

Table des matières

[Introduction 3](#_Toc453568791)

[Description du projet 3](#_Toc453568792)

[Motivation 3](#_Toc453568793)

[Étude d'opportunité 3](#_Toc453568794)

[Cahier des charges 3](#_Toc453568795)

[Particularité du projet 3](#_Toc453568796)

[Matériel Utilisé 4](#_Toc453568797)

[Logiciels/fichiers utilisés 4](#_Toc453568798)

[Analyse fonctionnelle 4](#_Toc453568799)

[Préambule 4](#_Toc453568800)

[Point de vue du client/Administrateur 6](#_Toc453568801)

[fonctionnalités spécifiques aux administrateur 7](#_Toc453568802)

[Choix des composants 7](#_Toc453568803)

[Schéma des applications 8](#_Toc453568804)

[Analyse organique 8](#_Toc453568805)

[Choix du type de communication 8](#_Toc453568806)

[Diagramme de classes de l'application "CLient" 8](#_Toc453568807)

[Diagramme de classes de l'application "SERVEUR" 8](#_Toc453568808)

[Fichiers utiles pour l'application 8](#_Toc453568809)

[Du côté "Client" 9](#_Toc453568810)

[Du côté "Serveur" 9](#_Toc453568811)

[Méthode de communication des applications 10](#_Toc453568812)

[Descritpion des méthodes principales du Client 11](#_Toc453568813)

[Classe Client 11](#_Toc453568814)

[Classe UpdateUsers 12](#_Toc453568815)

[Classe FrmProgram 13](#_Toc453568816)

[Descritpion des méthodes principales du Server 14](#_Toc453568817)

[Classe ClientConnectToServ 14](#_Toc453568818)

[Classe RequestSQL 16](#_Toc453568819)

[Protocole de tests 16](#_Toc453568820)

[Conclusion 17](#_Toc453568821)

[Bilan personnel 17](#_Toc453568822)

[Difficultés rencontrées 17](#_Toc453568823)

[Difficultés non résolues 19](#_Toc453568824)

[Amélioration future 19](#_Toc453568825)

# Introduction

## Description du projet

Le but de ce projet est de créer une messagerie instantanée pour que les employés d'un même service, puissent communiquer entre eux.

## Motivation

Durant ma formation j'ai appris à utiliser différents langages de programmation, et j'ai choisi de réaliser mon projet en "C Sharp", car j'ai plus de facilité à l'utiliser qu'un autre langage.

Mon choix de créer une messagerie instantanée m'a été inspiré par mes camarades de classe.

Cela remonte à ma première année au CFPT : mes camarades et moi avions parlé de créer un site internet avec lequel nous aurions pu nous envoyer des messages durant les cours.

Nous avons essayé, mais nous nous sommes vite rendu compte que nous n'avions pas encore toutes les compétences nécessaires à sa réalisation.

En début d'année scolaire, nous en avons parlé à nouveau. J'ai alors décidé de me lancer ce défi pour mon TPI.

# Étude d'opportunité

Durant l'élaboration de ce projet, j’ai eu l’occasion d’aller me renseigner sur les différents logicielles proposé sur internet. Parmi eux, j'ai choisi de m'inspirer du logiciel Skype pour afficher la liste des Employés.

1 Affichage de la liste d'amis de Skype

Pour ce qui concerne l'affichage des messages, je me suis inspiré des informations qu'affichait l'application WhatsApp sur l'heure interface.

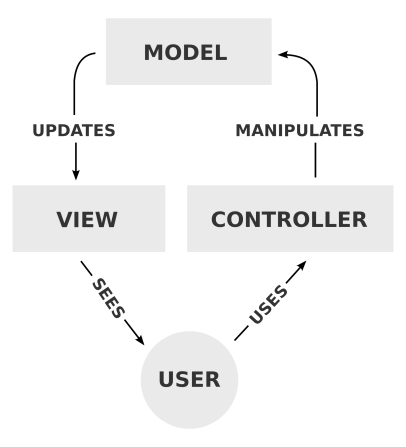
2 Affichages des informations relatives au utilisateur

Mon application va permettre d'afficher une liste des utilisateurs avec leur état de connexion, leur statue ainsi que le nombre de messages en absences.

# Cahier des charges

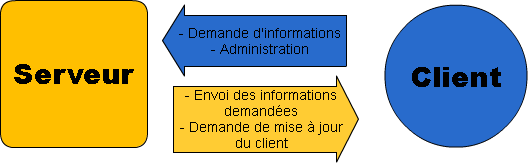
## Particularité du projet

Ce projet va être créé en C Sharp en Windows Form.

* Le programme va être réalisé en MVC (Modèle Vue Contrôleur).

Il y a 2 applications distinctes. Le côté serveur et le côté client

Concept du MVC



4 Explications des deux applications

## Matériel Utilisé

* PC Windows 7
* Disque dur externe
* Journal de bord (papier)

## Logiciels/fichiers utilisés

* Visual studio 2013
* Easyphp 14.1 VC9
* Suite office 2010
* Git hub desktop
* Cacoo.com
* mysql-connector-net-6.9.7
* WindowsBase.dll
* MysqlData.dll

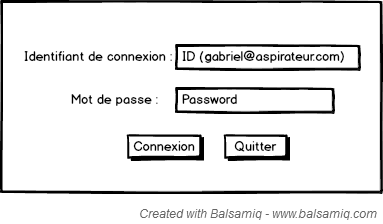
# Analyse fonctionnelle

## Préambule

Pour pouvoir utiliser cette application dans de bonnes conditions, plusieurs interfaces ont dû être créées.

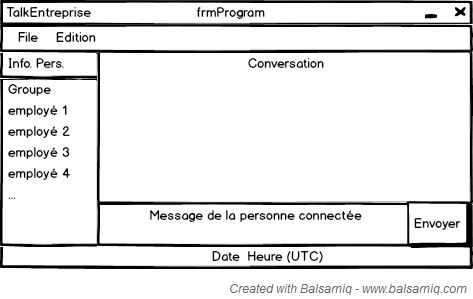
* Application "client"
  + Interface de connexion
  + Interface du programme
  + Interface de déconnexion
  + Interface "à propos"
  + Interface permettant de changer le mot de passe
* Application" serveur"
  + Interface de connexion
  + Interface du programme
  + Interface de déconnexion
  + Interface "à propos"
  + Interface permettant de changer le mot de passe
  + Interface permettant de modifier les informations des utilisateurs

## Point de vue du client/Administrateur



5 Interface de connexion

Cette interface permet à n’importe quel utilisateur de se connecter.



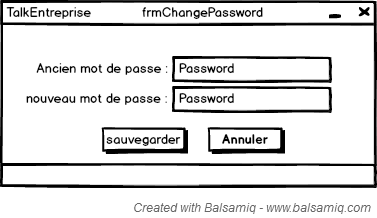
6 Interface du programme

Cette interface permet à l'utilisateur :

* d'envoyer un message
* d'accéder à la fenêtre de modification du

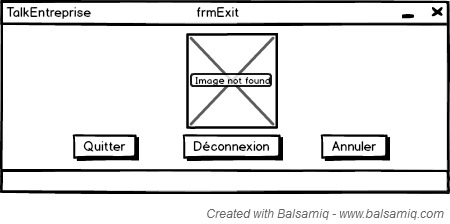
mot de passe

* de consulter la fenêtre "à propos"
* d'accéder à la fenêtre de déconnection.



7 Interface de modification du mot de passe

Cette interface permet à l'utilisateur la modification de son mot de passe.

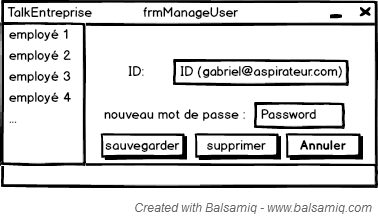


8 Interface de déconnexion

Cette interface permet à l’utilisateur :

* de quitter le programme
* de se déconnecter
* d’annuler la fermeture du programme.

## fonctionnalités spécifiques aux administrateur



9 Interface permettant la modification des utilisateurs ainsi que leur suppression

Cette interface permet à l'administrateur de modifier/supprimer un utilisateur. Elle est accessible depuis l'interface du programme **(voir image X)**

## Choix des composants

Mon programme est principalement composé de "textbox".

J'ai choisi ce composant car grâce à lui je peux récupérer les informations tapées par l'utilisateur.

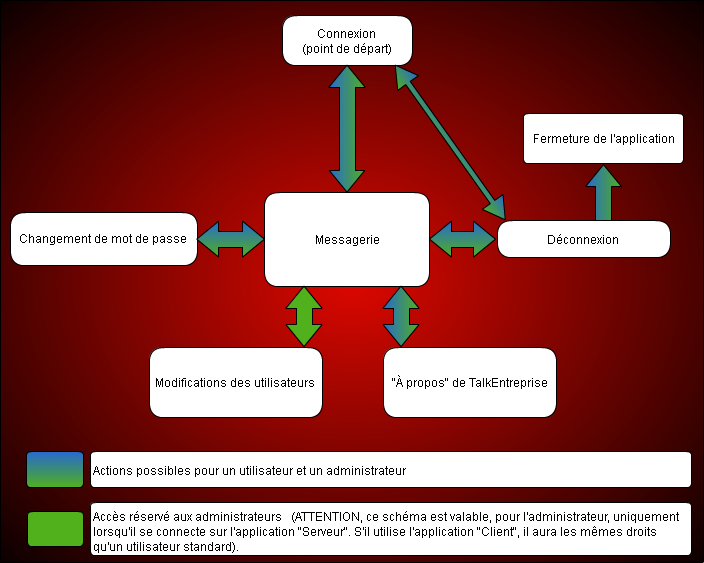
Je peux également décider de modifier, à tout moment, l'aspect visuel des messages dans une conversation.

Ce composant a la particularité d'adapter le texte à sa taille.

Pour l'affichage des utilisateurs, j'ai décidé d'utiliser une "listbox", car elle permet d'afficher plusieurs informations.  
Dans mon programme, je l'ai utilisée pour afficher certaines informations concernant les utilisateurs :

* Leur état de connexion
* Leur identifiant
* Leur statut (si administrateur)
* Leur nombre de messages non lus

## Schéma des applications



10 Schéma de l'application du point de vue du "Client" et de "l'Administrateur"

# Analyse organique

## Choix du type de communication

Lors de l'élaboration de mon projet, mon intention était de créer une application "Client" qui allait chercher les données automatiquement sur la base de données.

Mais après avoir discuté avec Monsieur Garcia, mon référent de TPI, de la manière de récupérer les informations présentes dans la base de données, il m'a conseillé d'utiliser l'envoi de données via le réseau.

Après réflexion, j'en ai conclu que l'envoi de donnés à travers le réseau était la meilleure solution.

## Diagramme de classes de l'application "CLient"

Vous trouverez le diagramme de classes en annexe 1

## Diagramme de classes de l'application "SERVEUR"

Vous trouverez le diagramme de classes en annexe 2

## Fichiers utiles pour l'application

### Du côté "Client"

#### Les vues :

* FrmConnections.cs
* FrmProgram.cs
* FrmSettings.cs (modification du mot de passe)
* FrmExit.cs (choix de la déconnection : "se déconnecter" ou "quitter l'application")
* FrmAbout.cs

#### Les classes :

* Controler.cs
* Client.cs
* ManageMessages.cs
* Message.cs
* User.cs
* UpdateUser.cs

### Du côté "Serveur"

#### Les vues :

* FrmConnection.cs

#### Les classes :

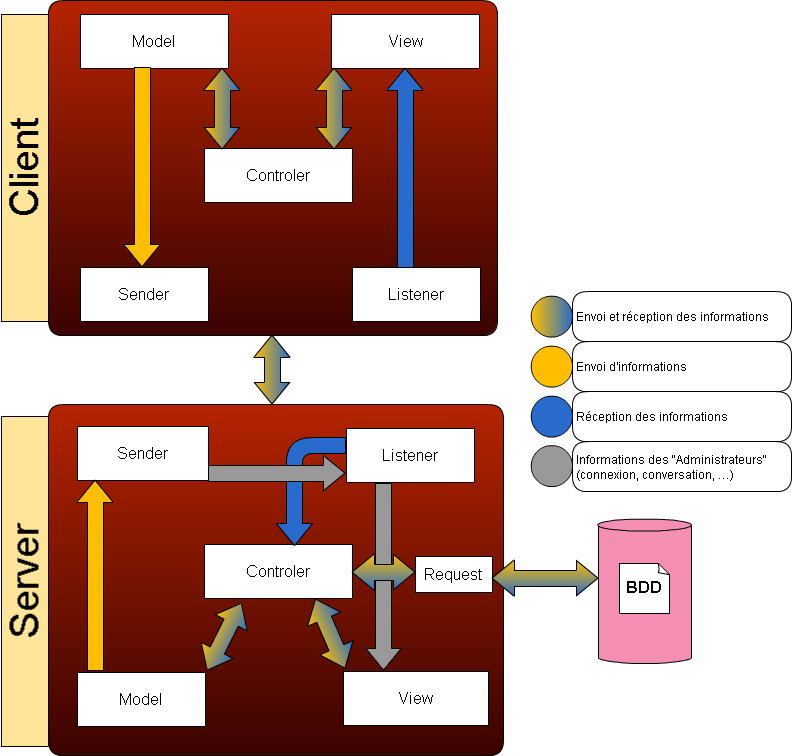
* Controler.cs
* ClientConnectToServ.cs
* UpdateUser.cs
* Message.cs
* RequestSQL.cs
* Server.cs
* User.cs

## Méthode de communication des applications

Les informations sont communiquées selon le schéma suivant :

Pour pouvoir échanger des messages entre les différentes applications ("Client" et "Serveur"), j'ai décidé de mettre un code de début et un code de fin.

Schéma de communication des applications

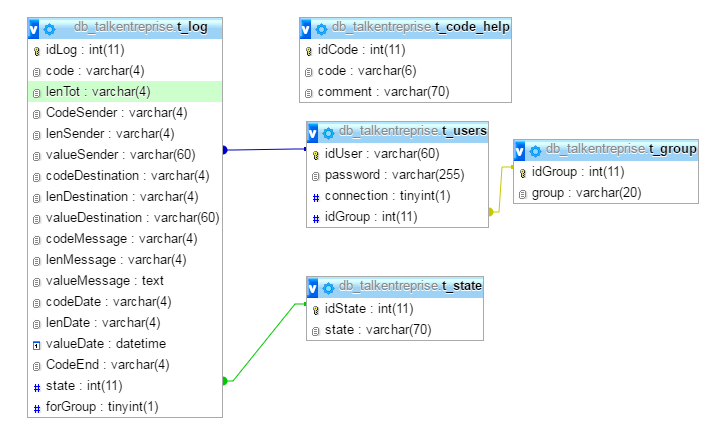


Le message commence par le caractère "#" suivi de 4 chiffres et finit par 4 "#". Le reste du message est généralement séparé par des "point-virgule", par des "tirets" ou par des "virgules". (Exemple : #0001;gabriel@aspirateur.com;motdepasse####)

|  |  |  |
| --- | --- | --- |
| Code d'identification | Signification | |
| #0001 | Permet de vérifier si l'utilisateur se trouve dans la base de données | |
| #0002 | Permet à l'utilisateur de se déconnecter | |
| #0003 | Permet d'envoyer le message rédigé par un utilisateur au serveur, pour l'enregistrer dans la base de données | |
| #0004 | Demande au serveur de lui donner les messages d'une conversation  Récupération des messages de la conversation | |
| #0005 | Envoi des informations de l'utilisateur au serveur | |
| #0006 | Mise à jour de l'état des messages de l'utilisateur | |
| #0007 | Demande au serveur de lui donner la liste des anciens messages  Récupération des anciens messages | |
| #0008 | Demande au serveur d'enregistrer le nouveau mot de passe de l'utilisateur  Récupération de la réponse du serveur | |
| #0015 | Récupération de la liste d'utilisateurs envoyée par le serveur |

## Base de données

### Model Conceptuel de données

Lors de la création de la base de données, j'ai dû modifier certaines tables car cela m'a permis d'éviter le traitement de données inutiles dans mon application.

Les explications concernant les différents champs de la base de données sont données en annexe.

## Descritpion des méthodes principales du Client

### Classe Client

Cette classe est importante, car grâce à elle, l'application "Client" peut demander des informations au serveur.

La plupart des méthodes se ressemblent, c'est pourquoi je ne les citerais pas toutes.

#### ResetConnection()

Cette méthode permet d'initialiser la communication entre le "Client" et le "Serveur".

ResetConnection()

{

Tant qu'il n'y a pas d'erreur

{

Création d'un nouveau client TCP

Création d'un nouveau canal de communication

Connection au serveur qui se trouve à l'adresse 127.0.0.1 et qui est branché sur le port 88888

Retourne oui

}

Sinon

{

Retourne non

}

}

#### ConnectionServer(…)

Cette méthode permet de demander au serveur si l'utilisateur se trouve dans la base de données.

ConnectionServer(…)

{

Attendre 10 millisecondes

Création un tableau de byte de 10025 cases

Création du message à envoyer

Transformer le message en tableau de byte

Envoi du message au serveur

Supprimer l'historique d'envoi

Récupération de la réponse du serveur

Transformer le tableau envoyé par le serveur en message

Retourne oui ou non

}

### Classe UpdateUsers

Cette classe permet uniquement de récupérer les informations émises par le serveur.

Elle est composée d'une unique méthode, dans laquelle on retrouve différentes conditions.

À nouveau, la plupart des conditions se ressemblent, je n'en décrirai qu'une seule.

#### Init()

Init(…)

{

Initialisation de variables utiles pour gérer les différentes conditions

Tant que la condition "True" est vraie alors

{

Efface la liste "lstUser"

Si lors de l'exécution du code il n'y a pas d'erreur alors

{

Récupération d'une information sur le réseau

Décodage de cette information et mise du résultat dans la variable "result"

Modification de la variable "result" en lui enlevant tout ce qui existe après les caractères suivant "####"

}

Sinon

{

Mise de la valeur de la variable "stop" à "True"

}

Si la valeur de la variable "stop" est à "True" alors

{

Sortie de la boucle Tant que

}

Si un des cas contient la suite de caractères que contient la variable "result" alors

{

Si le cas contient la chaîne de caractères suivante "#0008" alors

Lancement de la méthode nommée "PasswordIsChanged" avec comme arguments la chaîne de caractères qui contient "True" ou "False" et le mot de passe qui se trouve dans la variable "result"

Arrêt de la recherche de condition

S'il n'y a pas de condition valide alors

Mise de la valeur de stop à "True":

}

}

### Classe FrmProgram

Cette classe permet de gérer la fenêtre de la messagerie.

#### ShowMessages(…)

Cette méthode permet d'afficher les messages à l'utilisateur.

ShowMessages(…)

{

Création d'une méthode sécurisée

{

Création du variable de type "User" nommée "user" qui a comme valeur l'utilisateur sélectionné dans la liste "lstEmployees"

Si la variable "user" contient bien un utilisateur alors

{

Si le ou les messages envoyés par un utilisateur est/sont pour la conversation actuellement affichée sur le programme alors

{

Création d'une variable nommée "message" qui a comme valeur une chaîne de caractères vide

Tant que le nombre d'ancien message n'est pas égal au nombre de nouveaux messages

{

Création d'une variable nommée "msg" qui contient un des messages de la liste "lstNewMessages"

Si l'auteur du message n'est pas le même que celui qui se trouve dans "msg" alors

{

Ajout du nom de l'auteur dans la variable "message"

}

Sinon

{

Ajout du message ainsi que la date à la variable "message"

}

Changement du nombre de message de la variable "NbMessages" par le nombre de message présent dans la liste "lstNewMessages"

Ajout au composant "tbxMessages" la variable "message"

}

}

Sinon

{

Appel de la méthode "ServerClosed"

}

}

}

Fin de la méthode sécurisée

}

## Descritpion des méthodes principales du Server

### Classe ClientConnectToServ

Cette classe permet d'enregistrer un utilisateur qui s'est authentifié. Elle permet aussi d'envoyer des informations à/aux utilisateur-s concerné-s.

#### Init()

Cette méthode permet d'enregistrer les différents clients connectés.

Init()

{

Création d'un objet de type "TCpListener" qui se nomme "serverSocket". Sa fonction est d'écouter les demandes de connexion sur le port "8888"

Création d'un objet nommé "ClientSocket" qui contiendra un client par défaut.

Activation de la méthode "Start" sur l'objet "serverSocket" pour qu'il puisse réceptionner les connexions.

Tant que la condition "True" est vraie alors

{

L'objet "ClientSocket" contient le client que l'objet "serverSocket" lui donne.

Création et initialisation de plusieurs variables.

Création d'un objet de type "NetworkStream" nommé "networkStream" qui contient le canal de transfert de données entre le serveur et le client

Récupération des informations envoyées par le client

Décodage du message envoyé par le client

Si le message contient la chaîne de caractères suivante : "#0001" alors

{

Séparation du message par le caractère ';' et stockage de l'identifiant de connexion dans la variable "user" et le mot de passe dans la variable "password"

}

Si la méthode de validation retourne "True" alors

{

Appel de la méthode "SucessConnectionServer"

La variable "sendToClient" a maintenant comme valeur "True"

Ajout du client au tableau des clients qui se trouve dans la classe "Server".

}

Attente de 10 millisecondes

Envoi au client si la connexion a réussi

Si la connexion a réussi

{

Enregistrement dans la liste "userInfo" des informations relatives à l'utilisateur

Envoi des informations relatives à l'utilisateur, au client.

Création d'un nouveau processus qui permet de gérer les demandes que fait cet utilisateur

Ajout de ce processus dans la liste nommée "lstThreads" qui se trouve dans la classe "Serveur"

Activation du processus

}

} Fin du Tant que

Fermeture de l'écoute sur le réseau

}

### Classe RequestSQL

Cette classe contient toutes les requêtes qui permettent de récupérer les informations qui se trouvent dans la base de données.

#### ConnectionDB()

ConnectionDB(…)

{

Si lors de l'exécution du code il n'y a pas d'erreur alors

{

Information de connexion à la base de données

Création du lien entre la base de données et l'application

Ouverture de la connexion

Retourne "True"

}

Sinon

{

Retourne "False"

}

}

#### ValidateConnectionUser(…)

ValidateConnectionUser(…)

{

Création d'une variable de type "boolean" nommée "result" qui est par défaut à "False"

Si la méthode "ConnexionDB" retourne "True" alors

{

Préparation de la requête

Exécution de la requête qui a été créée

Création d'un objet "MySqlDataReader" qui a comme nom "reader" et qui va contenir la réponse donnée par la base de données.

Si l'utilisateur se trouve dans la base de données alors

{

La valeur de la variable "result" va passer à "True"

}

Fermeture de l'objet "MySqlDataReader" avec la méthode "Close"

Fermeture de la connexion "MYSQL" en utilisant la méthode "ShutdownConnectionDB"

}

Retourne la valeur de la variable " result "

}

# Protocole de tests

Vous trouverez les tests en annexe.

# Conclusion

## Bilan personnel

Ce projet m'a permis de mettre à l'épreuve ma manière de travailler.

Au commencement de mon TPI, j'étais un peu perdu, mais grâce aux conseils de Monsieur Garcia, j'ai réussi à structurer mes idées et à avancer.

Il m'a suggéré de faire une décomposition de ma méthode de connexion sur un fichier Excel.

De cette manière, j'ai pu structurer correctement mon programme et ainsi le créer.

D'une part, des moments difficiles, chargés de mécontentement se sont présentés à moi. En effet, lorsque certaines parties de l'application ne fonctionnaient pas ou lorsque du jour au lendemain le code présentait une erreur, il était compliqué de garder le cap. Mais heureusement, mon envie de comprendre m'a permis de trouver les solutions à la plupart des problèmes rencontrés.

D'autre part, j'ai éprouvé une grande satisfaction, car j'ai pu apprendre de nouvelles choses, en faisant des recherches par moi-même. Cela m'a permis d'enrichir mes connaissances dans le domaine du CSharp.

J'ai également appris à prendre l'essentiel et à mettre de côté certaines choses que j'aurais voulu faire, car je devais respecter les délais qui étaient fixés et gérer mon temps.

Pour conclure, je tiens à adresser mes remerciements aux personnes qui m'ont aidé et soutenu pendant cette période d'examen.

En premier lieu, je remercie Monsieur Francisco GARCIA, mon professeur référent, qui m'a guidé et permis de me surpassé dans la réalisation de mon projet.

Je remercie également mes camarades de classe pour leur aide psychologique et la bonne ambiance qui m'a permis d'aller de l'avant.

## Difficultés rencontrées

Au cours de ce projet, j'ai rencontré un bon nombre de problèmes, principalement lors de l'utilisation de certaines fonctionnalités qui n'ont jamais été abordées durant ma formation.

Lors de ma première entrevue avec mon référent, Monsieur Garcia, nous avons discuté de la manière dont "l'application client" allait pouvoir chercher les informations dans la base de données.

En ce qui me concerne, avant l'entretien, j'avais l'intention de faire en sorte que "les utilisateurs" se mettent à jour automatiquement en allant chercher les informations directement dans la base de données.

Monsieur Garcia m'a proposé d'utiliser "l'application serveur" pour mettre à jour toutes "les applications client".

Bien que je n'avais jamais utilisé l'envoi de données via le réseau, j'ai décidé de relever ce défi.

J'ai alors commencé a cherché un site qui me permettrait d'apprendre comment envoyer des informations sur le réseau.

En cherchant des exemples de messageries instantanées, j'ai découvert un site[[1]](#footnote-1) qui expliquait la démarche pour envoyer des informations via le réseau avec une démonstration.

En utilisant le mode "pas à pas" sur Visual Studio, j'ai pu comprendre comment cela fonctionnait.

Mon second problème a été de lancer la fenêtre du programme dans un nouveau processus.

J'ai essayé de lancer cette fenêtre avec l'objet "Thread", mais cela ne fonctionnait pas.

En essayant certains exemples de code trouvés sur le web, mais n'ayant aucun résultat, j'ai consulté des forums et plusieurs d'entre eux parlaient d'utiliser l'objet "Dispatcher".

Mais Visual Studio n'acceptait pas le code pris sur le net.

J'ai alors pensé qu'il manquait une référence[[2]](#footnote-2), comme pour l'utilisation de MYSQL dans Visual Studio.

En cherchant plus loin, j'ai compris qu'il fallait bel et bien ajouter une référence. Le "dll" manquant était "WindowsBase.dll".

J'ai ensuite rencontré une difficulté supplémentaire lors de l'utilisation de l'objet "Thread".

Lors de la création de ma méthode d'affichage des "employés" présents dans ma base de données, les informations émises par le serveur étaient récupérées par "l'application client" et retransmises à mon objet "ListBox".

Lorsque j'ai lancé le programme, une erreur m'indiquait qu'il n'est pas possible de modifier un composant depuis un autre processus.

En recherchant mon problème sur le net, j'ai trouvé le moyen de donner ma liste "employés" à ma "ListBox" en utilisant l'objet "Invoke". Ce dernier permet d'encapsuler (de sécuriser) des méthodes et de pouvoir modifier les informations de composants. Je l'ai alors utilisé.

Une des fonctionnalités de mon application permet à un utilisateur d'envoyer un message à chaque personne présente dans son propre groupe. C'est lors de la récupération de la conversation d'un groupe qu'une autre complication s'est présentée.

En sachant que dans la base de données, les messages de groupe sont stockés de manière à envoyer le même message à chaque utilisateur du même groupe avec un intervalle de quelques millisecondes; lors de la récupération de la conversation, j'utilise le paramètre "DISTINCT", qui permet d'enlever les doublons.

Mais lorsqu'un utilisateur envoie un message à son groupe, dans la base de données, l'enregistrement se fait à intervalle de quelques millisecondes. Cela suffit parfois à ce que le message soit enregistré à un horaire précis et une seconde après (ex. 18:15:56 et 18:15:57). Pour l'utilisateur, ce décalage se traduit par l'affichage du message à double.

Pour pallier à cette complication, j'ai décidé de créer une autre méthode. Elle permet d'envoyer au serveur la liste de tous les utilisateurs. A partir de là, le serveur enregistre instantanément tous messages au même horaire. Ce qui a résolu mon problème.

Encore un problème, lorsqu'un utilisateur s'est déconnecté et se reconnecte une seconde fois, un message l'informe qu'il va être déconnecté car le serveur est éteint (ce qui n'est pas le cas).

Un second problème, lorsque l'utilisateur veut quitter l'application, un message d'erreur (qui n'apparaît pas lors de l'exécution de l'application sur Visual Studio) indique qu'il n'est pas possible d'afficher la fenêtre de connexion car elle n'existe plus.

Cela ce produit car la fenêtre principale du programme est visuellement fermée, mais dans l'objet "Dispatcher" cette fenêtre est toujours active.

Pour la résolution des deux problèmes, j'ai ajouté à l'évènement "Close" de ma fenêtre principale, un bout de code qui a permis de supprimer définitivement le processus de cette fenêtre.

Une autre difficulté est apparue, lorsque j'ai voulu créer la méthode de récupération des anciens messages, car je ne savais pas comment m'y prendre.

J'ai créé une méthode qui récupère les anciens messages et qui les stocke dans la une liste.

Avant d'afficher la conversation à l'utilisateur, la liste des anciens messages est concaténée à la nouvelle liste de messages.

## Difficultés non résolues

Depuis que j'ai commencé à coder l'application, il y a des petites erreurs sans gravité, qui apparaissent.

Lors de la première connexion, lorsque l'on veut envoyer un message à un autre personne (pour la première fois), il faut cliquer deux fois sur un la personne en question pour la sélectionner.

Lors de la connexion, il arrive parfois que l'affichage de la conversation n'apparaisse pas ou seulement une partie des messages sont affichés.

## Amélioration Envisageable

* Optimisation de l'envoi de données entre le serveur et clients.
* Correction des problèmes non résolues.
* Les administrateurs peuvent envoyer des messages aux différents groupes.

1. Nom du site dans la bibliographie [↑](#footnote-ref-1)
2. Librairie de fichiers ".dll" qui permet l'utilisation de différents types d'objets [↑](#footnote-ref-2)