Gestionnaire des droits d'utilisateurs des applications

05/01/2026 – 06/02/2026

GUEREKOYAZANDE MALHON

Sommaire

1 Expression des besoins

1.1 Contexte, domaine, processus métier

Présentation d'Aperam Gueugnon

Aperam Gueugnon est un site industriel de référence situé en Saône-et-Loire, appartenant à la division *Aperam Stainless France*. Véritable poids lourd de l'économie locale, l'entreprise emploie environ **660 collaborateurs** et contribue activement à la performance financière d'un groupe réalisant plusieurs milliards d'euros de chiffre d'affaires.

Spécialisée dans la transformation d'acier inoxydable, l'usine se distingue par son expertise de pointe dans le **laminage à froid** et le **recuit brillant**. Elle transforme la matière brute en produits plats à haute valeur ajoutée, avec une capacité de production annuelle avoisinant les **400 000 tonnes**. Ces aciers d'exception alimentent des marchés exigeants tels que l'automobile, l'électroménager et l'industrie. Le site est également un partenaire privilégié du secteur du bâtiment, son savoir-faire s'illustrant notamment sur des façades prestigieuses comme celle du **Musée des Confluences à Lyon**.

Contexte du projet de stage

Ce projet a pour but de structurer et de fiabiliser la gestion des habilitations en remplaçant un fonctionnement actuel fragmenté par une véritable architecture base de données relationnelle. Aujourd'hui, les informations concernant les utilisateurs et les applications ne sont pas gérées via un système dynamique unifié, mais proviennent de sources disparates, notamment transmises sous forme de fichiers plats (Excel). L'objectif est donc d'importer et de consolider ces données au sein d'une structure pérenne organisée autour de **Groupes d'Utilisateurs**. Cette centralisation permettra au service informatique de passer d'une gestion de données statiques à un pilotage dynamique des droits sur les solutions et les écrans, sans dépendre de traitements manuels ou de fichiers externes.

1.2 Demandeur, acteurs, utilisateurs

Le demandeur est le service informatique, qui souhaite reprendre le contrôle total sur la gouvernance des données et la sécurité applicative. Ce service sera le seul habilité à manipuler, via la future interface, l'attribution des droits sur les solutions et les écrans. Les acteurs du système sont donc les administrateurs IT, qui configureront la matrice des droits à partir des données consolidées, et les utilisateurs finaux, dont l'accès aux fonctionnalités ne dépendra plus de fichiers statiques, mais sera déterminé en temps réel par leur appartenance aux nouveaux groupes d'utilisateurs définis en base de données.

1.3 Etude de l'existant, diagnostic

Le système en place dispose d'une gestion de droits, mais celle-ci s'avère limitée et peu évolutive en raison de l'absence d'une base de données dédiée et structurée pour les habilitations. Les données référentielles des utilisateurs et des applications ne sont pas interconnectées en temps réel ; elles sont actuellement figées et issues d'imports statiques (type Excel), ce qui rend la traçabilité et la mise à jour des accès complexes. Ce fonctionnement en "silos" de données empêche toute modulation fine et réactive des privilèges. Le diagnostic met en évidence la nécessité impérative de sortir de cette logique de fichiers statiques pour construire un modèle de données relationnel capable d'interpréter dynamiquement les règles d'accès définies par le service informatique.

1.4 Description de la demande, objectifs, bénéfices attendus

La demande émane du service informatique qui souhaite abandonner la gestion obsolète des accès basée sur des fichiers statiques (Excel) et du code figé pour passer à une véritable architecture dynamique. L'objectif principal est de migrer vers un modèle relationnel centralisé où les droits sont gérés par "Groupes d'Utilisateurs", permettant de définir précisément qui a accès à quelle Solution et à quel Écran. Les bénéfices attendus sont une sécurisation accrue des données, une autonomie totale du service informatique pour modifier les droits sans intervention dans le code source, et une suppression des incohérences liées aux sources de données disparates actuelles.

1.5 Spécifications fonctionnelles

Afin de répondre aux objectifs de centralisation et de sécurisation, les fonctionnalités du projet ont été définies sous forme de récits utilisateurs (User Stories) :

- **User story n°1 :** En tant qu'Administrateur IT, je veux pouvoir créer, modifier et supprimer des "Groupes d'Utilisateurs", afin de structurer les rôles et faciliter l'attribution des droits en masse.

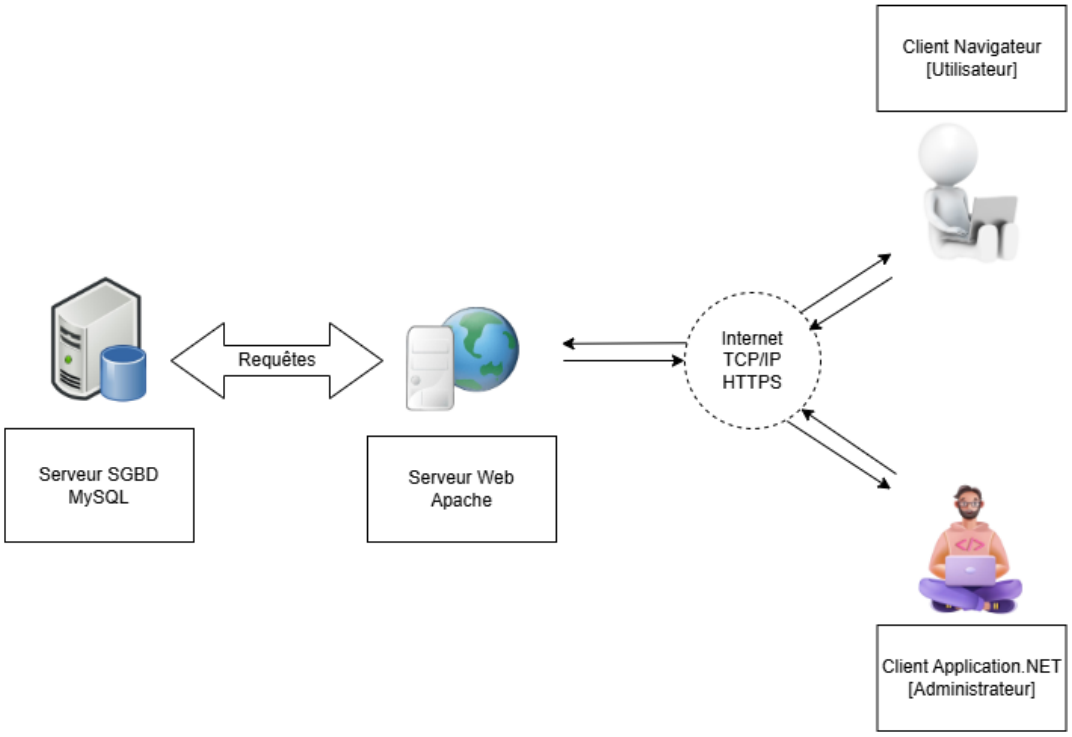
- **User story n°2 :** En tant qu'Administrateur IT, je veux gérer les permissions fonctionnalité par fonctionnalité (écrans, boutons, menus), afin de contrôler finement les accès au sein des solutions.

- **User story n°3 :** En tant qu'Administrateur IT, je veux associer directement des profils individuels à des groupes définis, afin d'automatiser la gestion des droits sans devoir traiter chaque utilisateur manuellement.

- **User story n°4 :** En tant qu'Administrateur IT, je veux que l'interface de mon application s'adapte dynamiquement (affichage/masquage d'éléments) à mon démarrage, afin de n'afficher que les fonctions autorisées par mon groupe.

2 Conception, Spécifications Techniques

2.1 Architecture matérielle et logicielle de la solution (schémas)



2.2 Outils logiciels de la solution

- **Schéma d'architecture de la solution :** Le document contient un schéma d'architecture qui représente les interactions entre un serveur de gestion de base de données de type SQL Server, un serveur Web Apache et des clients (application .NET) via Internet (TCP/IP HTTPS).

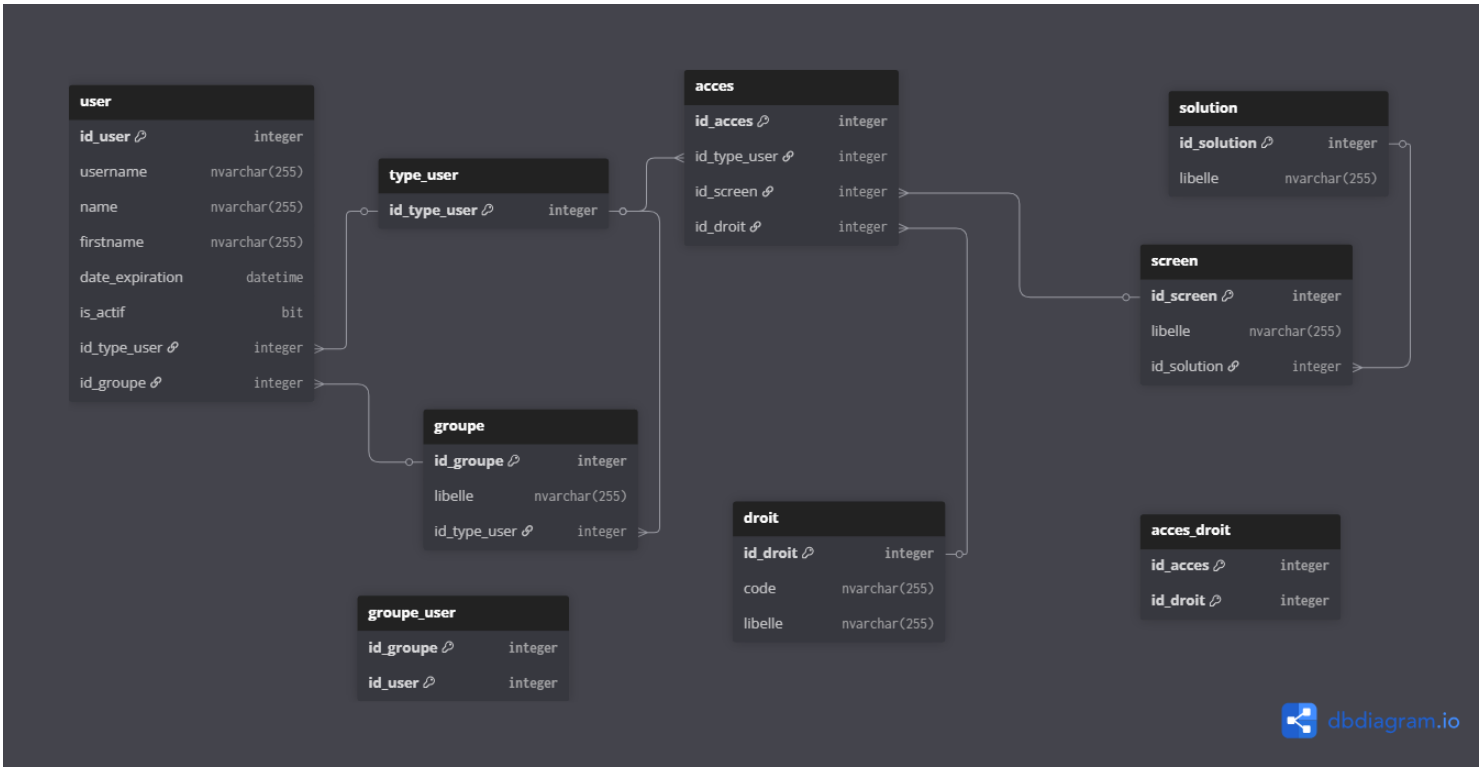
- **Composants logiciels clés :**

    - **Visual Basic.NET :** Langage de développement utilisé pour créer l'outil.

    - **Microsoft Visual Studio :** L'environnement de développement intégré (IDE) pour le développement de l'application.

    - **SQL Server :** Le système de gestion de base de données utilisé est de type SQL Server.

2.5 Analyse des données (modélisation, diagramme de classes, schéma relationnel)



[Attirez l'attention du lecteur avec une citation du document ou utilisez cet espace pour mettre en valeur un point clé. Pour placer cette zone de texte n'importe où sur la page, faites-la simplement glisser.]

Tables	Champs
Table user : La table user est le référentiel central des identités numériques. Elle stocke les informations de connexion et l'état administratif de chaque collaborateur.	**id_user** : L'identifiant unique de l'utilisateur (Clé primaire). **username** : L'identifiant de connexion (login) utilisé pour l'authentification (ex: abaptiste, adminlauncher). **name** : Le nom de famille de l'utilisateur. **firstname** : Le prénom de l'utilisateur. **date_expiration** : La date à partir de laquelle le compte n'est plus valide (sécurité temporelle). **is_actif** : Un indicateur booléen (1 ou 0) permettant d'activer ou de désactiver un compte sans le supprimer physiquement. **id_type_user** : Une clé étrangère liant l'utilisateur à un type spécifique (rôle technique ou métier). **id_groupe** : Une clé étrangère liant l'utilisateur à un groupe principal.
Table groupe : La table groupe permet de rassembler des utilisateurs sous une entité logique (département, équipe, rôle générique) pour faciliter l'attribution des droits.	**id_groupe** : L'identifiant unique du groupe (Clé primaire). **libelle** : Le nom désignant le groupe. **id_type_user** : Une référence vers un type d'utilisateur associé à ce groupe.
Table type_user : La table type_user définit des catégories ou profils d'utilisateurs. Elle sert de d'héritage pour la définition des accès dans la table acces.	**id_type_user** : L'identifiant unique du type d'utilisateur (Clé primaire).
Table solution La table solution répertorie l'ensemble des applications ou suites logicielles gérées par le système (ex: ToolBox IT, Gestion du Personnel, Laminage Consignes).	**id_solution** : L'identifiant unique de la solution (Clé primaire). **libelle** : Le nom commercial ou technique de l'application.
Table screen : La table screen représente les vues, fenêtres ou modules spécifiques à l'intérieur d'une solution. Elle permet une granularité fine des droits (accès à un écran précis plutôt qu'à toute l'application).	**id_screen** : L'identifiant unique de l'écran (Clé primaire). **libelle** : Le nom de l'écran ou du module (ex: DIF, Gestion du Personnel). **id_solution** : La clé étrangère indiquant à quelle solution logicielle cet écran appartient.

Table acces : La table acces est la table de jointure principale qui croise toutes les données pour définir une permission. Elle répond à la question : "Quel type d'utilisateur a quel droit sur quel écran de quelle solution ?".	**id_acces** : L'identifiant unique de la règle d'accès (Clé primaire). **id_type_user** : Le profil utilisateur concerné par la permission. **id_solution** : L'application concernée. **id_screen** : L'écran spécifique concerné. **id_droit** : Le droit accordé (ex: Lecture) pour cette combinaison.
Table groupe_user (Table de réserve) : Cette table a pour vocation de gérer la cardinalité "Many-to-Many" (N-N). Un groupe possède plusieurs utilisateur. Un utilisateur peut appartenir à plusieurs groupes	**id_groupe** : L'identifiant unique du groupe (Clé primaire et étrangère). **id_user** : L'identifiant unique de l'utilisateur (Clé primaire et étrangère). La gestion actuelle se basant sur un groupe unique par utilisateur (via la table user), cette table est conservée vide et déconnectée pour le moment.
Table acces_droit (Table de réserve) : Cette table permettrait de dissocier la définition de l'accès (le triplet Profil/Solution/Écran) des droits qui lui sont attachés, permettant de cumuler plusieurs droits sur un même accès sans redondance.	**id_acces** : L'identifiant unique de la règle d'accès (Clé primaire). **id_droit** : Le droit accordé (ex: Lecture) pour cette combinaison.

Les tables qui sont en réserve ont été créées dans la base de données pour garantir l'évolutivité du projet, mais **elles ne sont pas utilisées dans la logique de gestion actuelle**. **Note technique :** Pour cette première itération, les contraintes d'intégrité (clés étrangères) reliant ces tables au reste du schéma ont été volontairement coupées (non implémentées) afin d'alléger le développement initial.

3 Développement

3.1 Réalisation des interfaces et programmes conformes aux spécifications fonctionnelles attendues

Requêtes	Actions / Correponsdance
`CREATE VIEW [DROIT].[V_acces] AS` `SELECT a.id_acces AS acces_id, u.username AS user_username, u.name AS user_name, u.firstname AS user_firstname, s.libelle AS user_solution, sc.libelle AS user_screen, d.libelle AS user_droit` `FROM droit.[user] u` `-- Jointure User -> Type_User` `INNER JOIN droit.type_user tu ON u.id_type_user = tu.id_type_user` `-- Jointure Type_User -> Acces` `INNER JOIN droit.acces a ON a.id_type_user = tu.id_type_user` `-- Jointure Acces -> Screen` `INNER JOIN droit.screen sc ON a.id_screen = sc.id_screen` `-- Jointure Screen -> Solution` `INNER JOIN droit.solution s ON sc.id_solution = s.id_solution` `-- Jointure Acces -> Droit` `INNER JOIN droit.droit d ON d.id_droit = a.id_droit` `GO`	**Vues :** V_acces
`CREATE VIEW [DROIT].[V_droit] AS` `SELECT *` `FROM droit.[droit]` `GO`	**Vues :** V_droit
`CREATE VIEW [DROIT].[V_screen] AS` `SELECT sc.id_screen, sc.libelle AS libelle_screen,s.libelle AS libelle_solution` `FROM DROIT.[screen] sc` `INNER JOIN DROIT.[solution] s ON sc.id_solution = s.id_solution` `GO`	**Vues :** V_screen
`CREATE VIEW [DROIT].[V_solution] AS` `SELECT *` `FROM droit.[solution]` `GO`	**Vues :** V_solution
`CREATE VIEW [DROIT].[V_user] AS` `SELECT *` `FROM droit.[user]` `GO`	**Vues :** V_user

```
CREATE PROCEDURE [DROIT].[spc_UserSelect]
AS
    DECLARE @ERROR_NUMBER INT;

    SET @ERROR_NUMBER = 0

                                BEGIN TRY
        SELECT *
        FROM DROIT.V_user

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT
        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE =
ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE =
ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH
GO
```

Procédure stockée : de la visualisation d'un utilisateur en fonction de la vue V_user

```
CREATE PROCEDURE  [DROIT].[spc_UserSelectByID]

    @Id_User INT
AS
    DECLARE @QUERY NVARCHAR(1000);
    DECLARE @PARAMETERS NVARCHAR(500);

    BEGIN TRY

    SET @PARAMETERS = N'@parameterId_User INT'

        SET @QUERY = 'SELECT * FROM [DROIT].[V_user] WHERE id_user = @parameterId_User'

        PRINT @QUERY

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_User = @Id_User

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_NUMBER INT;
        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT

        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH
GO
```

Procédure stockée : Récupération des données d'un utilisateur en fonction de l'id

```
CREATE PROCEDURE [DROIT].[spc_UserInsert]
    @username VARCHAR(100),
    @name VARCHAR(100),
    @firstname VARCHAR(100),
    @date_expiration DATETIME
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);
    DECLARE @IdTypeUser INT;

    BEGIN TRY

        INSERT INTO [DROIT].[type_user] DEFAULT VALUES;

        SET @IdTypeUser = SCOPE_IDENTITY();

        SET @PARAMETERS = N'@parameterUsername VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterName VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterFirstname VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterDate_expiration DATETIME, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterIdTypeUser INT';

        SET @QUERY = 'INSERT INTO [DROIT].[user] (username, name, firstname, date_expiration, is_actif, id_type_user)
        VALUES (@parameterUsername, @parameterName, @parameterFirstname, @parameterDate_expiration, 1, @parameterIdTypeUser)';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS,
        @parameterUsername = @username,
        @parameterName = @name,
        @parameterFirstname = @firstname,
        @parameterDate_expiration = @date_expiration,
        @parameterIdTypeUser = @IdTypeUser;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Insertion de données d'un utilisateur

```
CREATE PROCEDURE [DROIT].[spc_UserUpdate]
    @id_user INT,
    @username VARCHAR(100),
    @name VARCHAR(100),
    @firstname VARCHAR(100),
    @date_expiration DATETIME,
    @is_actif BIT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY
        SET @PARAMETERS = N'@parameterIdUser INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterUsername VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterName VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterFirstname VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterDate_expiration DATETIME, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterIsActif BIT';

        SET @QUERY = 'UPDATE [DROIT].[user] SET username = @parameterUsername,
        name = @parameterName,
        firstname = @parameterFirstname,
        date_expiration = @parameterDate_expiration,
        is_actif = @parameterIsActif
        WHERE id_user = @parameterIdUser';
        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS,
        @parameterIdUser = @id_user,
        @parameterUsername = @username,
        @parameterName = @name,
        @parameterFirstname = @firstname,
        @parameterDate_expiration = @date_expiration,
        @parameterIsActif = @is_actif;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(),
               @ERROR_SEVERITY = ERROR_SEVERITY(),
               @ERROR_STATE = ERROR_STATE();

        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Modification des données d'un utilisateur

```
CREATE PROCEDURE [DROIT].[spc_UserDelete]
    @id_user INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);
    DECLARE @IdTypeUser INT;

    BEGIN TRY

        SELECT @IdTypeUser = id_type_user
        FROM [DROIT].[user]
        WHERE id_user = @id_user;

        SET @PARAMETERS = N'@parameterIdUser INT';
        SET @QUERY = 'DELETE FROM [DROIT].[user] WHERE id_user = @parameterIdUser';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterIdUser = @id_user;

        IF @IdTypeUser IS NOT NULL
        BEGIN
            SET @PARAMETERS = N'@parameterIdTypeUser INT';

            SET @QUERY = 'DELETE FROM [DROIT].[type_user] WHERE id_type_user = @parameterIdTypeUser';

            PRINT @QUERY;

            EXEC sp_executesql @QUERY, @PARAMETERS, @parameterIdTypeUser = @IdTypeUser;
        END

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(),
               @ERROR_SEVERITY = ERROR_SEVERITY(),
               @ERROR_STATE = ERROR_STATE();

        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Suppression des données d'un utilisateur

```
CREATE PROCEDURE  [DROIT].[spc_SolutionSelect]
AS
    DECLARE @ERROR_NUMBER INT;

    SET @ERROR_NUMBER = 0


    BEGIN TRY
        SELECT *
        FROM DROIT.V_solution

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT
        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH

GO
```

Procédure stockée : de la visualisation d'une Solution fonction de la vue V_solution

```
CREATE PROCEDURE  [DROIT].[spc_SolutionSelectByID]

    @Id_Solution INT
AS
    DECLARE @QUERY NVARCHAR(1000);
    DECLARE @PARAMETERS NVARCHAR(500);

    BEGIN TRY

    SET @PARAMETERS = N'@parameterId_Solution INT'

        SET @QUERY = 'SELECT * FROM [DROIT].[V_solution] WHERE id_solution = @parameterId_Solution'

        PRINT @QUERY

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Solution = @Id_Solution

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_NUMBER INT;
        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT

        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH
GO
```

Procédure stockée : Récupération des données de la solution en fonction de l'id

```
CREATE PROCEDURE [DROIT].[spc_SolutionInsert]
    @libelle VARCHAR(100)
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterlibelle VARCHAR(100)';

        SET @QUERY = 'INSERT INTO [DROIT].[solution] (libelle) VALUES (@parameterLibelle)';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterLibelle = @libelle;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Insertion d'une solution

```sql
CREATE PROCEDURE [DROIT].[spc_SolutionUpdate]
    @Id_Solution INT,
    @libelle VARCHAR(100)
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Solution INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterLibelle VARCHAR(100)';

        SET @QUERY = 'UPDATE [DROIT].[solution] SET libelle = @parameterLibelle WHERE id_solution = @parameterId_Solution';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS,
            @parameterId_Solution = @Id_Solution,
            @parameterLibelle = @libelle;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Modification des données d'une solution

Procédure stockée : Suppression des données d'une solution

```sql
CREATE PROCEDURE  [DROIT].[spc_ScreenSelect]
AS
    DECLARE @ERROR_NUMBER INT;

    SET @ERROR_NUMBER = 0


    BEGIN TRY
        SELECT *
        FROM DROIT.V_screen

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT
        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH

GO
```

Procédure stockée : de la visualisation d'un écran en fonction de la vue V_screen

```sql
CREATE PROCEDURE  [DROIT].[spc_SceenSelectByID]

    @Id_Screen INT
AS
    DECLARE @QUERY NVARCHAR(1000);
    DECLARE @PARAMETERS NVARCHAR(500);

    BEGIN TRY

    SET @PARAMETERS = N'@parameterId_Screen INT'

        SET @QUERY = 'SELECT * FROM [DROIT].[V_screen] WHERE id_screen = @parameterId_Screen'

        PRINT @QUERY

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Screen = @Id_Screen

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_NUMBER INT;
        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT

        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH
GO
```

Procédure stockée : Récupération des données de la solution en fonction de l'id

```
CREATE PROCEDURE [DROIT].[spc_ScreenInsert]
    @libelle VARCHAR(100),
    @Id_Solution INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterLibelle VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterId_Solution INT';

        SET @QUERY = 'INSERT INTO [DROIT].[screen] (libelle, id_solution) VALUES (@parameterLibelle, @parameterId_Solution)';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterLibelle = @libelle, @parameterId_Solution = @Id_Solution;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Insertion d'un écran

```
CREATE PROCEDURE [DROIT].[spc_AccesUpdate]
    @Id_Type_User INT,
    @Id_Screen INT,
    @Id_Droit INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Type_User INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterId_Screen INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterId_Droit INT';

        SET @QUERY = 'UPDATE [DROIT].[acces]
                      SET id_screen = @parameterId_Screen,
                          id_droit = @parameterId_Droit
                      WHERE id_type_user = @parameterId_Type_User';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS,
            @parameterId_Type_User = @Id_Type_User,
            @parameterId_Screen = @Id_Screen,
            @parameterId_Droit = @Id_Droit;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(),
               @ERROR_SEVERITY = ERROR_SEVERITY(),
               @ERROR_STATE = ERROR_STATE();

        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Modification des données d'un écran

```sql
CREATE PROCEDURE [DROIT].[spc_ScreenDelete]
    @Id_Screen INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Screen INT';

        SET @QUERY = 'DELETE FROM [DROIT].[screen] WHERE id_screen = @parameterId_Screen';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Screen = @Id_Screen;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Suppression des données d'un écran

```sql
CREATE PROCEDURE  [DROIT].[spc_DroitSelect]
AS
    DECLARE @ERROR_NUMBER INT;

    SET @ERROR_NUMBER = 0


    BEGIN TRY
        SELECT *
        FROM DROIT.V_droit

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT
        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH

GO
```

Procédure stockée : de la visualisation d'un utilisateur en fonction de la vue V_droit

```sql
CREATE PROCEDURE  [DROIT].[spc_DroitSelectByID]

    @Id_Droit INT
AS
    DECLARE @QUERY NVARCHAR(1000);
    DECLARE @PARAMETERS NVARCHAR(500);

    BEGIN TRY

    SET @PARAMETERS = N'@parameterId_Droit INT'

        SET @QUERY = 'SELECT * FROM [DROIT].[V_droit] WHERE id_droit = @parameterId_Droit'

        PRINT @QUERY

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Droit = @Id_Droit

    END TRY

    BEGIN CATCH

        DECLARE @ERROR_NUMBER INT;
        DECLARE @ERROR_MESSAGE NVARCHAR(4000)
        DECLARE @ERROR_SEVERITY INT
        DECLARE @ERROR_STATE INT

        SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE()
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

    END CATCH
GO
```

Procédure stockée : Récupération des données de la solution en fonction de l'id

```sql
CREATE PROCEDURE [DROIT].[spc_DroitInsert]
    @Code VARCHAR(100),
    @Libelle VARCHAR(100)
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterCode VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterLibelle VARCHAR(100)';

        SET @QUERY = 'INSERT INTO [DROIT].[droit] (code, libelle) VALUES (@parameterCode, @parameterLibelle)';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterCode = @Code, @parameterLibelle = @Libelle;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Insertion d'un écran

```sql
CREATE PROCEDURE [DROIT].[spc_DroitUpdate]
    @Id_Droit INT,
    @Code VARCHAR(100),
    @Libelle VARCHAR(100)
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Droit INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterCode VARCHAR(100), '
        SET @PARAMETERS = @PARAMETERS + N'@parameterLibelle VARCHAR(100)';

        SET @QUERY = 'UPDATE [DROIT].[droit] SET code = @parameterCode, libelle = @parameterLibelle WHERE id_droit = @parameterId_Droit';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Droit = @Id_Droit, @parameterCode = @Code, @parameterLibelle = @Libelle;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Modification des données d'un écran

```sql
CREATE PROCEDURE [DROIT].[spc_DroitDelete]
    @Id_Droit INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Droit INT';

        SET @QUERY = 'DELETE FROM [DROIT].[droit] WHERE id_droit = @parameterId_Droit';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Droit = @Id_Droit;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Suppression des données d'un droit

```
CREATE PROCEDURE  [DROIT].[spc_AccesSelect]
AS
	DECLARE @ERROR_NUMBER INT;

	SET @ERROR_NUMBER = 0


	BEGIN TRY
		SELECT *
		FROM DROIT.V_acces

	END TRY

	BEGIN CATCH

		DECLARE @ERROR_MESSAGE NVARCHAR(4000)
		DECLARE @ERROR_SEVERITY INT
		DECLARE @ERROR_STATE INT
		SELECT @ERROR_NUMBER = ERROR_NUMBER(), @ERROR_MESSAGE =
ERROR_MESSAGE(), @ERROR_SEVERITY = ERROR_SEVERITY(), @ERROR_STATE =
ERROR_STATE()
		RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE)

	END CATCH

GO
```

Procédure stockée : de la visualisation d'un accès utilisateur en fonction de la vue V_acces

```
CREATE PROCEDURE [DROIT].[spc_AccesInsert]
        @Id_Type_User INT,
        @Id_Screen INT,
        @Id_Droit INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Type_User INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterId_Screen INT,'
        SET @PARAMETERS = @PARAMETERS + N'@parameterId_Droit INT';

        SET @QUERY = 'INSERT INTO [DROIT].[acces] (id_type_user, id_screen,
id_droit)
                VALUES (@parameterId_Type_User, @parameterId_Screen,
@parameterId_Droit)';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Type_User =
@Id_Type_User,
                @parameterId_Screen = @Id_Screen,
                @parameterId_Droit = @Id_Droit;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(), @ERROR_SEVERITY =
ERROR_SEVERITY(), @ERROR_STATE = ERROR_STATE();
        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Insertion d'un accès d'utilisateur

```sql
CREATE PROCEDURE [DROIT].[spc_AccesUpdate]
    @Id_Type_User INT,
    @Id_Screen INT,
    @Id_Droit INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Type_User INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterId_Screen INT, '
        SET @PARAMETERS = @PARAMETERS + N'@parameterId_Droit INT';

        SET @QUERY = 'UPDATE [DROIT].[acces]
                      SET id_screen = @parameterId_Screen,
                          id_droit = @parameterId_Droit
                      WHERE id_type_user = @parameterId_Type_User';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS,
            @parameterId_Type_User = @Id_Type_User,
            @parameterId_Screen = @Id_Screen,
            @parameterId_Droit = @Id_Droit;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(),
               @ERROR_SEVERITY = ERROR_SEVERITY(),
               @ERROR_STATE = ERROR_STATE();

        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Modification des données d'un d'un accès d'utilisateur

```sql
CREATE PROCEDURE [DROIT].[spc_AccesDelete]
    @Id_Acces INT
AS
    DECLARE @QUERY NVARCHAR(2000);
    DECLARE @PARAMETERS NVARCHAR(1000);

    BEGIN TRY

        SET @PARAMETERS = N'@parameterId_Acces INT ';

        SET @QUERY = 'DELETE FROM [DROIT].[acces]
                      WHERE id_acces = @parameterId_Acces';

        PRINT @QUERY;

        EXEC sp_executesql @QUERY, @PARAMETERS, @parameterId_Acces =
@Id_Acces;

    END TRY
    BEGIN CATCH
        DECLARE @ERROR_MESSAGE NVARCHAR(4000);
        DECLARE @ERROR_SEVERITY INT;
        DECLARE @ERROR_STATE INT;

        SELECT @ERROR_MESSAGE = ERROR_MESSAGE(),
               @ERROR_SEVERITY = ERROR_SEVERITY(),
               @ERROR_STATE = ERROR_STATE();

        RAISERROR (@ERROR_MESSAGE, @ERROR_SEVERITY, @ERROR_STATE);
    END CATCH;
GO
```

Procédure stockée : Suppresion d'un accès d'utilisateur

Explications	Déploiement des données vers la solution
Création de la fonction GetAcces() qui permet de retourner les données de la table acces via le dataset	`Public Function GetAcces() As Ent.DataSetDroit` `Dim dsDroit As New Ent.DataSetDroit` `Try` `Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString)` `Dim dataAdapater As New DataSetDroitTableAdapters.AccesTableAdapter` `dataAdapater.Connection = connexion` `dataAdapater.Fill(dsDroit.Acces)` `End Using` `Catch ex As Exception` `End Try` `Return dsDroit` `End Function`
	`Public Function GetUserId(ByVal pIdUser As Integer) As Ent.DataSetDroit` `Dim dsDroit As New Ent.DataSetDroit` `Try` `Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString)` `Dim dataAdapter As New DataSetDroitTableAdapters.UserTableAdapter` `dataAdapter.Connection = connexion` `dataAdapter.FillById(dsDroit.User, pIdUser)` `End Using` `Catch ex As Exception` `End Try` `Return dsDroit` `End Function`
Création de la fonction GetUser() qui permet de retourner les données de la table user via le dataset	`Public Function GetUser() As Ent.DataSetDroit` `Dim dsDroit As New Ent.DataSetDroit` `Try` `Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString)` `Dim dataAdapater As New DataSetDroitTableAdapters.UserTableAdapter` `dataAdapater.Connection = connexion` `dataAdapater.Fill(dsDroit.User)` `End Using` `Catch ex As Exception` `End Try` `Return dsDroit` `End Function`

Création de la fonction GetSolution() qui permet de retourner les données de la table solution via le dataset	Public Function GetSolution() As Ent.DataSetDroit Dim dsDroit As New Ent.DataSetDroit Try Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString) Dim dataAdapater As New DataSetDroitTableAdapters.SolutionTableAdapter dataAdapater.Connection = connexion dataAdapater.Fill(dsDroit.Solution) End Using Catch ex As Exception End Try Return dsDroit End Function
	Public Function GetSolutionId(ByVal pIdSolution As Integer) As Ent.DataSetDroit Dim dsDroit As New Ent.DataSetDroit Try Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString) Dim dataAdapter As New DataSetDroitTableAdapters.SolutionTableAdapter dataAdapter.Connection = connexion dataAdapter.FillById(dsDroit.Solution, pIdSolution) End Using Catch ex As Exception End Try Return dsDroit End Function
Création de la fonction GetAcces() qui permet de retourner les données de la table acces via le dataset	Public Function GetScreen() As Ent.DataSetDroit Dim dsDroit As New Ent.DataSetDroit Try Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString) Dim dataAdapater As New DataSetDroitTableAdapters.ScreenTableAdapter dataAdapater.Connection = connexion dataAdapater.Fill(dsDroit.Screen) End Using Catch ex As Exception End Try Return dsDroit End Function

	Public Function GetScreenId(ByVal pIdScreen As Integer) As Ent.DataSetDroit Dim dsDroit As New Ent.DataSetDroit Try Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString) Dim dataAdapter As New DataSetDroitTableAdapters.ScreenTableAdapter dataAdapter.Connection = connexion dataAdapter.FillById(dsDroit.Screen, pIdScreen) End Using Catch ex As Exception End Try Return dsDroit End Function
Création de la fonction GetAcces() qui permet de retourner les données de la table acces via le dataset	Public Function GetDroit() As Ent.DataSetDroit Dim dsDroit As New Ent.DataSetDroit Try Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString) Dim dataAdapater As New DataSetDroitTableAdapters.DroitTableAdapter dataAdapater.Connection = connexion dataAdapater.Fill(dsDroit.Droit) End Using Catch ex As Exception End Try Return dsDroit End Function
	Public Function GetDroitId(ByVal pIdDroit As Integer) As Ent.DataSetDroit Dim dsDroit As New Ent.DataSetDroit Try Using connexion As SqlClient.SqlConnection = New SqlClient.SqlConnection(Fx.Configuration.MESConnectionString) Dim dataAdapter As New DataSetDroitTableAdapters.DroitTableAdapter dataAdapter.Connection = connexion dataAdapter.FillById(dsDroit.Droit, pIdDroit) End Using Catch ex As Exception End Try Return dsDroit End Function

```
Public Function AddUser(ByVal username As String, ByVal name As
String, ByVal firstname As String, ByVal expiration As Date) As
Ent.DataSetDroit
        If (log4.IsInfoEnabled) Then log4.InfoFormat("BEGIN : AddUser
with username [{0}], name [{1}], firstname [{2}], date_expiration
[{3}]", username, name, firstname, expiration)

        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String = "[DROIT].[spc_UserInsert]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@username", DbType.String, 50,
ParameterDirection.Input, username)
                .ParameterPart.Add("@name", DbType.String, 50,
ParameterDirection.Input, name)
                .ParameterPart.Add("@firstname", DbType.String, 30,
ParameterDirection.Input, firstname)
                .ParameterPart.Add("@date_expiration", DbType.Date,
Nothing, ParameterDirection.Input, expiration)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.User.Merge(dtResult)
                End If
            Catch ex As Exception
                Throw (ex)
                log4.Error("Error lors de l'exécution de la procédure
stockée : " & ex.Message)
            End Try
        Catch ex As Exception
            Throw (ex)
        End Try

        If (log4.IsInfoEnabled) Then log4.InfoFormat("END : AddUser
with username [{0}], name [{1}], firstname [{2}], date_expiration
[{3}]", username, name, firstname, expiration)
        Return dsDroit
    End Function
```

```
Public Function UpdateUser(ByVal idUser As Integer, ByVal username As
String, ByVal name As String, ByVal firstname As String, ByVal
expiration As Date, isactif As Boolean) As Ent.DataSetDroit
        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String = "[DROIT].[spc_UserUpdate]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@id_user", DbType.Int32, Nothing,
ParameterDirection.Input, idUser)
                .ParameterPart.Add("@username", DbType.String, 50,
ParameterDirection.Input, username)
                .ParameterPart.Add("@name", DbType.String, 50,
ParameterDirection.Input, name)
                .ParameterPart.Add("@firstname", DbType.String, 30,
ParameterDirection.Input, firstname)
                .ParameterPart.Add("@date_expiration", DbType.Date,
Nothing, ParameterDirection.Input, expiration)
                .ParameterPart.Add("@is_actif", DbType.Int32, Nothing,
ParameterDirection.Input, isactif)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.User.Merge(dtResult)
                End If
            Catch ex As Exception
                Throw (ex)
                log4.Error("Error lors de l'exécution de la procédure
stockée : " & ex.Message)
            End Try
        Catch ex As Exception
            log4.Error("Error lors de la préparation de la procédure
stockée : " & ex.Message)
        End Try

        Return dsDroit
    End Function
```

```
Public Function DeleteUser(ByVal idUser As Integer) As
Ent.DataSetDroit
        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String = "[DROIT].[spc_UserDelete]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@id_user", DbType.Int32, Nothing,
ParameterDirection.Input, idUser)
            End With


            dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

            If (Not dsResult Is Nothing AndAlso dsResult.Tables.Count
> 0) Then
                Dim dtResult As DataTable = dsResult.Tables(0)
                dsDroit.User.Merge(dtResult)
            End If

        Catch ex As Exception
            Throw (ex)
            log4.Error("Error lors de l'exécution de la procédure
stockée : " & ex.Message)
        End Try

        Return dsDroit
    End Function
```

```
Public Function AddSolution(ByVal solution As String) As
Ent.DataSetDroit
        If (log4.IsInfoEnabled) Then log4.InfoFormat("BEGIN :
AddSolution with solution [{0}]", solution)

        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String =
"[DROIT].[spc_SolutionInsert]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@libelle", DbType.String, Nothing,
ParameterDirection.Input, solution)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.Solution.Merge(dtResult)
                End If
            Catch ex As Exception
                log4.Error("Erreur lors de l'exécution de la procédure
stockée : " & ex.Message)
                Throw
            End Try
        Catch ex As Exception
            log4.Error("Erreur dans AddSolution : " & ex.Message)
            Throw
        End Try

        If (log4.IsInfoEnabled) Then log4.InfoFormat("END :
AddSolution with datelog [{0}], nom [{1}], param [{2}], typ [{3}]",
solution)
        Return dsDroit
    End Function
```

```
Public Function UpdateSolution(ByVal id As Integer, ByVal solution As
String) As Ent.DataSetDroit
        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String =
"[DROIT].[spc_SolutionUpdate]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@Id_Solution", DbType.String, 50,
ParameterDirection.Input, id)
                .ParameterPart.Add("@libelle", DbType.String, 50,
ParameterDirection.Input, solution)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.Solution.Merge(dtResult)
                End If
            Catch ex As Exception
                Throw (ex)
                log4.Error("Error lors de l'exécution de la procédure
stockée : " & ex.Message)
            End Try
        Catch ex As Exception
            log4.Error("Error lors de la préparation de la procédure
stockée : " & ex.Message)
        End Try

        Return dsDroit
    End Function
```

```
Public Function AddScreen(ByVal screen As String, ByVal solution As
Integer) As Ent.DataSetDroit
        If (log4.IsInfoEnabled) Then log4.InfoFormat("BEGIN :
AddScreen with screen [{0}], solution [{2}]", screen, solution)

        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String =
"[DROIT].[spc_ScreenInsert]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@libelle", DbType.String, 50,
ParameterDirection.Input, screen)
                .ParameterPart.Add("@Id_Solution", DbType.Int32,
Nothing, ParameterDirection.Input, solution)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.Screen.Merge(dtResult)
                End If
            Catch ex As Exception
                log4.Error("Erreur lors de l'exécution de la procédure
stockée : " & ex.Message)
                Throw
            End Try
        Catch ex As Exception
            log4.Error("Erreur dans AddSolution : " & ex.Message)
            Throw
        End Try

        If (log4.IsInfoEnabled) Then log4.InfoFormat("END : AddScreen
with screen [{0}], solution [{2}]", solution)
        Return dsDroit
    End Function
```

```
Public Function UpdateScreen(ByVal id As Integer, ByVal screen As
String, ByVal solution As Integer) As Ent.DataSetDroit
        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String =
"[DROIT].[spc_ScreenUpdate]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@Id_Screen", DbType.Int32,
Nothing, ParameterDirection.Input, id)
                .ParameterPart.Add("@libelle", DbType.String, 50,
ParameterDirection.Input, screen)
                .ParameterPart.Add("@Id_Solution", DbType.Int32,
Nothing, ParameterDirection.Input, solution)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.Screen.Merge(dtResult)
                End If
            Catch ex As Exception
                Throw (ex)
                log4.Error("Error lors de l'exécution de la procédure
stockée : " & ex.Message)
            End Try
        Catch ex As Exception
            log4.Error("Error lors de la préparation de la procédure
stockée : " & ex.Message)
        End Try

        Return dsDroit
    End Function
```

```
Public Function AddDroit(ByVal code As String, ByVal droit As String)
As Ent.DataSetDroit
        If (log4.IsInfoEnabled) Then log4.InfoFormat("BEGIN : AddDroit
with code [{0}], libelle [{2}]", code, droit)

        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String =
"[DROIT].[spc_DroitInsert]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@Code", DbType.String, 50,
ParameterDirection.Input, code)
                .ParameterPart.Add("@Libelle", DbType.String, Nothing,
ParameterDirection.Input, droit)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.Droit.Merge(dtResult)
                End If
            Catch ex As Exception
                log4.Error("Erreur lors de l'exécution de la procédure
stockée : " & ex.Message)
                Throw
            End Try
        Catch ex As Exception
            log4.Error("Erreur dans AddSolution : " & ex.Message)
            Throw
        End Try

        If (log4.IsInfoEnabled) Then log4.InfoFormat("END : AddDroit
with code [{0}], libelle [{2}]", code, droit)
        Return dsDroit
    End Function
```

```
Public Function UpdateDroit(ByVal id As Integer, ByVal code As String,
ByVal libelle As String) As Ent.DataSetDroit
        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String =
"[DROIT].[spc_DroitUpdate]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@Id_Droit", DbType.Int32, Nothing,
ParameterDirection.Input, id)
                .ParameterPart.Add("@Code", DbType.String, 50,
ParameterDirection.Input, code)
                .ParameterPart.Add("@Libelle", DbType.String, 50,
ParameterDirection.Input, libelle)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.Droit.Merge(dtResult)
                End If
            Catch ex As Exception
                Throw (ex)
                log4.Error("Error lors de l'exécution de la procédure
stockée : " & ex.Message)
            End Try
        Catch ex As Exception
            log4.Error("Error lors de la préparation de la procédure
stockée : " & ex.Message)
        End Try

        Return dsDroit
    End Function
```

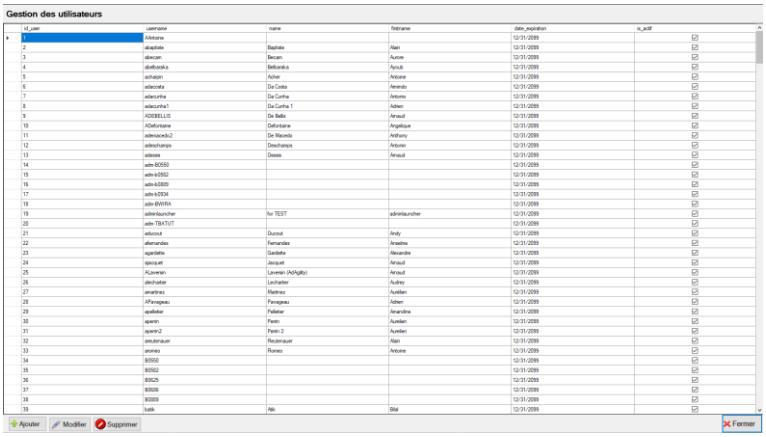
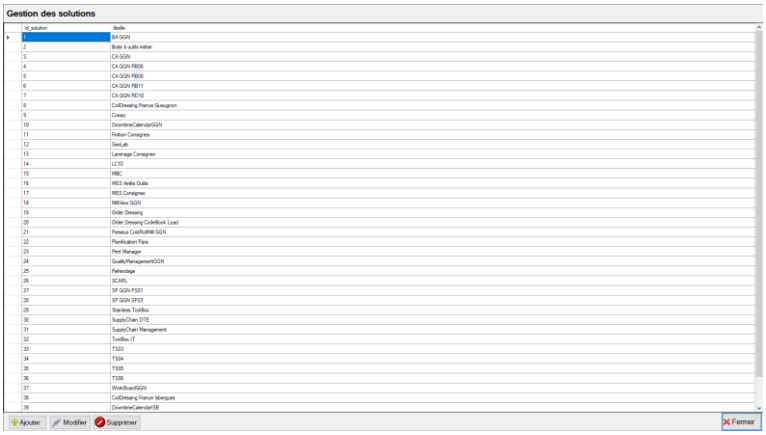
```
Public Function AddAcces(ByVal typeuser As Integer, ByVal screen As
Integer, ByVal droit As Integer) As Ent.DataSetDroit
        If (log4.IsInfoEnabled) Then log4.InfoFormat("BEGIN : AddAcces
with typeuser [{0}], screen [{1}], droit [{2}]", typeuser, screen,
droit)

        Dim dsDroit As New Ent.DataSetDroit
        Dim storedProcedureName As String =
"[DROIT].[spc_AccesInsert]"
        Dim dsResult As DataSet = Nothing
        Dim storedProcCmdBuilder As New
DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider)

        Try
            With storedProcCmdBuilder
                .StoredProcedureName = storedProcedureName
                .ParameterPart.Add("@Id_Type_User", DbType.Int32,
Nothing, ParameterDirection.Input, typeuser)
                .ParameterPart.Add("@Id_Screen", DbType.Int32,
Nothing, ParameterDirection.Input, screen)
                .ParameterPart.Add("@Id_Droit", DbType.Int32, Nothing,
ParameterDirection.Input, droit)
            End With

            Try
                dsResult =
Database.ExecuteDataSet(Fx.Configuration.MESConnectionString,
storedProcCmdBuilder)

                If (Not dsResult Is Nothing AndAlso
dsResult.Tables.Count > 0) Then
                    Dim dtResult As DataTable = dsResult.Tables(0)
                    dsDroit.Acces.Merge(dtResult)
                End If
            Catch ex As Exception
                Throw (ex)
                log4.Error("Error lors de l'exécution de la procédure
stockée : " & ex.Message)
            End Try
        Catch ex As Exception
            Throw (ex)
        End Try

        If (log4.IsInfoEnabled) Then log4.InfoFormat("END : AddAcces
with typeuser [{0}], screen [{1}], droit [{2}]", typeuser, screen,
droit)
        Return dsDroit
    End Function
```

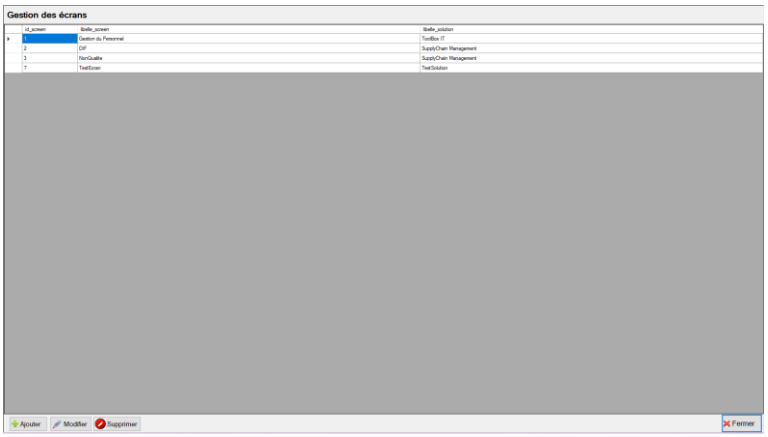
	`` Public Function UpdateAcces(ByVal id As Integer, ByVal typeuser As Integer, ByVal screen As Integer, ByVal droit As Integer) As Ent.DataSetDroit Dim dsDroit As New Ent.DataSetDroit Dim storedProcedureName As String = "[DROIT].[spc_AccesUpdate]" Dim dsResult As DataSet = Nothing Dim storedProcCmdBuilder As New DBStoredProcedureCommandBuilder(Fx.Configuration.SqlFactoryProvider) Try With storedProcCmdBuilder .StoredProcedureName = storedProcedureName .ParameterPart.Add("@Id_Acces", DbType.Int32, Nothing, ParameterDirection.Input, id) .ParameterPart.Add("@Id_Type_User", DbType.String, 50, ParameterDirection.Input, typeuser) .ParameterPart.Add("@Id_Screen", DbType.String, 50, ParameterDirection.Input, screen) .ParameterPart.Add("@Id_Droit", DbType.String, 50, ParameterDirection.Input, droit) End With Try dsResult = Database.ExecuteDataSet(Fx.Configuration.MESConnectionString, storedProcCmdBuilder) If (Not dsResult Is Nothing AndAlso dsResult.Tables.Count > 0) Then Dim dtResult As DataTable = dsResult.Tables(0) dsDroit.Acces.Merge(dtResult) End If Catch ex As Exception Throw (ex) log4.Error("Error lors de l'exécution de la procédure stockée : " & ex.Message) End Try Catch ex As Exception log4.Error("Error lors de la préparation de la procédure stockée : " & ex.Message) End Try Return dsDroit End Function ```
	`` Public Function GetAcces() As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetAcces() End Using End Using End Function ```
	`` Public Function GetUser() As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetUser() End Using End Using End Function ```
	`` Public Function GetUserId(ByVal pIdUser As Integer) As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetUserId(pIdUser:=pIdUser) End Using End Using End Function ```
	`` Public Function GetSolution() As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetSolution() End Using End Using End Function ```

	Public Function GetSolutionId(ByVal pIdSolution As Integer) As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetSolutionId(pIdSolution:=pIdSolution) End Using End Using End Function
	Public Function GetScreen() As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetScreen() End Using End Using End Function
	Public Function GetScreenId(ByVal pIdScreen As Integer) As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetScreenId(pIdScreen:=pIdScreen) End Using End Using End Function
	Public Function GetDroit() As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetDroit() End Using End Using End Function
	Public Function GetDroitId(ByVal pIdDroit As Integer) As Ent.DataSetDroit Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.GetDroitId(pIdDroit:=pIdDroit) End Using End Using End Function
	Public Function AddUser(ByVal username As String, ByVal name As String, ByVal firstname As String, ByVal expiration As Date) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.AddUser(username:=username, name:=name, firstname:=firstname, expiration:=expiration) End Using End Using Catch ex As Exception Throw (ex) End Try End Function
	Public Function UpdateUser(ByVal idUser As Integer, ByVal username As String, ByVal name As String, ByVal firstname As String, ByVal expiration As Date, isactif As Boolean) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.UpdateUser(idUser:=idUser, username:=username, name:=name, firstname:=firstname, expiration:=expiration, isactif:=isactif) End Using End Using Catch ex As Exception Throw (ex) End Try End Function

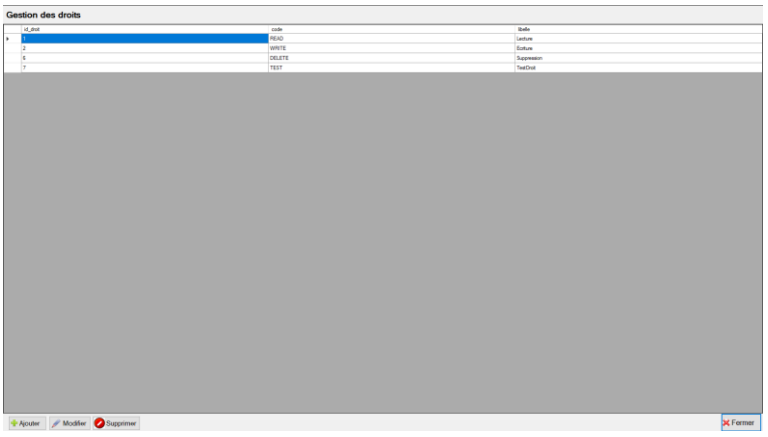
	Public Function DeleteUser(ByVal idUser As Integer) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.DeleteUser(idUser:=idUser) End Using End Using Catch ex As Exception Throw (ex) End Try End Function
	Public Function AddSolution(ByVal solution As String) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.AddSolution(solution:=solution) End Using End Using Catch ex As Exception Throw (ex) End Try End Function
	Public Function UpdateSolution(ByVal id As Integer, ByVal solution As String) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.UpdateSolution(id:=id, solution:=solution) End Using End Using Catch ex As Exception Throw (ex) End Try End Function
	Public Function AddScreen(ByVal screen As String, ByVal solution As Integer) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.AddScreen(screen:=screen, solution:=solution) End Using End Using Catch ex As Exception Throw (ex) End Try End Function
	Public Function UpdateScreen(ByVal id As Integer, ByVal screen As String, ByVal solution As Integer) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.UpdateScreen(id:=id, screen:=screen, solution:=solution) End Using End Using Catch ex As Exception Throw (ex) End Try End Function

	`` Public Function AddDroit(ByVal code As String, ByVal droit As String) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.AddDroit(code:=code, droit:=droit) End Using End Using Catch ex As Exception Throw (ex) End Try End Function ```
	`` Public Function UpdateDroit(ByVal id As Integer, ByVal code As String, ByVal droit As String) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.UpdateDroit(id:=id, code:=code, libelle:=droit) End Using End Using Catch ex As Exception Throw (ex) End Try End Function ```
	`` Public Function AddAcces(ByVal typeser As Integer, ByVal screen As Integer, ByVal droit As Integer) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.AddAcces(typeuser:=typeser, screen:=screen, droit:=droit) End Using End Using Catch ex As Exception Throw (ex) End Try End Function ```
	`` Public Function UpdateAcces(ByVal id As Integer, ByVal typeuser As Integer, ByVal screen As Integer, ByVal droit As Integer) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.UpdateAcces(id:=id, typeuser:=typeuser, screen:=screen, droit:=droit) End Using End Using Catch ex As Exception Throw (ex) End Try End Function ```
	`` Public Function DeleteAcces(ByVal id As Integer) As Ent.DataSetDroit Try Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer) Using data As New Stainless.MESITToolboxGGN.Dl.User(MetaData) Return data.DeleteAcces(id:=id) End Using End Using Catch ex As Exception Throw (ex) End Try End Function ```

Front-Office/Images	Back-Office/Code source
	`Private Sub LoadData()` `Dim result As Ent.DataSetDroit.UserDataTable` `Try` `Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer)` `Using data As New Stainless.MESITToolboxGGN.Bl.User(MetaData)` `result = data.GetUser().User` `' Rafraîchissement des données liées au DataGridView` `BindingSourceUser.DataSource = result` `BindingSourceUser.ResetBindings(False)` `End Using` `End Using` `Catch ex As Exception` `log4.Error("Erreur lors du chargement des données : " & ex.Message)` `End Try` `End Sub` `Private Sub User_Load(sender As Object, e As EventArgs) Handles MyBase.Load` `LoadData()` `End Sub`
	`Private Sub LoadData()` `Dim result As Ent.DataSetDroit.SolutionDataTable` `Try` `Using New Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic, mLayer)` `Using data As New Stainless.MESITToolboxGGN.Bl.User(MetaData)` `result = data.GetSolution().Solution` `' Rafraîchissement des données liées au DataGridView` `BindingSourceSolution.DataSource = result` `BindingSourceSolution.ResetBindings(False)` `End Using` `End Using` `Catch ex As Exception` `log4.Error("Erreur lors du chargement des données : " & ex.Message)` `End Try` `End Sub` `Private Sub User_Load(sender As Object, e As EventArgs) Handles MyBase.Load` `LoadData()` `End Sub`




```
    Private Sub LoadData()
        Dim result As Ent.DataSetDroit.ScreenDataTable
        Try

            Using New
Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic,
mLayer)
                Using data As New
Stainless.MESITToolboxGGN.Bl.User(MetaData)
                    result = data.GetScreen().Screen

                    ' Rafraîchissement des données liées au
DataGridView
                    BindingSourceScreen.DataSource = result

BindingSourceScreen.ResetBindings(False)
                End Using
            End Using
        Catch ex As Exception
            log4.Error("Erreur lors du chargement des
données : " & ex.Message)
        End Try
    End Sub

    Private Sub User_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
        LoadData()
    End Sub
```




```
    Private Sub LoadData()
        Dim result As Ent.DataSetDroit.DroitDataTable
        Try

            Using New
Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic,
mLayer)
                Using data As New
Stainless.MESITToolboxGGN.Bl.User(MetaData)
                    result = data.GetDroit().Droit

                    ' Rafraîchissement des données liées au
DataGridView
                    BindingSourceDroit.DataSource = result
                    BindingSourceDroit.ResetBindings(False)
                End Using
            End Using
        Catch ex As Exception
            log4.Error("Erreur lors du chargement des
données : " & ex.Message)
        End Try
    End Sub

    Private Sub User_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
        LoadData()
    End Sub
```


```
    Private Sub LoadData()
        Dim result As Ent.DataSetDroit.AccesDataTable
        Try

            Using New
Stainless.Framework.Diagnostics.Tracer(MetaData.Diagnostic,
mLayer)
                Using data As New
Stainless.MESITToolboxGGN.Bl.User(MetaData)
                    result = data.GetAcces().Acces

                    ' Rafraîchissement des données liées au
DataGridView
                    BindingSourceAcces.DataSource = result
                    BindingSourceAcces.ResetBindings(False)
                End Using
            End Using
        Catch ex As Exception
            log4.Error("Erreur lors du chargement des
données : " & ex.Message)
        End Try
    End Sub

    Private Sub Acces_Load(sender As Object, e As
EventArgs) Handles MyBase.Load
        LoadData()
    End Sub
```

3.3 Difficultés rencontrées (Bugs, Reste à faire)

La phase d'implémentation technique, identifiée comme le principal défi, a débuté avec succès. La structure théorique a été transposée concrètement dans le serveur SQL :

- **Le schéma de base de données** « droit » est en place et les tables ont été créées .
- **La sécurisation et l'accès aux données** se sont considérablement étoffés. Le système repose désormais sur **5 vues** (contre une seule initialement) et un ensemble de **17 procédures stockées** . Ces dernières couvrent désormais une large part des besoins fonctionnels : gestion des utilisateurs, des solutions, des écrans et des accès (lecture, insertion, mise à jour et suppression).

Reste à faire : Si la couche de données est quasiment finalisée avec l'ajout récent des procédures de suppression, l'effort se concentre maintenant sur l'aboutissement de l'application cliente. Les fonctionnalités de visualisation des pages et d'ajout d'un utilisateur sont opérationnelles. Il reste à finaliser l'intégration complète de la gestion des droits sur l'ensemble des écrans et à effectuer les derniers tests globaux (le "rush final") pour garantir la robustesse de la solution avant la livraison.

5 Bilan

Semaine 1 : Analyse et Maquettage Cette première semaine a été consacrée à l'immersion dans le projet et à l'appropriation des besoins du service informatique. J'ai travaillé sur la conception des interfaces utilisateurs (UI), en réalisant les maquettes qui ont servi de base pour valider l'ergonomie de la future application .NET.

Semaine 2 : Architecture de données L'accent a été mis sur le back-end avec la conception physique de la base de données sous SQL Server. J'ai procédé à la création des tables relationnelles (User, Groupe, Solution, Screen) et à l'insertion de jeux de données de test pour valider la structure et les clés étrangères.

Semaine 3 : Logique métier et début du développement J'ai optimisé la base de données en développant les procédures stockées nécessaires au CRUD (Create, Read, Update, Delete). Parallèlement, le développement de l'application client a débuté, permettant d'interfacer le code VB.NET avec la base SQL.

État actuel et Reste à faire (Mise à jour)

Au terme de cette période, l'implémentation technique a bien progressé. La structure de la base de données est opérationnelle. Concernant la couche applicative, j'ai finalisé :

L'intégration des **procédures de suppression** (DELETE) pour certaines tables, complétant ainsi les fonctionnalités de gestion de données.

La **visualisation des pages** principales de l'application.

Le module d'**ajout d'un utilisateur**, qui est désormais fonctionnel et relié à la base de données via la procédure stockée spc_UserInsert.

Le "rush final" consistera à étendre ces fonctionnalités à l'ensemble des écrans (gestion des droits par solution) et si j’ai la possibilité de finaliser l'interface de gestion des groupes.