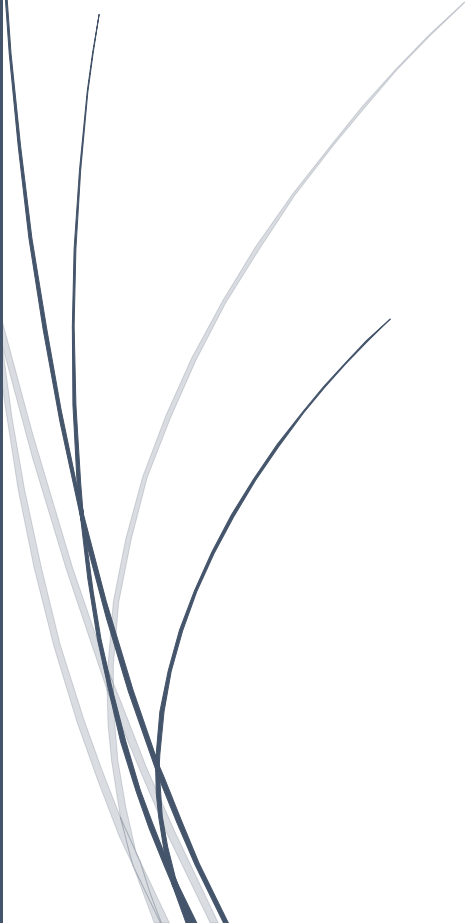


A thick dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

03/07/2020

# Projet développement

Several thin, curved lines in dark blue and light grey originate from the bottom left and curve upwards and to the right.

Guillaume ROBERT – Robin Maisano – Pacôme  
CHU-Lejeune  
CESI EXIA A4

# Table des matières

Introduction.....	2
Rappel du besoin.....	3
Découpage du projet.....	4
Planification initiale.....	4
Répartition des tâches.....	6
Modélisation de l'application distribuée .....	7
Architecture logicielle globale.....	7
Architecture physique globale .....	8
Modélisation diagramme de classes par couche et par plateforme .....	9
Diagramme de cas d'activité .....	11
Diagramme de séquence par couche et par plateforme .....	12
Analyse des écarts.....	14
Partie C# (client lourd) .....	14
Partie C# (middleware).....	16
Partie JEE .....	16
Analyse des compétences acquises par étudiants.....	17
Pacôme Chu-Lejeune.....	17
Robin Maisano.....	17
Guillaume Robert .....	17
Bilan.....	18

## Introduction

Pour ce projet, il nous est demandé de réaliser une application type “Client Lourd”, avec Windows Presentation Foundation (WPF), permettant d’envoyer des fichiers chiffrés vers un Middleware développé en Windows Communication Foundation (WCF). Ce middleware est chargé de déchiffrer les fichiers puis de les transmettre à un troisième composant en Java Entreprise Edition qui se chargera lui de l’authentification des documents avant de les retourner à l’utilisateur.

Le but premier de ce projet est de nous pousser à mettre en pratique les principes SOA (Service Oriented Architecture) et de nous faire travailler sur de multiples technologies qui doivent agir de concert.

## Rappel du besoin

Plusieurs besoins sont exprimés sur ce projet :

- L'utilisateur doit pouvoir s'authentifier et la validation des identifiants fournis est réalisée par un service WCF qui répondra avec un token utilisateur. Lors des prochains accès aux autres services, l'application joindra à sa requête son token utilisateur pour éviter à l'utilisateur d'avoir à s'authentifier à chaque accès.
- L'utilisateur doit pouvoir sélectionner un ou plusieurs fichiers qu'il souhaite décrypter et les envoyer au Middleware WCF.
- Le Middleware, lorsqu'il reçoit une demande d'accès à un service, vérifie l'authenticité des identifiants ou du token utilisateur fourni et que le niveau d'accès de l'utilisateur correspond à celui du service. Une fois l'authenticité de l'utilisateur et ses droits d'accès vérifiés, le Middleware lance le déchiffrement des fichiers reçus
- Le déchiffrement consiste en la génération d'une suite de clé de quatre lettres, majuscules, françaises et non accentués. Chaque clé est ensuite appliquée sur le contenu du fichier avec un déchiffrement de type XOR. Chaque tentative est envoyée à l'authentificateur JavaEE qui va se charger de vérifier la validité de ce déchiffrement.
- Le déchiffrement s'effectue de manière parallèle sur autant de fichiers qu'il y a de cœur de processeur disponibles. Si l'ensemble des cœurs sont utilisés, les fichiers suivants attendront qu'un cœur se libère.
- Le service Java EE stocke dans une Queue JMS chaque demande provenant du Middleware. Un EJB (Entreprise Java Beans) va ensuite comparer les mots du contenu déchiffré avec une liste de mots français stockés dans sa Base de Données. Si un nombre conséquent de mots correspondent, la clé associée sera considérée comme la clé de cryptage original.
- Le service Java EE, lorsqu'un fichier est identifié comme français, cherche "l'information secrète" potentiellement cachée dans le fichier.
- Le service Java EE, lorsqu'un fichier est déchiffré, contacte le Middleware pour l'informer que le déchiffrement est terminé sur le fichier en question et lui passer les informations liées à cette réussite : NomFichier, Clé, TauxConfiance, InformationSecrete.

Le Middleware WCF, lorsque tous les fichiers sont déchiffrés, renvoi les données les concernant au Client et génère une Rapport PDF avant de l'envoyer par mail au client.

## Découpage du projet

Le projet est découpé en 3 grandes parties, dont deux possédant une Base de Données :

- Un Client Lourd en WPF (Windows Presentation Foundation), permettant l'authentification de l'utilisateur, la sélection des fichiers à déchiffrer et la visualisation des résultats du déchiffrement.
- Un Middleware WCF (Windows Communication Foundation) et sa Base de Données SQL Server, permettant d'authentifier un utilisateur, de décrypter les fichiers envoyés et de renvoyer les résultats, au travers de trois services :
  - Service de login (*LoginService*)
  - Service de déchiffrement (*DecryptService*)
  - Service de mise à jour de l'état de déchiffrement (*IsDecryptedService*)
- Un Webservice Java EE et sa Base de Données Oracle, permettant de stocker les demandes de vérification du client, de traiter les demandes de vérification afin de s'assurer du bon déchiffrement du texte et l'envoi des données de déchiffrement au Middleware WCF.

## Planification initiale

Pour notre planification initiale nous avons réalisé le Gantt ci-dessous, en découpant chaque partie en grandes tâches et en tentant d'estimer la durée de chaque tâche via un planning poker en mettant nos connaissances et nos expériences passées à profit.

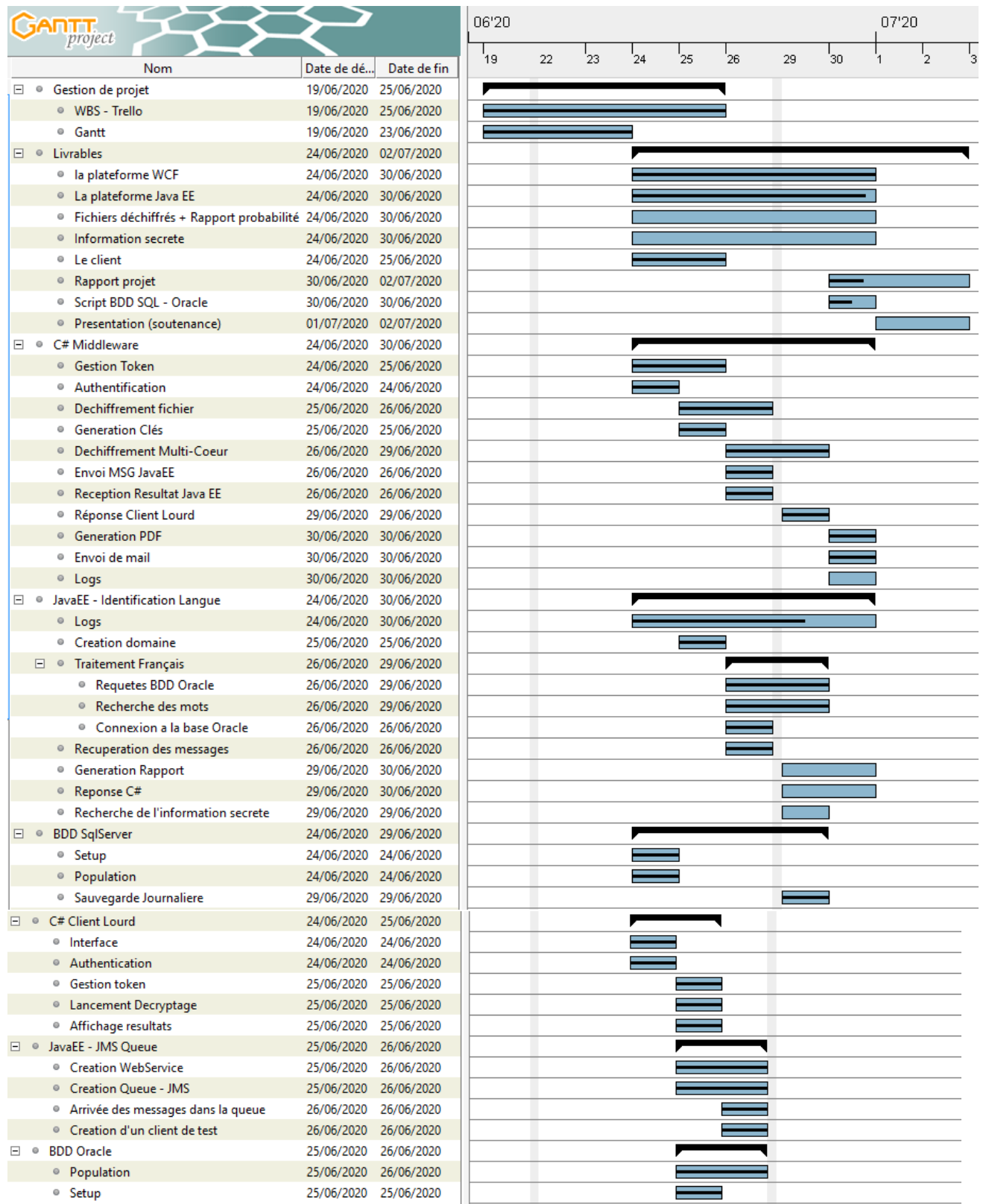


Diagramme de Gantt représentant la planification initiale du projet

## Répartition des tâches

Ci-dessous la répartition des différentes tâches identifiées, que nous avons appliquée durant toute la durée du projet. A noter que nous avons tous eu à intervenir sur les différentes tâches pour mettre en commun nos travaux, discuter d'une solution technique ou résoudre un problème causé par un morceau de nos codes respectifs.

- **Modélisation & Architecture**

Pacôme – Robin – Guillaume

- **Client Lourd (C#)**

Pacôme

- **Middleware WCF & Base de Données SQL Serveur**

Robin

- **Webservice Java EE & Base de Données Oracle**

Guillaume

- **Livrables**

Pacôme – Robin – Guillaume

# Modélisation de l'application distribuée

## Architecture logicielle globale

Ci-dessous l'architecture logicielle globale de la solution. Les bases de données ne sont pas représentées par soucis de clarté.

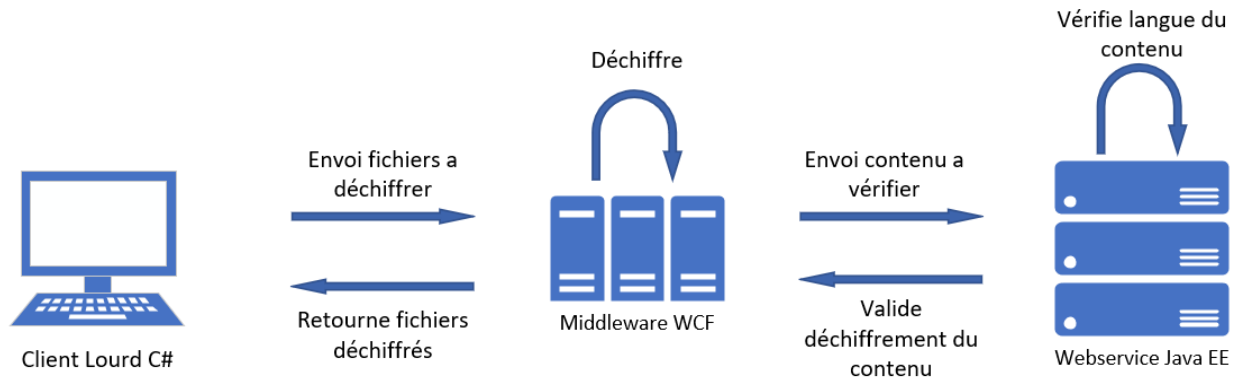


Diagramme d'architecture logicielle globale



## Architecture physique globale

Ci-dessous l'architecture physique mise en place pour réaliser la solution.

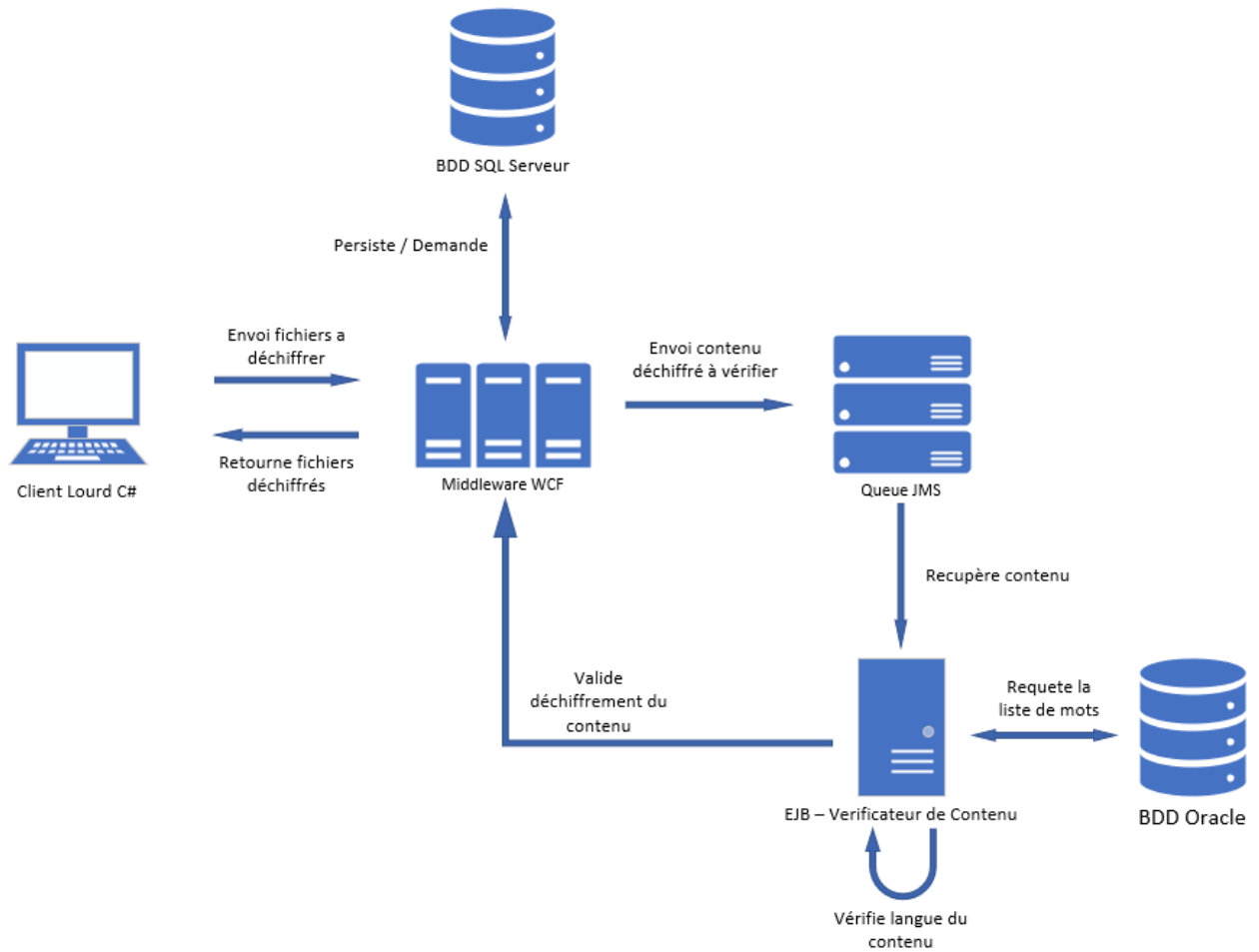


Diagramme d'architecture physique globale

## Modélisation diagramme de classes par couche et par plateforme

Ci-dessous les différents diagrammes de classe de la solution finale. De par leur taille liée à leur niveau de détails, ces diagrammes peuvent être peu lisibles. Nous vous invitons à aller consulter les différents fichiers disponibles en pièces jointes de ce document pour pouvoir les parcourir à votre guise.

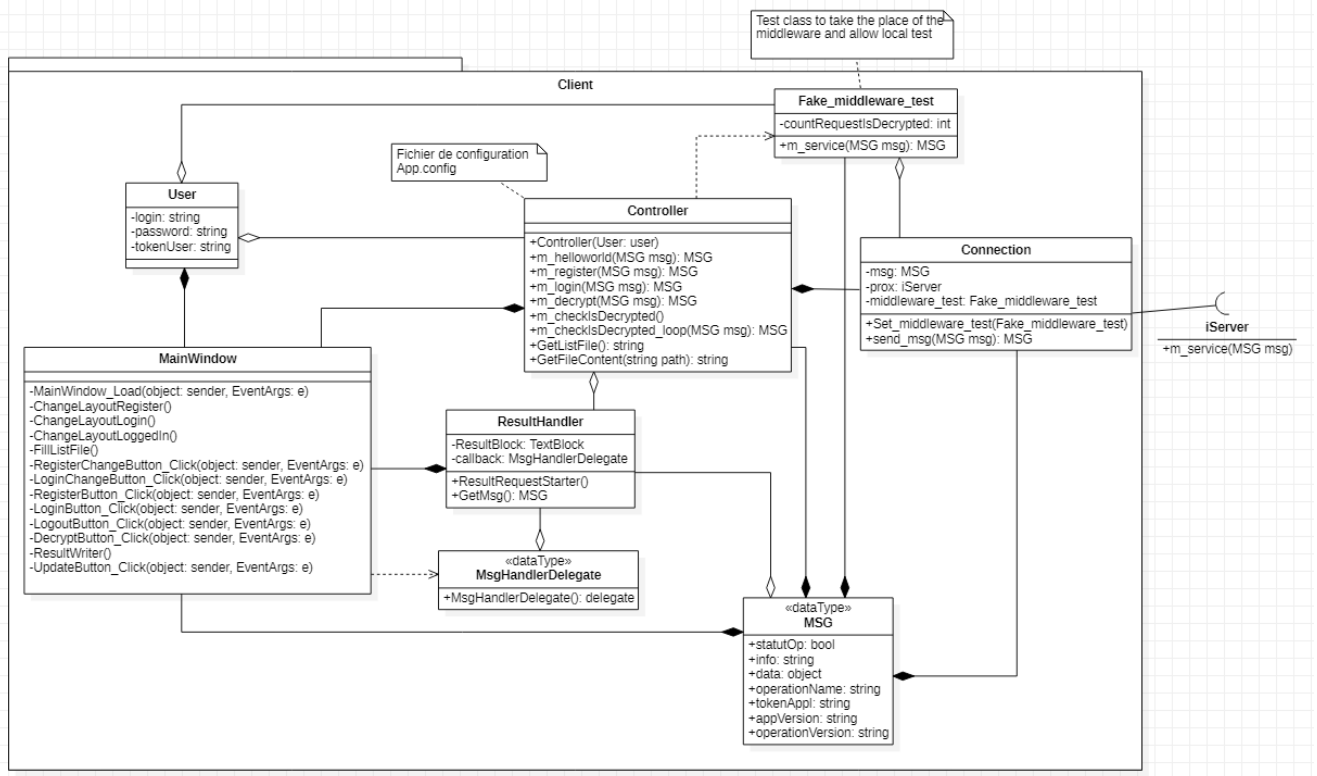


Diagramme de classe : Client Lourd

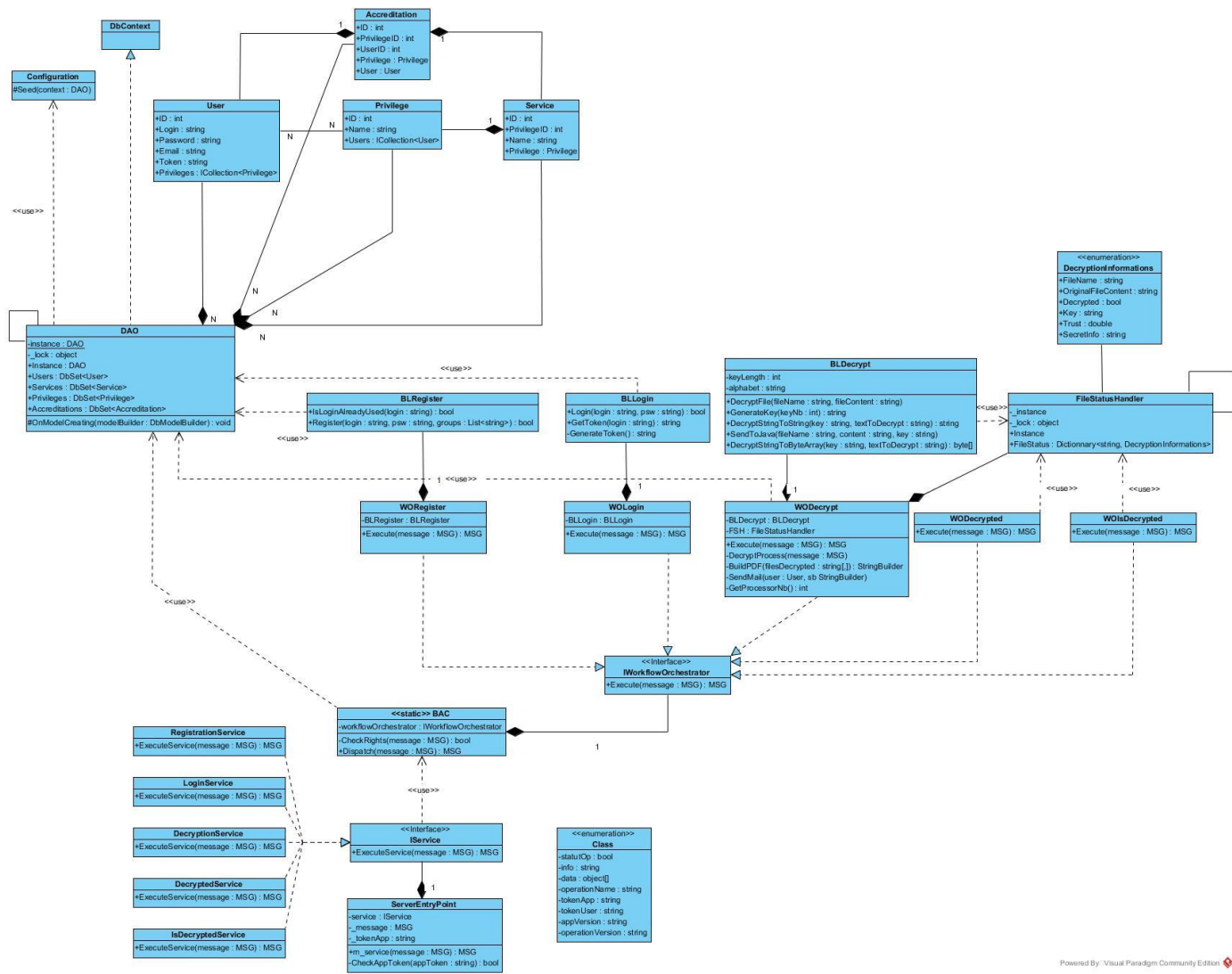


Diagramme de classe : Middleware C#

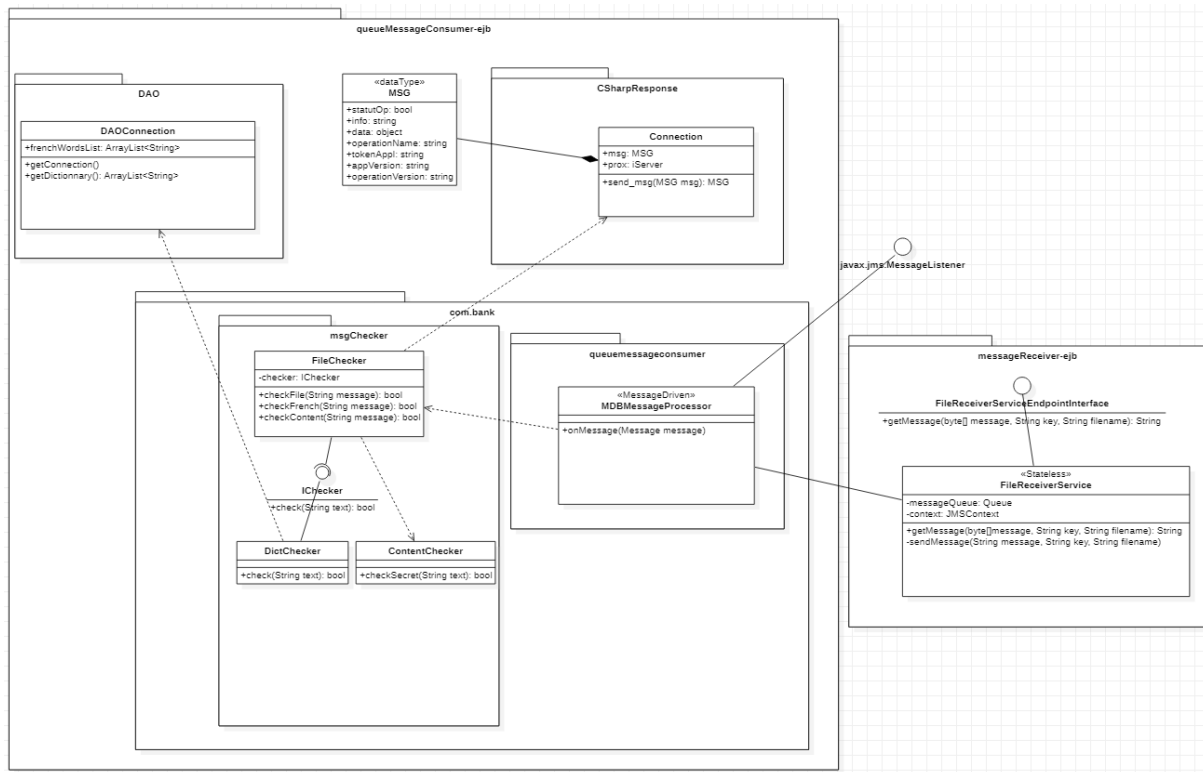


Diagramme de classe : Webservice Java EE

## Diagramme de cas d'activité

Ci-dessous le diagramme de cas d'activité représentant l'acteur et les actions qu'il peut réaliser via le Client Lourd.

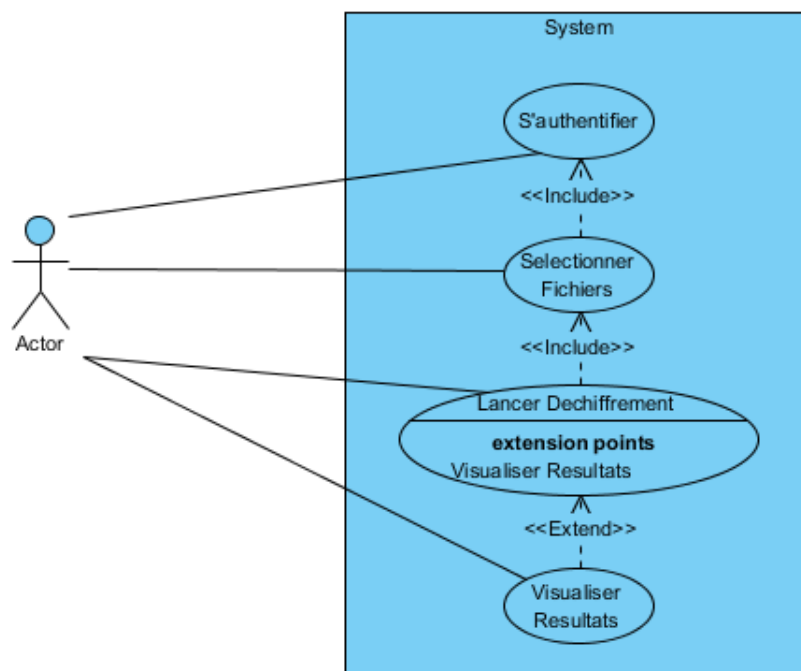


Diagramme de cas d'utilisation

## Diagramme de séquence par couche et par plateforme

Ci-dessous le diagramme de séquence système, représentant toutes les étapes du système complet.

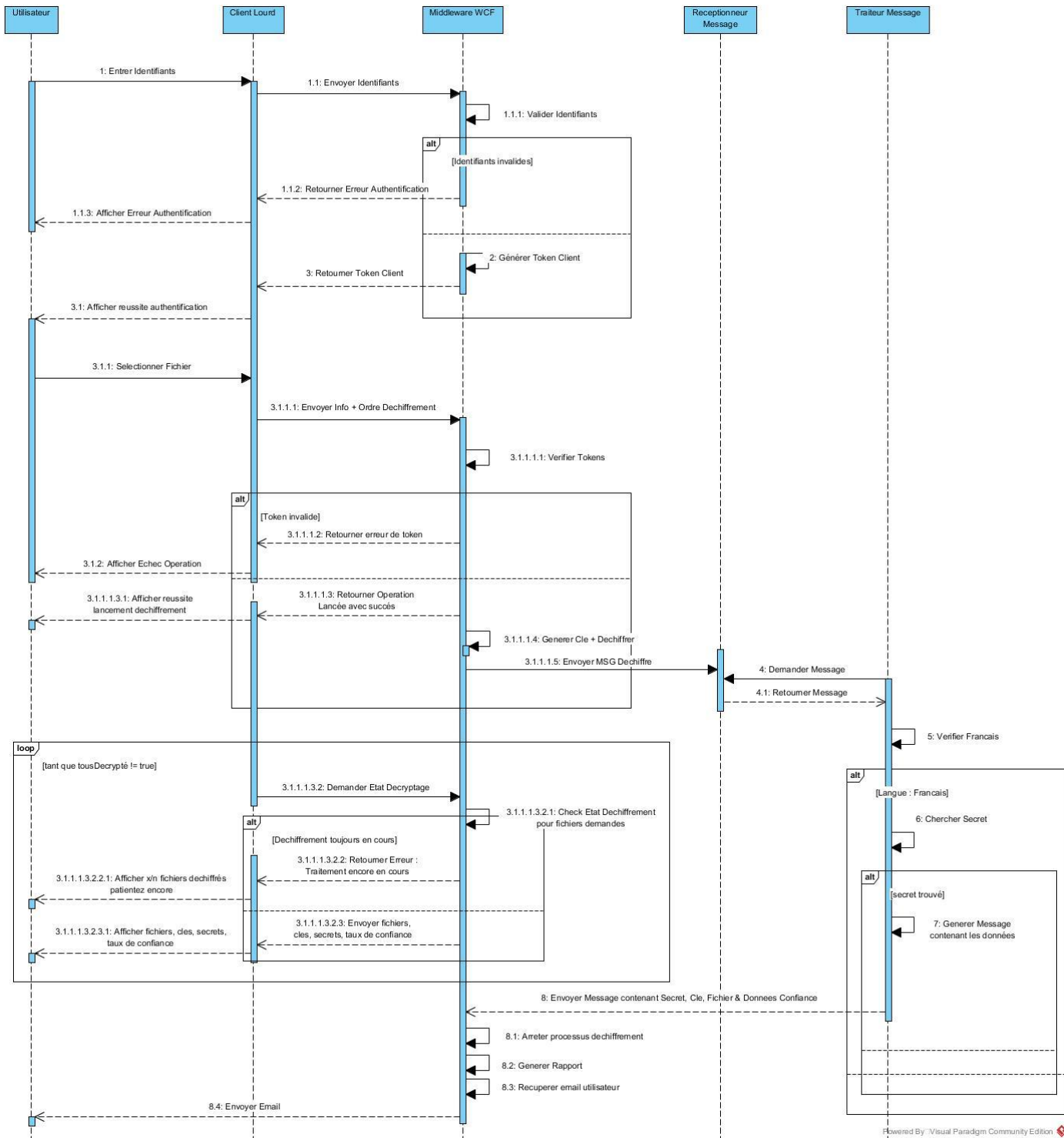
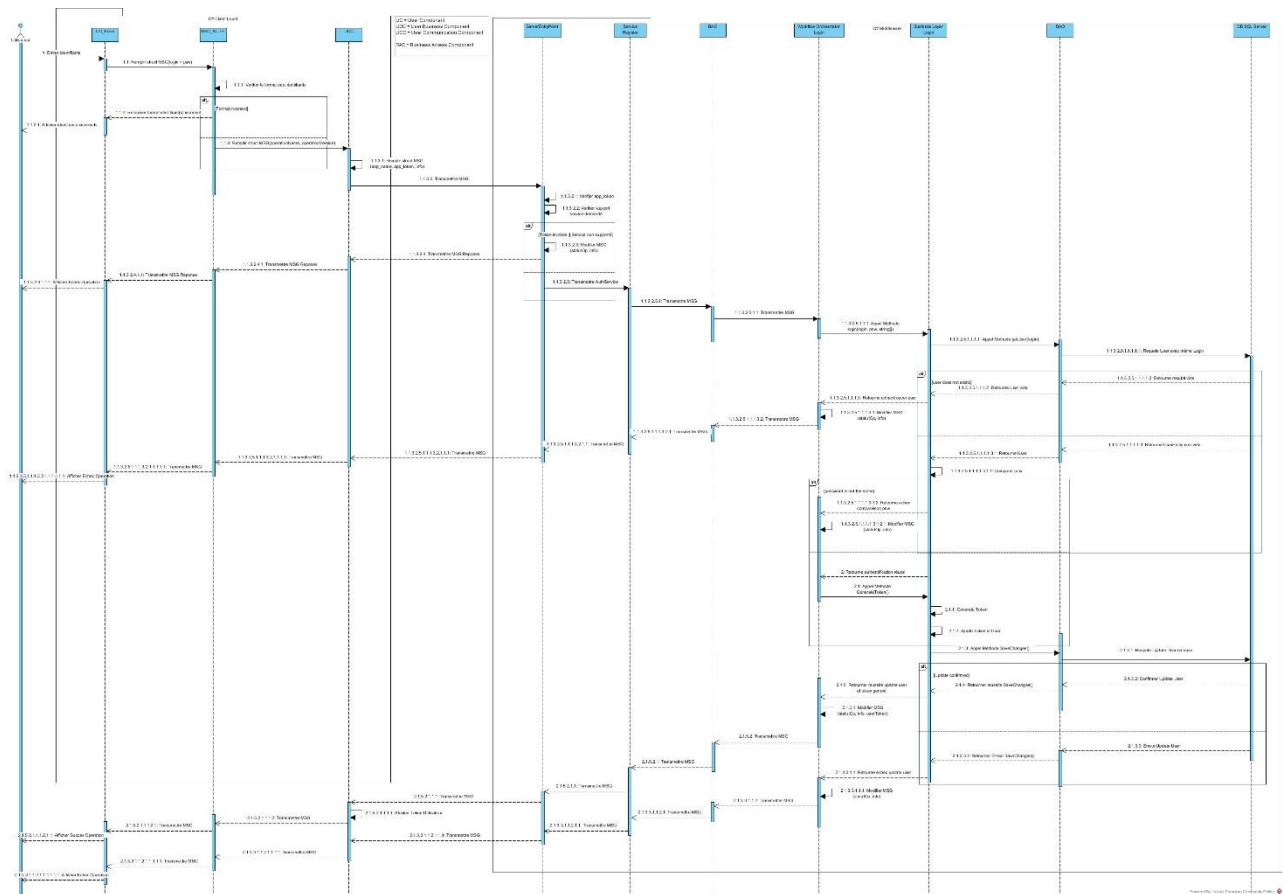


Diagramme de séquence système

Ci-dessous le diagramme de séquence étendu, représentant en détails toutes les étapes de la séquence de login. De par sa taille liée à son niveau de détails, ce diagramme est peu lisible. Nous vous invitons à aller consulter le fichier `Diagramme_Sequence_Etendu.jpg` pour pouvoir le parcourir à votre guise.



### Diagramme de séquence étendu - Login

## Analyse des écarts

Lors de la réalisation de ce projet, nous avons rencontré des difficultés à différents points de l'implémentation de la solution. C'est ce que nous allons présenter ici est l'analyse des écarts entre les attendus globaux et les rendus que nous fournissons.

L'un des écarts les plus importants est que nous n'avons pas pu ouvrir certains fichiers cryptés mis à notre disposition car ils étaient trop volumineux. Pour pallier ce problème nous avons encodé nos propre fichiers (file1.txt, file2.txt et file3.txt dans le dossier WCFClientDecrypt\WCFClientDecrypt\fileToDecrypt\)) qui sont plus léger pour pouvoir les utiliser dans notre programme. Ces fichiers permettent de tester le bon fonctionnement de l'application, à savoir la communication client lourd vers middleware ainsi que middleware vers JEE, l'envoi de message avec JMS et enfin la vérification du français et la recherche du secret.

Nous allons commencer par ce qui a été réalisé pour ensuite décrire et expliquer les écarts.

### Partie C# (client lourd)

Pour le client lourd C#, les tâches réalisées sont :

- S'inscrire
  - Lors du démarrage du client lourd, l'utilisateur doit s'authentifier mais s'il n'existe pas dans la base de données du middleware, il a la possibilité de s'inscrire via le layout désigné "Register".
  - Pour cela il fournit un login, un mot de passe et une adresse mail. Si ces trois entrées correspondent aux exigences de base (login correct, mot de passe de longueur et complexité suffisante, format d'adresse mail correct), le client envoie une requête spécifiée "Register" au middleware
  - Il doit recevoir ensuite un token Utilisateur qui lui permettra de communiquer avec le middleware et de vérifier son identité.
  - Il accèdera enfin au layout "LoggedInUser" où il pourra demander le déchiffrement des fichiers cryptés.
- Authentification
  - Si l'utilisateur existe déjà en base, l'utilisateur peut via le layout "Login" s'authentifier en entrant son login et son mot de passe
  - Le client envoie donc une requête "Login" au middleware qui doit lui renvoyer le token Utilisateur en cas de réponse positive à la requête.
  - Comme dans la tâche s'inscrire, après l'authentification, l'utilisateur peut commencer à demander le déchiffrement des fichiers cryptés.
- Déchiffrement
  - Après s'être authentifié l'utilisateur a accès à la sélection de fichier à déchiffrer (il peut spécifier la location du dossier les contenant à l'aide du fichier de configuration (app.config)).

- Suite à la sélection des fichiers, il n'a plus qu'à appuyer sur le bouton Decrypt pour lancer la demande de déchiffrement au serveur en envoyant le contenu des fichiers ainsi que le titre des fichiers, sans oublier d'inclure le token Utilisateur.
- Si la requête de déchiffrement est acceptée par le middleware, le client devrait recevoir une confirmation et il déclenche la tâche d'affichage.
- Affichage des résultats
  - Afin d'améliorer le client lourd et la partie de traitement des résultats, nous avons voulu ajouter une nouvelle fonctionnalité par rapport aux exigences de départ spécifié.
  - Nous avons réfléchi sur le problème du retour des résultats du déchiffrement au client. Si on se base sur l'architecture de départ du projet, soit le client lourd ne récupère pas directement les résultats, soit il doit rester bloqué jusqu'à ce que le middleware lui revoie les résultats.
  - Afin de régler ce problème, nous avons envisagé deux solutions :
    - Ajouter une partie "serveur" sur le client pour que le middleware puisse lui envoyer les résultats de son initiative (par exemple, un service REST). Lors de la requête de déchiffrement, le client lui enverrait une URL où le middleware pourrait lui communiquer les résultats.
    - Autrement, le service pourrait requêter le middleware en boucle avec un intervalle jusqu'à recevoir les résultats.
  - Nous avons décidé d'utiliser la deuxième solution afin de ne pas multiplier les points de communications, ceci dans l'intérêt de limiter la complexité de l'architecture et d'augmenter la sécurité en limitant les points d'attaque.
  - Lorsque le client lourd reçoit l'acceptation de la demande de déchiffrement par le middleware, le client lourd démarre un processus parallèle (thread parallèle) qui a pour mission d'envoyer des requêtes par intervalle de 30 secondes au middleware pour demander l'avancement du déchiffrement (requête "IsDecrypted"). Quand le déchiffrement est complété, cette requête est répondue avec confirmation ainsi que le rapport de déchiffrement. Le processus parallèle affiche ensuite le rapport dans une partie spécifique dédié de l'UI.
  - L'intérêt d'utiliser un processus parallèle est de permettre à l'utilisateur d'effectuer d'autre action au lieu de bloquer le client lourd à l'envoi de requête. Pour l'instant, le client ne peut que demander de faire des déchiffrements au middleware mais on pourrait penser que l'utilisateur pourrait avoir accès à d'autre fonctionnalité dans le futur.
- Vue du client lourd :
  - Le client lourd a été construit en WPF
  - Utilisation de thread pour le lancement de requêtes afin de ne pas bloquer l'utilisateur.

Dans l'ensemble, toutes les fonctionnalités exigées pour le client lourd ont été respecté.



## Partie C# (middleware)

Les tâches réalisées par le Middleware C# sont les suivantes :

- Authentification
  - Via le service 'Login', les utilisateurs ayant un compte dans la base de données peuvent se connecter et se voient retourner un Token client pour leurs prochains appels à un service.
- Déchiffrement
  - Via le service 'Decrypt', les utilisateurs ayant les droits d'accès suffisant peuvent envoyer un ou plusieurs fichiers à décrypter.
  - Une fois reçus, les fichiers sont répartis dans des threads afin d'augmenter la cadence de déchiffrement.
  - Des clés de déchiffrement au format "AAAA" sont générées de manière séquentielle et appliqués sur les fichiers.
  - A chaque application d'une clé, le contenu potentiellement déchiffré est envoyé au Webservice Java EE avec la clé utilisée et le nom du fichier.
  - Une fois tous les fichiers déchiffrés et authentifiés par le Webservice Java EE, un mail contenant le rapport de tous les fichiers initialement envoyés est généré puis envoyé par mail à l'utilisateur.
- Validation déchiffrement
  - Via le service 'Decrypted', le Webservice Java EE peut confirmer un déchiffrement correct et fournir la clé correspondante, la confiance en ce déchiffrement et l'information secrète si elle existe.
  - Ce processus entraine l'arrêt du thread traitant auparavant ce fichier, qui sera remplacé par un nouveau.
- Mise à jour Client Lourd
  - Via le service 'IsDecrypted', le Client Lourd peut demander une mise à jour de l'état de déchiffrement des fichiers initialement demandés.

## Partie JEE

Concernant le JEE, les tâches qui ont été réalisées sont :

- La réception de message C#
  - Grâce au webservice exposé à l'aide de JEE, le middleware C# a pu envoyer un message à la plateforme de réception JEE qui a ensuite pu récupérer ce message.
- Traduction pour Queue JMS
  - Une fois récupéré, le message a dû être traduit pour être envoyé dans une « JMS queue ».
  - Cette « JMS Queue » a donc dû être créée et associée au bon domaine.
- Création d'un Message Driven Bean (MDB) pour récupérer message
  - En effet pour pouvoir consommer les messages de la « JMS Queue » il a fallu créer un MDB qui peut récupérer les messages et ensuite l'envoyer dans les méthodes de traitement associées.
- Traitement des messages
  - Lorsqu'un message arrive, il y a un appel à la Base de Données (BDD), qui retourne une liste de mot français.

- Cette liste permet la comparaison de tous les mots présents dans le texte avec cette dernière. Cela aide selon un taux de confiance à savoir si le texte est bien déchiffré et donc est bien en français.
- Une fois qu'on trouve que le texte est bien en français, on peut essayer de trouver l'information secrète qui se cache si elle est présente.
- Affichage des résultats
  - C'est ici où il y a eu un écart. En effet, une fois la confirmation que le message est en français et éventuellement que le message secret a été trouvé, il fallait envoyer cela à la plateforme de réception C#. Au lieu de cela, qui n'est pas entièrement fonctionnel dans le programme, nous avons affiché les résultats, clés et nom du fichier dans la console lorsqu'un message est bien en français et un autre lorsque l'information secrète est trouvée. L'inconvénient est que le programme ne s'arrête pas de s'exécuter lorsque l'on trouve le message français.

## Analyse des compétences acquises par étudiants

### Pacôme Chu-Lejeune

J'ai pu au cours de ce projet approfondir mes compétences en C# notamment en ce qui concerne WPF (vue) et la programmation parallèle avec des threads ainsi que la combinaison des deux qui est légèrement particulière. J'ai aussi pu développer mes connaissances et compétences dans l'usage de Windows Communication Foundation. Participer dans la conception et la réalisation d'un projet utilisant l'architecture SOA et multiple de multiples technologies différentes a également été très bénéfiques pour moi.

### Robin Maisano

Ce projet m'a permis d'approfondir mes connaissances et d'acquérir de meilleures compétences en C# et surtout dans l'usage de Windows Communication Foundation.

### Guillaume Robert

Ce projet m'a permis d'approfondir mes connaissances sur le JEE en utilisant plusieurs fonctionnalités de cette technologie par exemple JMS pour envoyer et recevoir des messages à l'aide de « *Queue* », « *Message Driven Bean* », etc... Aussi grâce aux autres parties du projet j'ai pu approfondir mes connaissances sur le WCF et C#.

## Bilan

Malgré des connaissances parfois incomplètes pour la réalisation du projet, nous avons réussi à répondre à la plupart des besoins. On peut également dire que grâce à ce projet nous avons pu être en contact et mettre en place les bonnes pratiques de développement d'une application SOA. Le couplage faible grâce aux services est un réel avantage.