Brevet de Technicien Supérieur SN
Session 20XX
Lycée Saint Gabriel – Saint Michel

SG01 – Studio des Langues

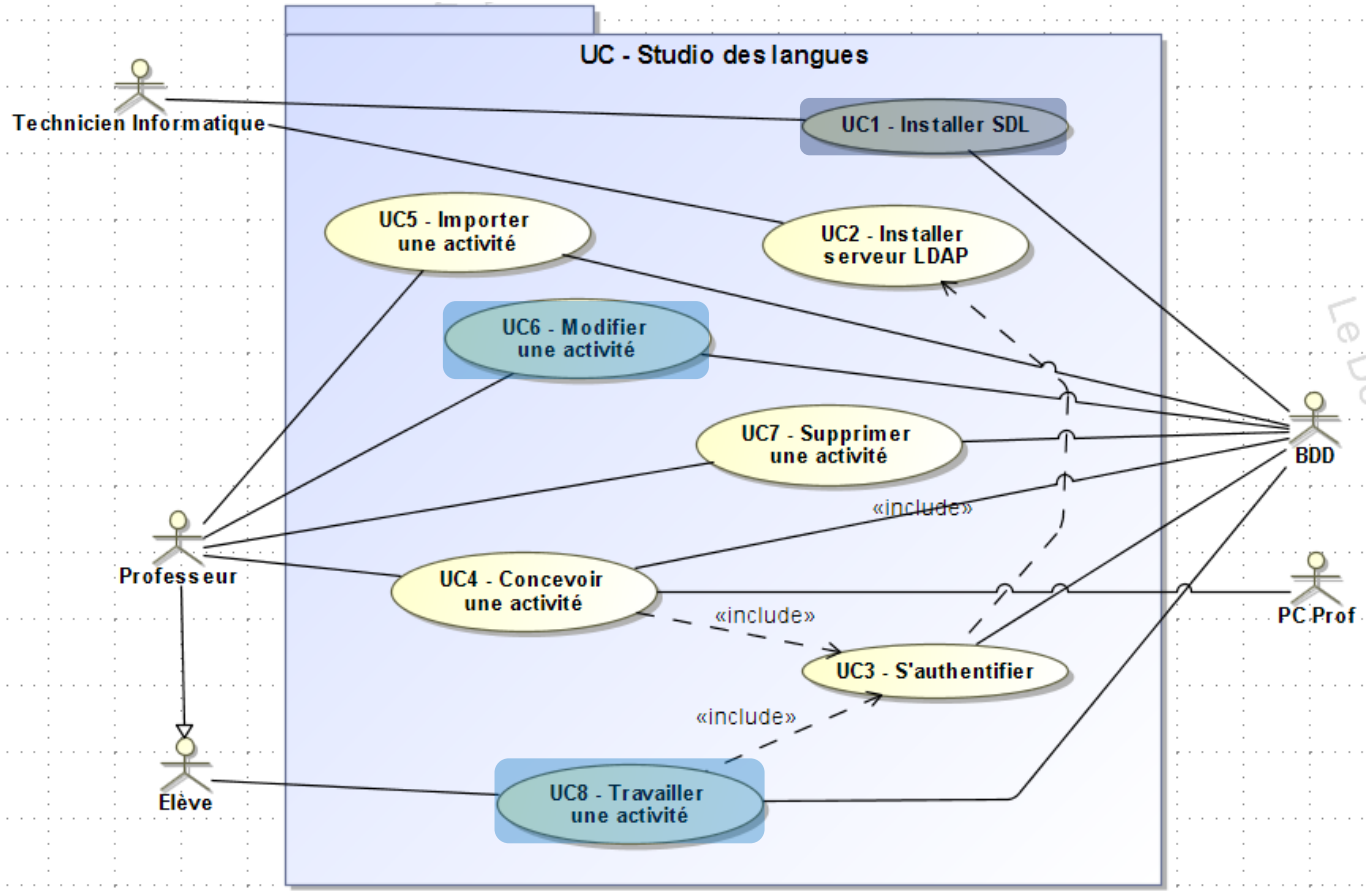
MAUDET Thibault

Dossier technique du projet – partie individuelle

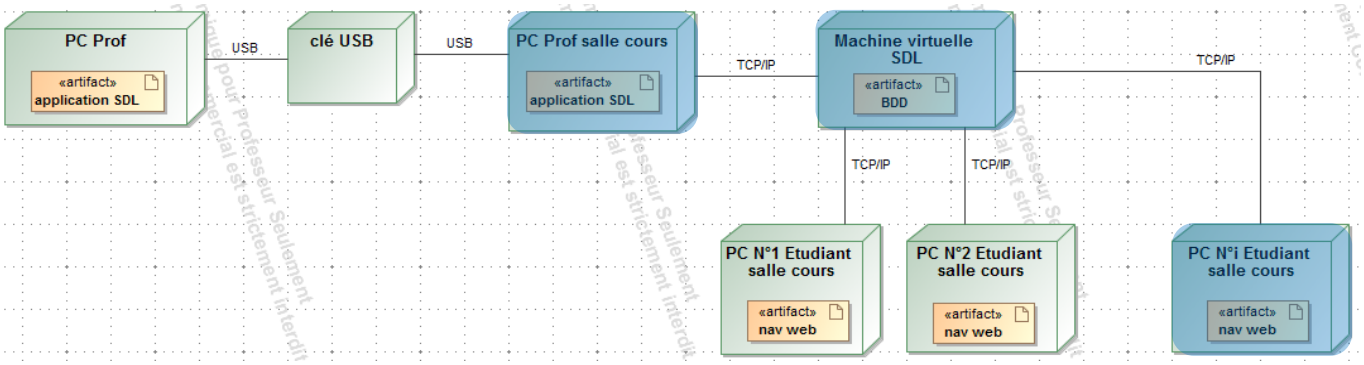
I. Situation dans le projet

I.1. Synoptique de la réalisation

Durant la phase d'étude du cahier des charges, nous avons décidé d'utiliser l'ASP.NET au lieu du Framework Symphony comme demandé dans le cahier des charges. En effet, Symphony étant un Framework fonctionnant avec le PHP, nous avons privilégié l'ASP.NET qui fonctionne avec le langage C# ce qui nous permet d'avoir un seul environnement de travail (sous Visual Studio). De plus, le fait d'avoir qu'un seul langage de programmation, cela nous permet de réutiliser certaines classes pour l'application professeur et l'application élèves sans avoir besoin d'adapter le code.



: Les cas d'utilisations sur lesquels je dois travailler dans le projet.



: Nœuds sur lesquels je suis intervenu.

Le nœud « *PC Prof salle cours* » permet au professeur qui se trouve dans une salle de cours de créer ou modifier une activité qui est stockée sur la base de données présente sur la Machine virtuelle SDL.

Le nœud « *Machine virtuelle SDL* » contient la base de données avec l'intégralité des activités importées depuis l'application professeur et le serveur WEB intégrant l'application Elèves.

Les nœuds « *PC N°i Etudiant salle cours* » permet aux élèves de travailler sur une activité se trouvant dans la base de données grâce à l'application stockée sur le serveur WEB.

I.2. Introduction

Pour la réalisation de ce projet, je dois effectuer plusieurs tâches telles que

- L'installation de la machine virtuelle qui héberge le serveur WEB - qui contient l'application élève - et la base de données qui permet de stocker les activités et les résultats des élèves.
- La configuration du serveur web, de la base de données, notamment les utilisateurs et les privilèges de ces derniers.
- La réalisation du digramme de classes de l'application WEB, du codage de l'application WEB et du codage de la fonction « Modifier une activité » en relation avec Antoine (Etudiant n°3).
- La documentation avec les différentes procédures d'installation et de configuration de la machine virtuelle SDL et le dictionnaire de classes.
-

Dans un premier temps, nous avons étudié le cahier des charges et nous avons mis en place la convention de nommage, les diagrammes de classes candidates des applications et la structure relationnelle de la base de données.

Ensuite, nous avons commencé l'installation et la configuration des machines virtuelles (SDL et serveur contenant l'Active Directory) et des différents services nécessaires pour la suite du projet (Serveur WEB, Base De Données, LDAP, etc…).

Une fois ces étapes terminées, nous avons commencé à coder les différentes applications. Afin de fournir des applications stables, nous avons effectué des tests unitaires, fonctionnels, etc…

Enfin, une fois toutes les autres étapes terminées, nous avons fait toute la documentation afin de permettre à d'autres personnes de pouvoir réinstaller les applications et la machine virtuelle SDL à l'avenir.

II. Réalisation de la fonction et/ou cas d'utilisation 6 : Modifier une activité

Le cas d'utilisation n°6 correspond à la modification d'une activité présente dans la base de données par un professeur.

Cette fonction permet au professeur de :

- Modifier le texte
- Ajouter et/ou supprimer des mots fournis,
- Modifier le titre de l'activité
- Gérer les classes qui peuvent voir
- Travailler l'activité et gérer la visibilité.

Pour effectuer la liaison entre l'application et la base de données, j'ai utilisé la classe **System.Data.SqlClient** fournit par le Framework Microsoft .NET qui permet la communication avec la base de données.

II.1. Conception détaillée

II.1.1. Diagramme de classes de l'application Elèves

Ci-dessous se trouve le diagramme de classes simplifié de l'application professeur. L'application fonctionne avec plusieurs fenêtres (qui sont aussi des classes car ces dernières héritent de la classe **Window**) et utilise aussi des classes externes – grâce à la DLL SDLL.dll – permettant notamment la communication avec la Base de Données ou le serveur d'**A**ctive **D**irectory grâce au protocole LDAP.

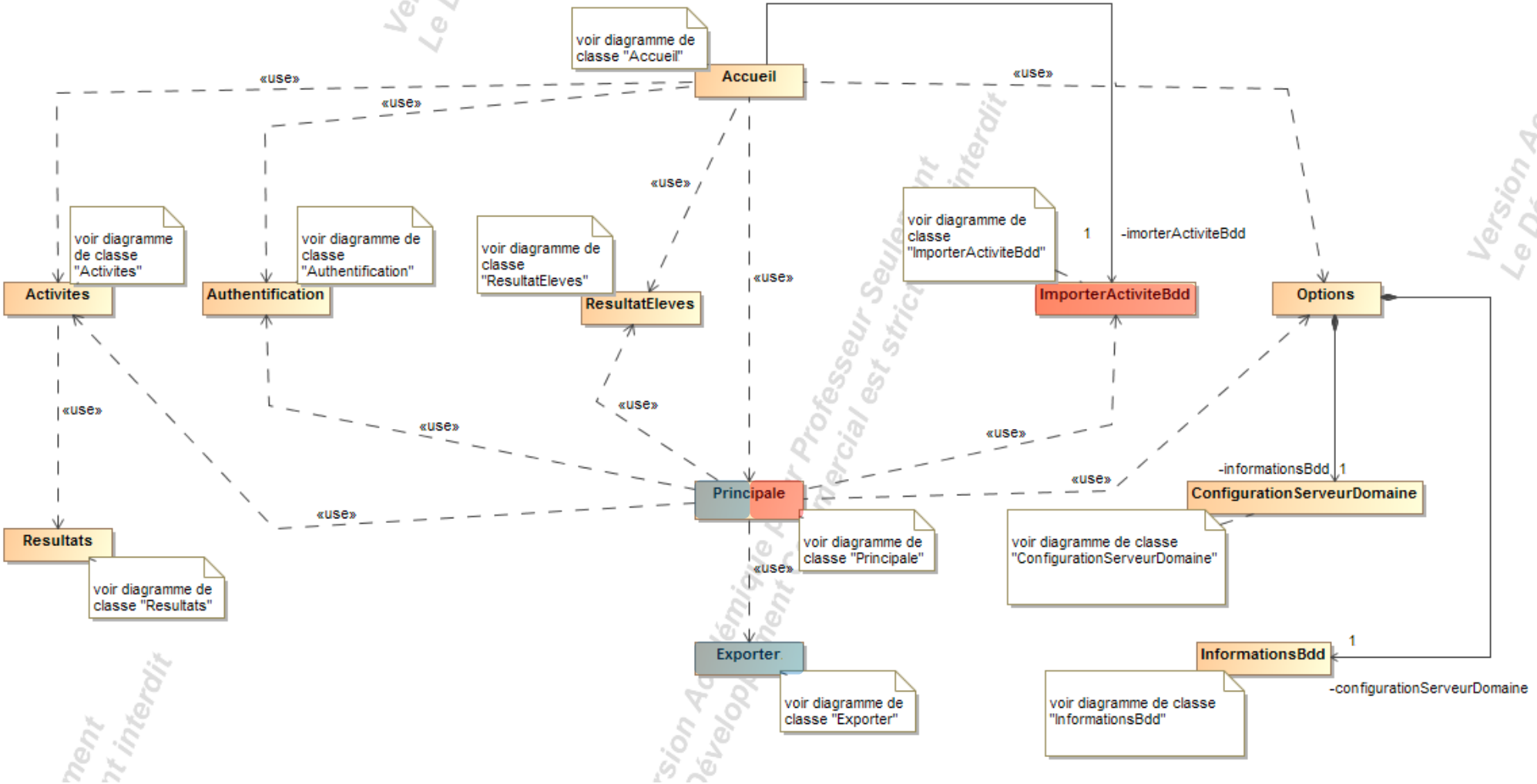
Pour la modification d'une activité, j'utilise les classes en surbrillance.

Les classes rouges sont utilisées lors de l'ouverture d'une activité tandis que les classes bleues permettent de sauvegarder une activité.

La classe « **ImporterActiviteBdd** » est la classe affichant la fenêtre permettant de choisir une activité à ouvrir.

La classe « **Exporter** » permet d'afficher la fenêtre de sauvegarde.

La classe « **Principale** » affiche la fenêtre où l'utilisateur saisit son texte et peut gérer la liste des mots fournis.


voir diagramme de classe "Accueil"
Accueil
«use»
«use»
«use»
«use»
voir diagramme de classe "Activites"
Activites
voir diagramme de classe "Authentification"
Authentification
voir diagramme de classe "ResultatEleves"
ResultatEleves
voir diagramme de classe "ImporterActiviteBdd"
1
-imorterActiviteBdd
ImporterActiviteBdd
Options
«use»
«use»
«use»
«use»
«use»
«use»
«use»
Principale
voir diagramme de classe "Principale"
-informationsBdd
1
ConfigurationServeurDomaine
voir diagramme de classe "ConfigurationServeurDomaine"
Resultats
voir diagramme de classe "Resultats"
«use»
Exporter
voir diagramme de classe "Exporter"
InformationsBdd
1
-configurationServeurDomaine
voir diagramme de classe "InformationsBdd"

Lors de l'ouverture d'une activité, l'application récupère l'ensemble des informations comprises dans la table **Activite** et les affichent dans l'application.

Au moment de la sauvegarde, la requête permettant d'insérer les données dans la base de données récupère les informations nécessaires et ajoute ou actualise l'activité dans la table **Activite** avec une interaction avec la table **AffectationClasses.**[1]

II.1.2. IHM de l'application

La fenêtre principale permet à l'utilisateur d'écrire du texte, d'ajouter et/ou de supprimer des mots fournis et d'accéder à d'autres modes tels que la consultation des résultats (nombre de mots trouvés et refusés sur le nombre de mots total) ou l'ouverture d'un résultat en particulier avec l'affichage des renseignements associés.

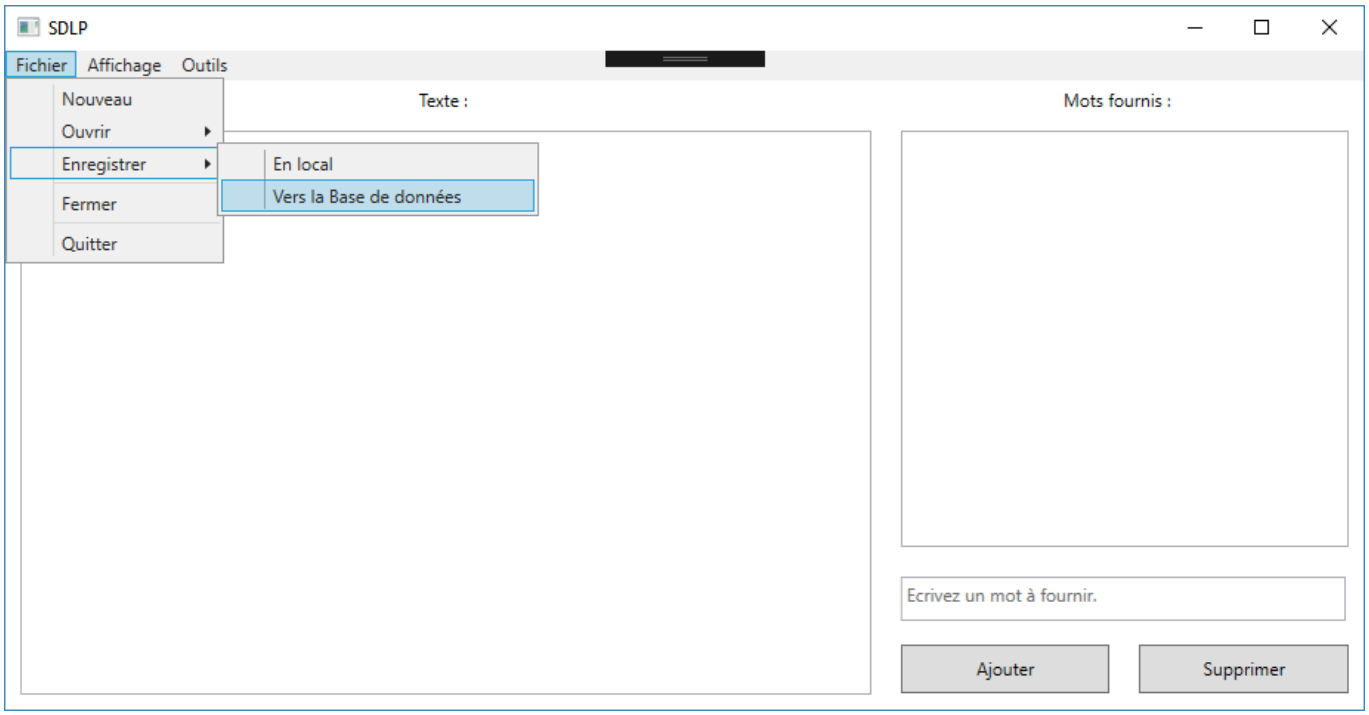


Image 1 : IHM principale de l'application

[1] Plus d'informations sur la structure de la base de données en Annexe 1

I.1.1.a. Ouverture d'une activité

La fenêtre **Importer une activité** permet au professeur de choisir l'activité à ouvrir. Pour cela, il suffit de sélectionner une activité et appuyer sur le bouton « **Ouvrir** » ou double-cliquez sur l'activité que l'on souhaite ouvrir.

Pour afficher la fenêtre d'importation, il suffit de se rendre dans Fichier → Ouvrir → Vers la base de données.

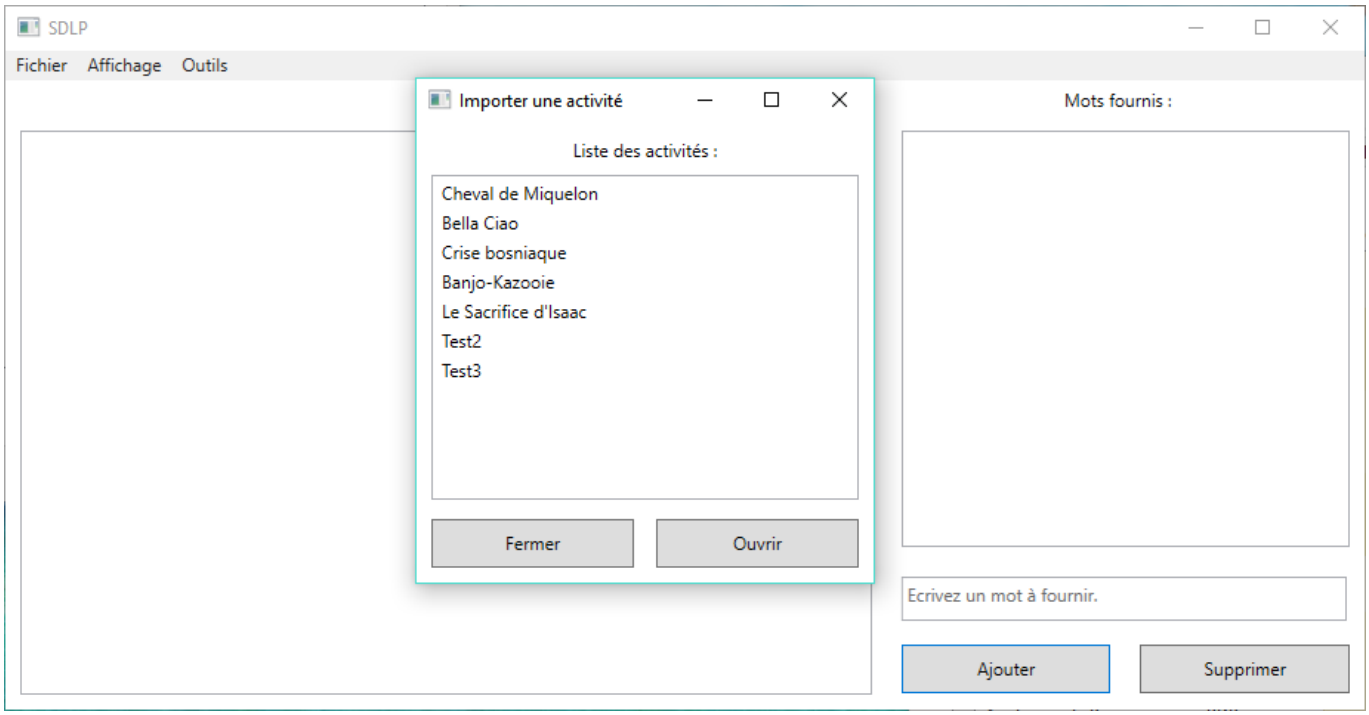


Image 2 : Ouvrir une activité.

I.1.1.b. Sauvegarde d'une activité

La fenêtre **Exporter une activité** permet au professeur de sauvegarder une activité dans la base de données. Si l'activité est déjà présente dans la base de données, les différents champs (nom de l'activité, listes des classes et visibilité de l'activité) sont pré-remplis avec les données enregistrées précédemment.

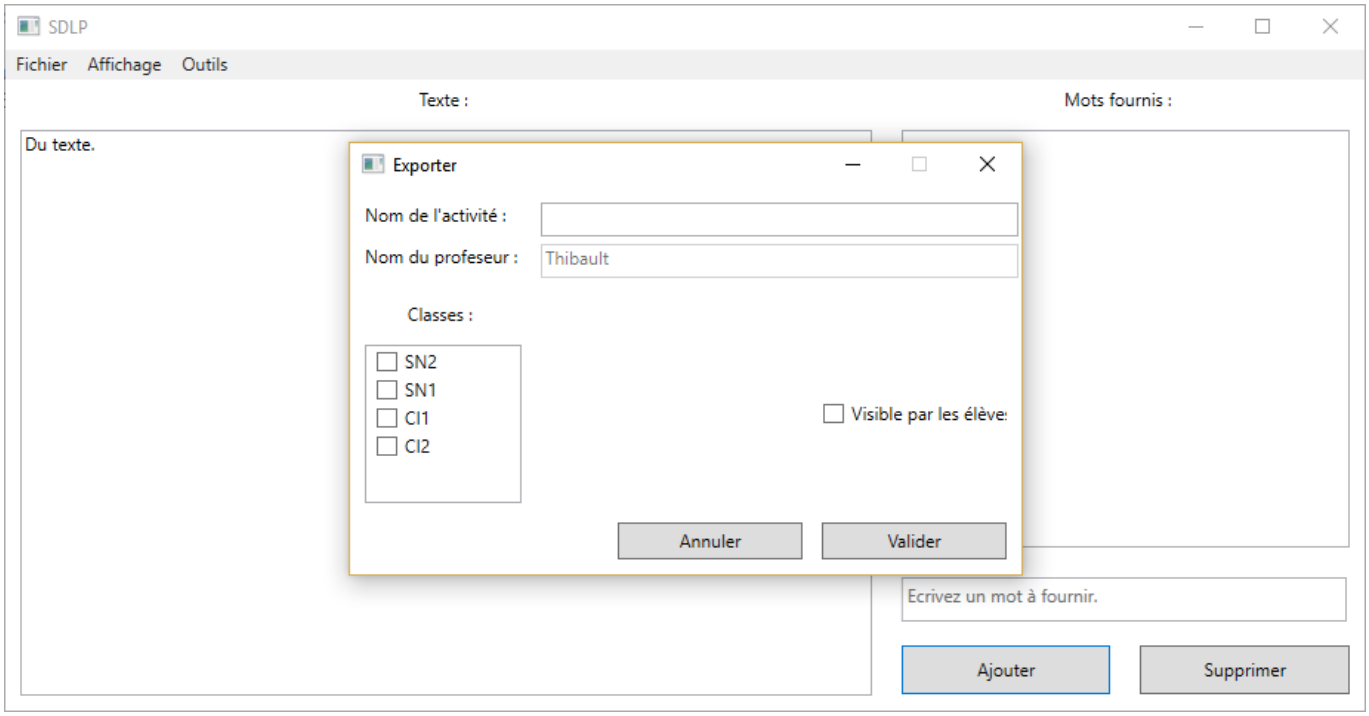


Image 3 : Exporter une activité dans la base de données

II.1.3. Extraits de code du cas d'utilisation

I.1.3.a. Ouverture d'une activité

Le code suivant est appelé lors de l'appui sur le bouton « **Ouvrir** » ou un double clic sur une activité.

```
public Activite RecupererActivite(int id)
{
    if (!ConnexionEstOk())
        return default(Activite);

    Activite activite = null;

    try
    {
        // Création de la commande
        SqlCommand command = connexion.CreateCommand();
        command.CommandText = "SELECT * FROM Activite WHERE Activite.IDActivite = @idActivite";
        command.Parameters.AddWithValue("@idActivite", id);

        // Lecture du résultat de la commande
        SqlDataReader dataReader = command.ExecuteReader();


        while (dataReader.Read())
            activite = new Activite() { IdActivite = dataReader.GetInt32(0), NomActivite = dataReader.GetString(1), NomProfesseur = dataReader.GetString(5),
              Texte = dataReader.GetString(3), MotsFournis = dataReader.GetString(2).Split(';').ToList(), Visibilite = dataReader.GetBoolean(4) };

        dataReader.Close();

        return activite;
    }
    catch (Exception e)
    {
        Information = e.Message;
        MessageBoxImage = MessageBoxImage.Error;

        return default(Activite);
    }
}
```

I.1.3.b. Sauvegarde d'une activité.

L'extrait suivant permet d'enregistrer une activité dans la base de données. Une vérification est préalablement effectuée pour vérifier le statut de l'activité (soit nouvelle/soit en cours de modification) grâce à l'ID de cette dernière.

Si une modification est effectuée le code suivant est utilisé par l'application.

```
    private void SauvegarderProfesseur(Activite activite)
    {
        try
        {
            // Création de la commande SQL.
            SqlCommand sqlCommand = connexion.CreateCommand();

            // Vérifier l'existence de l'activité → -2 signifie que l'activité est nouvelle.
            if (activite.IdActivite != -2)
            {
                // Supprimer l'ensemble des classes affectées à l'activité.
                sqlCommand.CommandText = "DELETE FROM AffectationClasse WHERE IDActivite =
@IDActivite";
                sqlCommand.Parameters.AddWithValue("@IDActivite", activite.IdActivite);
                sqlCommand.ExecuteNonQuery(); // Exécuter la commande.
                sqlCommand.Parameters.Clear(); // Effacer les paramètres précédemment saisis.

                // Mettre à jour l'activité dans la base de données.
                sqlCommand.CommandText = "UPDATE Activite SET Nom = @NomActivite,
NomProfesseur = @NomProfesseur, MotsFournis = @MotsFournis, Texte = @Texte, Visibilite =
@Visibilite WHERE IDActivite = @IDActivite";
                sqlCommand.Parameters.AddWithValue("@IdActivite", activite.IdActivite);
                sqlCommand.Parameters.AddWithValue("@NomActivite", activite.NomActivite);
                sqlCommand.Parameters.AddWithValue("@NomProfesseur", activite.NomProfesseur);
                sqlCommand.Parameters.AddWithValue("@MotsFournis",                string.Join(";",
activite.MotsFournis));
                sqlCommand.Parameters.AddWithValue("@Texte", activite.Texte);
                sqlCommand.Parameters.AddWithValue("@Visibilite", activite.Visibilite);

                sqlCommand.ExecuteNonQuery(); // Exécuter la commande.
                AffecterClasse(activite); // Affecter les classes à l'activité.

                Information = "Activité mise à jour";
            }
        }
    catch (Exception e) {    // Récupération et traitement de l'erreur. }
    }
```

II.2. Tests fonctionnels

II.2.1. Test fonctionnel du module matériel/logiciel

Élément testé :	Modification d'une activité.		
Objectif du test :	Tester qu'aucun problème ne se passe lorsqu'une activité présente sur la base de données subit une modification.		
Nom du testeur :	Thibault MAUDET	Date :	19/04/2018
Moyens mis en œuvre :	Logiciel : SDLP.exe	Matériel : PC sous Windows 10 Professionnel	Outil de développement : Visual Studio Enterprise 2017

Procédure du test :

Id	Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation (O/N)
1	L'utilisateur n'est pas connecté à la base de données.	0x0002	0x0002	O
2	La sauvegarde s'est déroulée correctement.	0x0001	0x0001	O
3	L'activité n'existe pas/plus.	0x0003	0x0003	O
4	Aucune classe n'a été sélectionnée.	0x1003	0x1003	O
5	Aucun texte n'a été rentré dans l'activité.	0x1002	0x1002	O
6	Aucun nom d'activité n'a été saisi	0x1004	0x1004	O

Conclusion du test :	La fonction de modification d'une activité est opérationnelle.

II.2.2. Problèmes rencontrés

Lors de la phase des tests, j'ai rencontré un problème car n'ayant jamais utilisé le module « Live Unit Testing » – qui permet de valider la couverture de code. J'ai donc dû apprendre à utiliser correctement le module.

III. Réalisation de la fonction et/ou cas d'utilisation 8 : Travailler une activité

Le cas d'utilisation n°8 correspond au travail d'une activité par un élève ou un étudiant depuis l'application dédiée.

L'élève doit être capable, après s'être identifié sur l'interface WEB et avoir choisi une activité, de saisir les mots identifiés, sauvegarder son travail dans la base de données pour retravailler dessus ultérieurement.

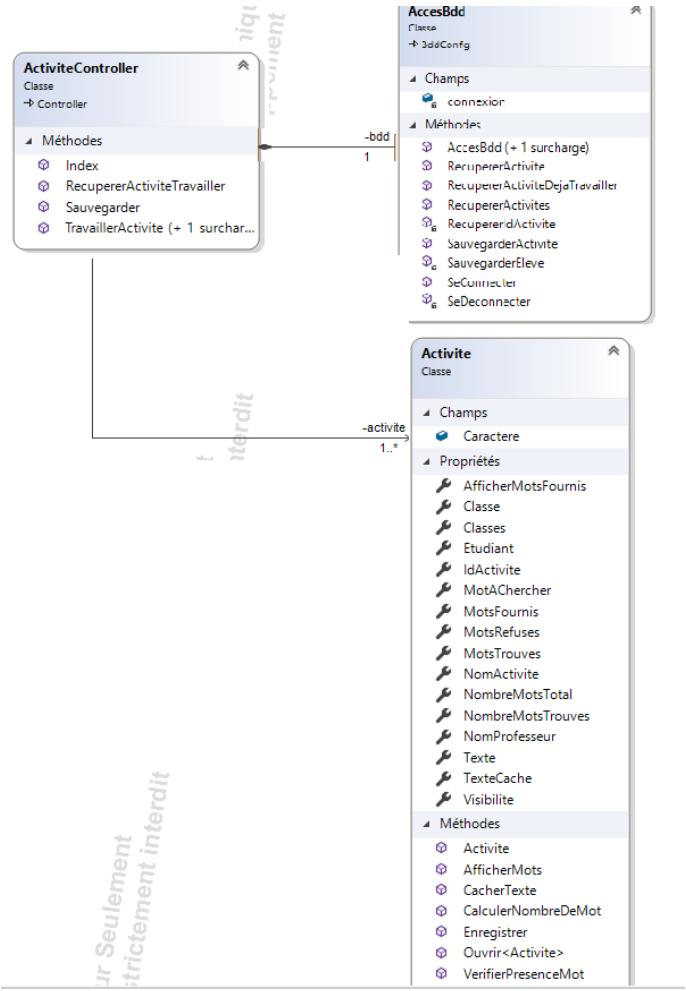
Travaillant avec Yoann (Étudiant 1) sur ce cas d'utilisation, je me suis occupé de la partie de la sauvegarde de l'activité dans la base de données.

Pour effectuer la liaison entre l'application et la base de données, j'ai utilisé la classe **System.Data.SqlClient** fournit par le Framework Microsoft .NET qui permet la communication avec la base de données.

III.1. Conception détaillée

III.1.1. Diagramme de classes de l'application

Ci-dessous se trouve le diagramme de classes de l'application élèves. Le contrôleur **ActiviteController** interagit avec les classes **AccesBDD** – qui permet l'interaction avec la base de données – et **Activite** – qui contient l'ensemble des données de l'activité.



Lors de la sauvegarde, les informations sont stockées dans la Base de Données et plus précisément dans la table **Resultats**. Une activité pouvant être travaillée pour la première fois ou être retravaillée, une vérification est effectuée pour permettre d'écraser les anciennes données si nécessaire.[2]

III.1.2. IHM de l'application

L'IHM suivante permet à l'utilisateur de travailler une activité, c'est-à-dire saisir le texte qu'il comprend lorsqu'il écoute une vidéo ou un fichier audio. L'utilisateur a la possibilité de sauvegarder son travail en appuyant sur le bouton « **Sauvegarder** ».

Image 4 : Page de travail d'une activité.

[2] Plus d'informations sur la structure de la base de données en Annexe 1

III.1.3. Extrait de code du cas d'utilisation

Le code suivant vérifie si l'étudiant travaille pour la première fois sur une activité ou continu son travail sur une activité. En fonction du résultat, l'activité est insérée est mise à jour ou les données insérées dans la table **Résultats**.

```
private void SauvegarderEleve(Activite activite)
{
    try
    {
        if (activite.Etudiant.IdEtudiant == -2)
            activite.Etudiant.IdEtudiant = IdEtudiant(activite.Etudiant.NomEtudiant, activite.Etudiant.PrenomEtudiant);

        if (activite.Etudiant.IdEtudiant == -1)
        {
            Information = "Impossible de trouver l'élève ou l'étudiant dans la base de donnée.";
            MessageBoxImage = MessageBoxImage.Error;
        }

        if (ActiviteExiste(activite))
        {
            SqlCommand command = connexion.CreateCommand();
            command.CommandText = "UPDATE Resultat SET MotsTrouves = @MotsTrouves, MotsRefuses = @MotsRefuses FROM Etudiant, Activite WHERE Activite.IDActivite = Resultat.IDActivite AND
              Resultat.IDEtudiant = Etudiant.IDEtudiant AND Activite.IDActivite = @IDActivite AND Etudiant.IDEtudiant = @IDEtudiant";
            command.Parameters.AddWithValue("@MotsTrouves", string.Join(";", activite.MotsTrouves));
            command.Parameters.AddWithValue("@MotsRefuses", string.Join(";", activite.MotsRefuses));
            command.Parameters.AddWithValue("@IDActivite", activite.IdActivite);
            command.Parameters.AddWithValue("@IDEtudiant", activite.Etudiant.IdEtudiant);

            command.ExecuteNonQuery();
        }
        else
        {
            SqlCommand command = connexion.CreateCommand();
            command.CommandText = "INSERT INTO Resultat (IDEtudiant, MotsTrouves, MotsRefuses, IDActivite, Texte) VALUES (@IDEtudiant, @MotsTrouves, @MotsRefuses, @IDActivite, @Texte)";
            command.Parameters.AddWithValue("@IDEtudiant", activite.Etudiant.IdEtudiant);
            command.Parameters.AddWithValue("@MotsTrouves", string.Join(";", activite.MotsTrouves));
            command.Parameters.AddWithValue("@MotsRefuses", string.Join(";", activite.MotsRefuses));
            command.Parameters.AddWithValue("@IDActivite", activite.IdActivite);
            command.Parameters.AddWithValue("@Texte", activite.Texte);

            command.ExecuteNonQuery();
        }
    }
    catch (Exception e)
    {
        Information = e.Message;
        MessageBoxImage = MessageBoxImage.Error;
    }
}
```

III.2. Tests unitaires

III.2.1. Test unitaire du module matériel/logiciel

Élément testé :	Sauvegarde d'une activité par un élève		
Objectif du test :	Vérifier que l'élève peut sauvegarder son travail dans la base de données.		
Nom du testeur :	Thibault MAUDET	Date :	
Moyens mis en œuvre :	Logiciel : SDLE	Matériel : PC sous Windows 10 Professionnel	Outil de développement : Visual Studio Enterprise 2017

Procédure du test :

Id	Description du vecteur de test	Résultat attendu	Résultat obtenu	Validation (O/N)

Conclusion du test :	

IV. Bilan de la réalisation personnelle

J'ai pu dans l'ensemble mener à bien l'ensemble des tâches que je devais faire. En effet, le cas d'utilisation n°1 qui consistait à installer la machine virtuelle SDL et le cas d'utilisation n°6 qui équivaut à la modification d'une activité par un professeur sont fonctionnels.

Cependant, le cas d'utilisation n°8 (Travailler une activité) n'est pas entièrement validé car je n'ai pas encore eu le temps de faire les tests sur la sauvegarde des résultats. De plus, je rencontre actuellement un problème sur la sauvegarde. En effet, la sauvegarde est entièrement fonctionnelle lorsque l'application est démarrée depuis Visual Studio mais lorsque le logiciel est lancé depuis le serveur WEB, la sauvegarde n'interagit pas avec la base de données.

Malgré que le projet ne soit pas entièrement terminé, plusieurs pistes d'amélioration peuvent être envisagés notamment sur l'interaction avec la base de données. En effet, actuellement, l'administrateur réseau est obligé d'ajouter et de supprimer manuellement les classes et les élèves, des outils d'administration peuvent donc un ajout à faire pour lui permettre d'automatiser l'actualisation des différentes listes.

De plus, pour le moment, un seul compte, le compte administrateur, permet d'interagir avec la base de données, la notion de gestion d'autorisation est aussi une amélioration possible. On peut imaginer par exemple :

- Un compte administrateur ayant le contrôle total sur l'ensemble de la base de données.
- Un compte « professeur » pouvant seulement interagir avec les tables « **Activites** » et « **Resultats** ».
- Un compte « élève » avec pour seule autorisation, la modification de la table « **Resultats** ».

Ce projet touchant plusieurs domaines notamment le développement logiciel et le développement WEB m'a permis d'acquérir de nouvelles connaissances. En effet, la partie WEB devant être développé avec le paterne MVC était une architecture totalement nouvelle pour moi. J'ai donc pu découvrir une nouvelle manière de développer un site internet.

De plus, j'ai pu apprendre à mieux utiliser les outils fournis par Visual Studio et plus particulièrement le module de tests unitaires et fonctionnels et le module VSTS (**V**isual **S**tudio **T**eam **S**ervices) qui permet de collaborer sur un projet.

GLOSSAIRE

A

- **A**ctive **D**irectory : Service d'annuaire LDAP mis en œuvre par Microsoft pour les systèmes d'exploitation Windows.

D

- DLL (**D**ynamic **L**ink **L**ibrairy) : Bibliothèque logicielle dont les fonctions sont chargées dynamiquement à l'exécution du programme.

L

- LDAP (**L**ightweight **D**irectory **A**ccess **P**rotocol) : Protocole permettant l'interrogation et/ou la modification des services d'annuaire.

V

- VSTS (**V**isual **S**tudio **T**eam **S**ervices) : Ensemble d'outils de collaboration basés sur le Cloud permettant de planifier, développer et gérer de manière efficace des projets logiciels de toutes tailles, dans n'importe quel langage de programmation.

ANNEXE 1 : STRUCTURE RELATIONNELLE DE LA BASE DE DONNÉES

La base de données est structurée de la manière suivante :

- La table « **Activites** » stockant le texte, la liste des mots fournis, sa visibilité et le nom du professeur ayant créé l'activité.
- La table « **AffectationClasse** » qui permet de lier une liste de classe à une activité.
- La table « **Classes** » qui contient l'intégralité des classes de l'établissement.
- La table « **Etudiant** » qui contient la liste des élèves et étudiants.
- La table « **Resultats** » qui permet de sauvegarder une activité travaillée dans l'application élèves. Cette table stocke diverses informations telles que la liste des mots trouvés et incorrects