <Bassette Chancereul Gwendoline>
-- Create date: <05/11/2019>
-- Description: <Supprimer un controle>
-- =====
CREATE PROCEDURE [dbo].[spSupprimeControleEffectue] (@idControle int)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    delete from ControleEffectue where id = @idControle
END
GO
```

On exécute ensuite la requête SQL en récupérant le nombre d'enregistrements traités, on ferme la connexion puis on retourne ce nombre :

```
// on exécute la requête
nbTraite = maCommand.ExecuteNonQuery();

//on ferme la connexion
Connexion.CloseConnection();
return nbTraite;
}
```

J'ai ensuite appelé cette méthode dans la couche BLL :

```
1 référence
public int SupprControleEffectue (int idControle)
{
    return ControleEffectueDAO.GetInstance().SupprControleEffectue(idControle);
}
```

On peut ensuite modifier la couche GUI pour que lors du clic sur le bouton « supprimer » la méthode de la classe `ControleEffectueManager` soit appelé. On appelle ensuite la méthode qui permet d'obtenir tous les contrôles effectués.

```
private void btn_supprimer_Click(object sender, EventArgs e)
{
    ControleEffectueManager.GetInstance().SupprControleEffectue(idControleModif);
    // On récupère tous les controles dans la BD et on met à jour le datagridview
    lesControlesEffectues = ControleEffectueManager.GetInstance().GetControlesEffectues();
}
```


Puis on met à jour le dataGridView avec la collection de contrôles effectués mise à jour :

```
//Les informations s'affichent dans le datagridview
dtgconsultContrEffect.DataSource = null;
dtgconsultContrEffect.DataSource = lesContrôlesEffectues;
dtgconsultContrEffect.Columns[0].Visible = false;
dtgconsultContrEffect.Columns[1].Visible = false;
dtgconsultContrEffect.Columns[2].Visible = false;
dtgconsultContrEffect.Columns[3].Visible = false;

dtgconsultContrEffect.Columns[4].HeaderText = "Date du contrôle";
dtgconsultContrEffect.Columns[5].HeaderText = "Résumé";
dtgconsultContrEffect.Columns[6].HeaderText = "Montant HT facturé";

dtgconsultContrEffect.Columns[7].Visible = false;
dtgconsultContrEffect.Columns[8].Visible = false;
dtgconsultContrEffect.Columns[9].Visible = false;
dtgconsultContrEffect.Columns[10].Visible = false;

dtgconsultContrEffect.Columns[11].HeaderText = "Type de contrôle";
dtgconsultContrEffect.Columns[12].HeaderText = "Zone de Stockage";
dtgconsultContrEffect.Columns[13].HeaderText = "Entreprise";
```

Le contrôle est ainsi supprimé et n'apparaît plus dans la liste.