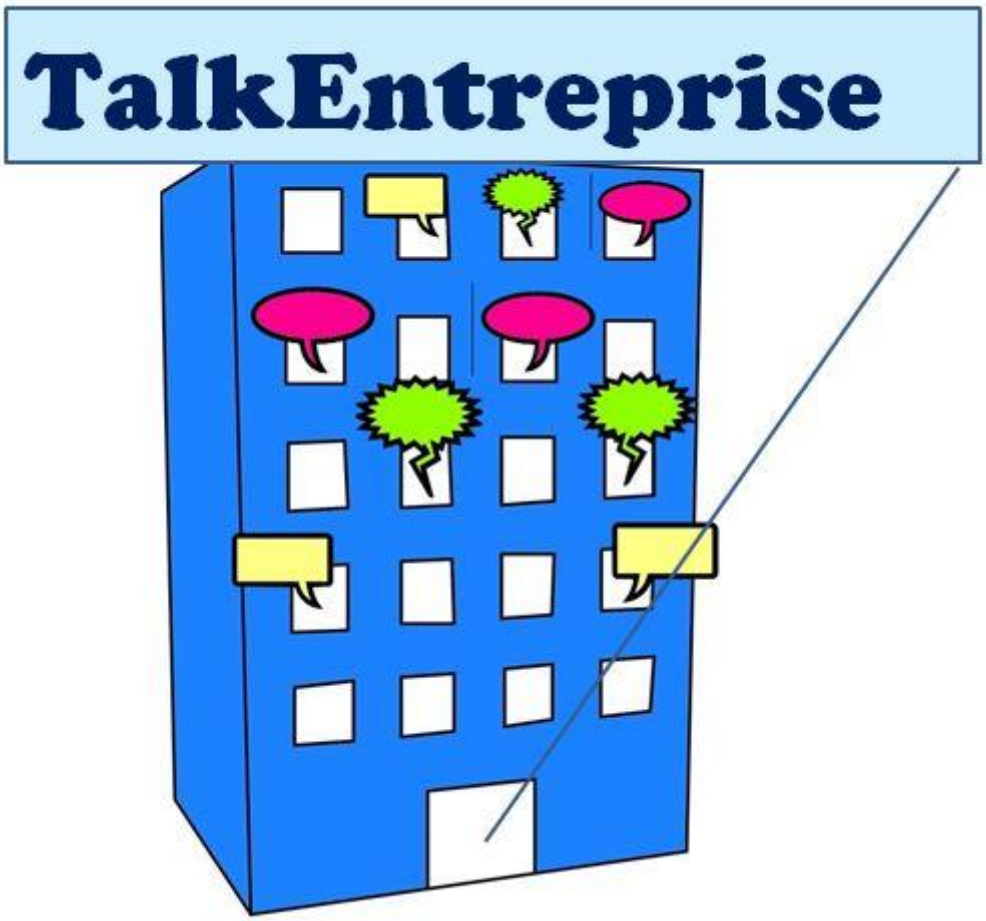
Gabriel STRANO Classe I.FA-P3A

Travail Pratique Individuel



Centre de Formation Professionnelle Technique

Juin 2016

Professeur référent : Monsieur Francisco GARCIA

Table des matières

1 INTRODUCTION

1.1 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.

1.2 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.

1.3 EXPLICATION DE CERTAINS TERMES

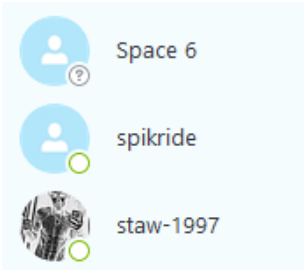
"Client" est utilisé pour parler de l'application du client.
Les employés utilisant l'application "Client" sont nommés "Utilisateurs".

L'application du serveur est composée de deux parties :

- ✓ "Serveur" qui sert à gérer les demandes des applications "Client"
- ✓ "Administrateur" qui permet de gérer les informations liées aux "Utilisateurs"

2 ÉTUDE D'OPPORTUNITÉ

Durant l'élaboration de ce projet, j'ai eu l'occasion d'aller me renseigner sur les différents logiciels proposés sur internet.
Parmi eux, j'ai choisi de m'inspirer du logiciel "Skype" pour l'affichage de la liste des "Utilisateurs",



1 Affichage de la liste d'amis de "Skype"

et de l'application "WhatsApp" pour l'affichage des informations des messages.

2 Affichages des informations relatives aux utilisateurs

L'interface de mon application permettra

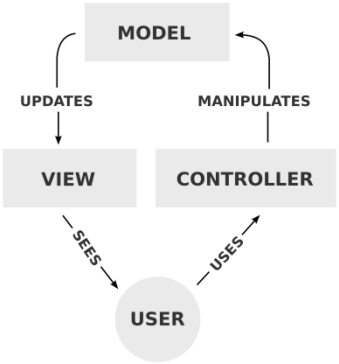
- ✓ d'afficher une liste avec le nom des employés d'un même service
- ✓ leur état de connexion
- ✓ leur statut
- ✓ le nombre de messages non lus

3 CAHIER DES CHARGES

3.1 PARTICULARITÉ DU PROJET

Ce projet est créé en C Sharp en Windows Form.

- L'application est réalisée en MVC (Modèle Vue Contrôleur).



3 Concept du MVC

Il y a 2 applications distinctes. Le côté "Serveur" et le côté "Client" :

4 Explications des deux applications

3.2 MATÉRIEL UTILISÉ

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

3.3 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

3.4 PLANNING

3.4.1 Initial

Gabriel Strano
IFA-P3A
Planning TPI

Activité	02.06.2016	03.06.2016	04.06.2016	05.06.2016	06.06.2016	07.06.2016	08.06.2016	09.06.2016	10.06.2016	11.06.2016	12.06.2016	13.06.2016	14.06.2016	15.06.2016
Mise au point de l'application à créer			weekend							weekend				
Création interface + Classes														
méthode de Connexion + déconnexion														
Quitter l'application														
méthodes liées à la communication														
Application dite "Serveur"														
Tests + Rapport de tests														
Splash Screen														
Journal de bord														
Documentation														

5 Planning fixé au début du projet

3.4.2 Final

Gabriel Strano
IFA-P3A
Planning TPI

Activité	02.06.2016	03.06.2016	04.06.2016	05.06.2016	06.06.2016	07.06.2016	08.06.2016	09.06.2016	10.06.2016	11.06.2016	12.06.2016	13.06.2016	14.06.2016	15.06.2016
Mise au point de l'application à créer			weekend							weekend				
Création interface + Classes														
méthode de Connexion + déconnexion														
Quitter l'application														
méthodes liées à la communication														
Application dite "Serveur"												si doc terminée		
Tests + Rapport de tests														
Splash Screen														
Journal de bord														
Documentation														

6 Planning réalisé

4 ANALYSE FONCTIONNELLE

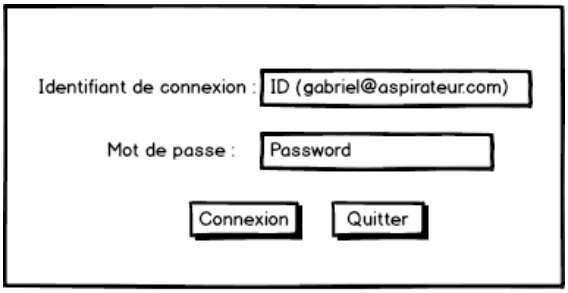
4.1 PRÉAMBULE

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

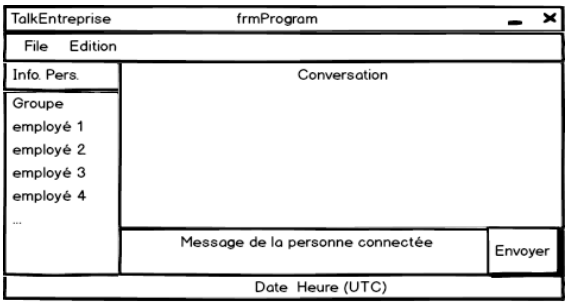
- Application" Serveur"
    - Interface de connexion
    - Interface du programme
    - Interface de déconnexion
    - Interface "à propos"
    - Interface permettant de changer le mot de passe
    - Interface permettant de modifier les informations des "Utilisateurs"

4.2 DU POINT DE VUE DU "CLIENT" ET DE L'"ADMINISTRATEUR"



7 Interface de connexion

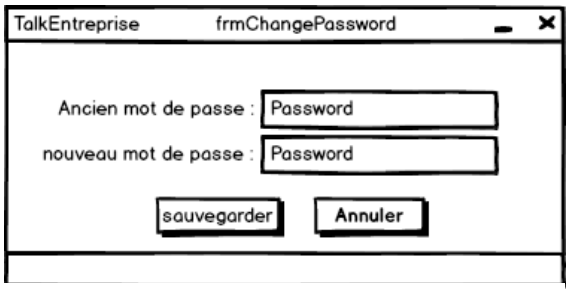
Cette interface permet à n'importe quel "Utilisateur" de se connecter.



8 Interface du programme

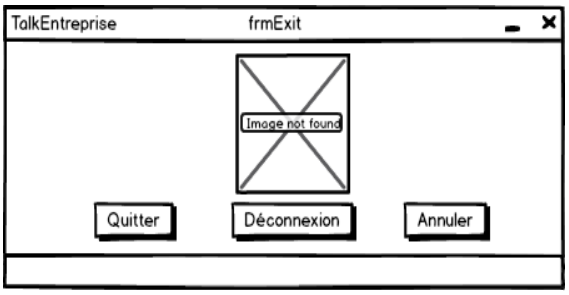
Cette interface permet à l'"Utilisateur" :

- 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.



9 Interface de modification du mot de passe

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

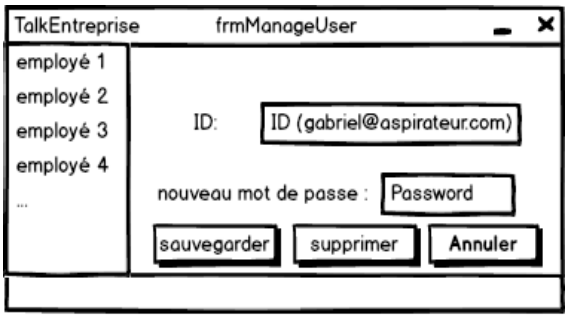


10 Interface de déconnexion

Cette interface permet à l'"Utilisateur" :

- de quitter le programme
- de se déconnecter
- d'annuler la fermeture du programme.

4.3 FONCTIONNALITÉS SPÉCIFIQUES AUX "ADMINISTRATEURS"



11 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 6).

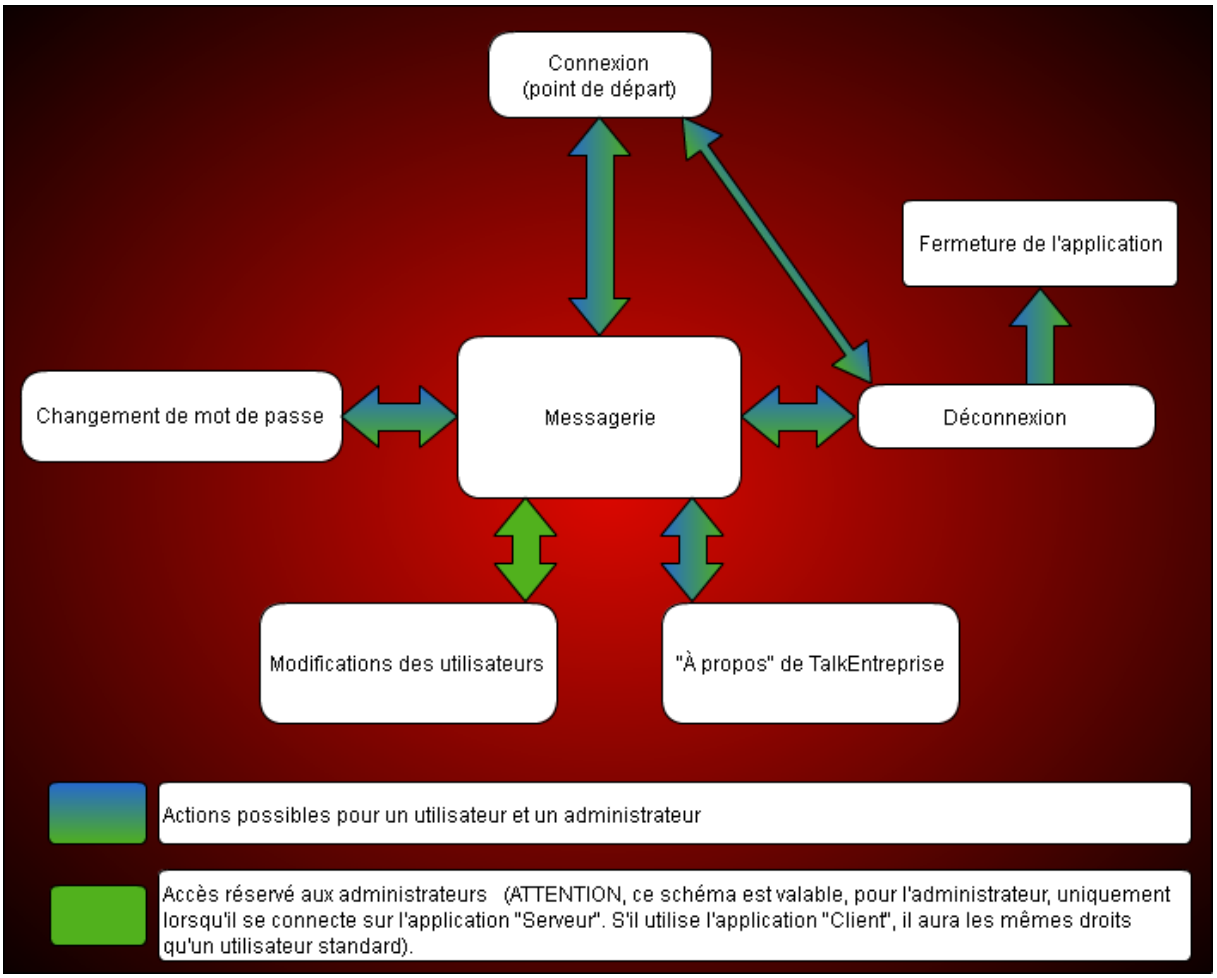
4.4 CHOIX DES COMPOSANTS

Mon programme est principalement composé de "textbox".
J'ai choisi ce composant, car grâce à lui, il est possible de récupérer les informations tapées par l'Utilisateur".
Il est également possible 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 le programme, elle est utilisée pour afficher certaines informations concernant les "Utilisateurs" :

- Leur état de connexion
- Leur identifiant
- Leur statut (s'il est également "Administrateur")
- Leur nombre de messages non lus

4.5 SCHÉMA DES APPLICATIONS



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

5 ANALYSE ORGANIQUE

5.1 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.

Après discussion avec mon référent TPI, Monsieur GARCIA, il m'a conseillé d'utiliser la méthode d'envoi des données via le réseau.

J'ai pris un temps de réflexion pour savoir comment j'allais m'y prendre, et j'ai finalement opté pour cette méthode.

5.2 DIAGRAMME DE CLASSES DE L'APPLICATION "CLIENT"

Se reporter à l'annexe 1

5.3 DIAGRAMME DE CLASSES DE L'APPLICATION "SERVEUR"

Se reporter à l'annexe 2

5.4 FICHIERS UTILES POUR L'APPLICATION

5.4.1 Du côté "Client"

5.4.1.1 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

5.4.1.2 Les classes :

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

5.4.2 Du côté "Serveur"

5.4.2.1 Les vues :

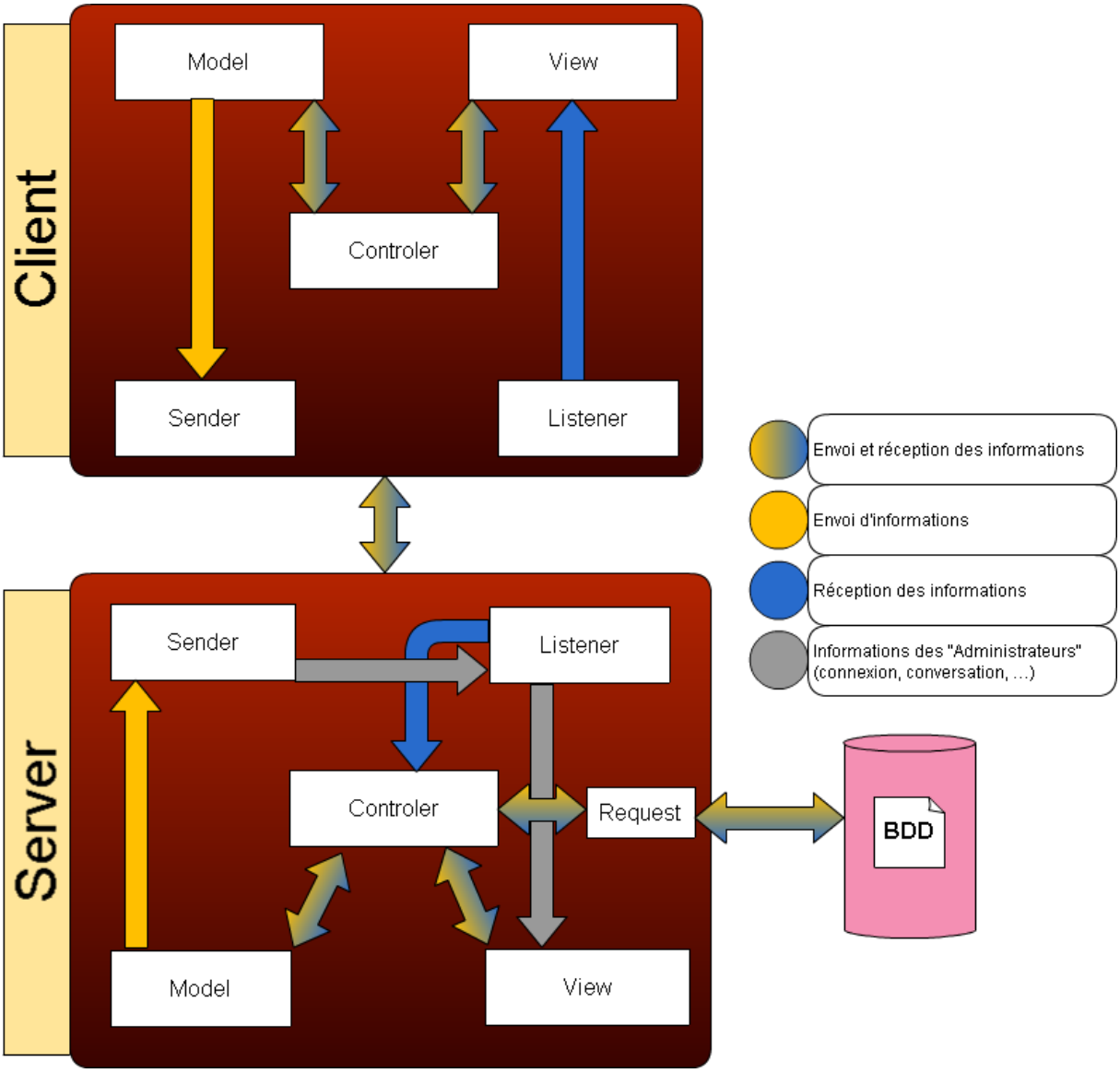
- FrmConnection.cs
- FrmProgram.cs

5.4.2.2 Les classes :

- Controler.cs
- ClientConnectToServ.cs
- UpdateUser.cs
- Message.cs
- RequestSQL.cs
- Server.cs
- User.cs
- Converter.cs

5.5 MÉTHODE DE COMMUNICATION DES APPLICATIONS

Les informations sont communiquées selon le schéma suivant :



13 Schéma de communication des applications

Pour pouvoir échanger des messages entre les différentes applications ("Client" et "Serveur"), j'ai placé un code de début et un code de fin, sur conseil de mon référent.
En effet, mon idée première était d'insérer dans la base de données une chaîne de caractères contenant toute les informations requises pour chaque cas (connexion, déconnexion, envoi de message…).
Après discussion avec Monsieur Garcia, il m'a recommandé, pour la création de la table "log", de créer des "Champs" qui reproduisent une "trame" internet (code de début, longueur totale des informations, informations, code de fin).

En gardant ce principe, j'ai décidé que chaque message allait commencer par le caractère "#" suivi de 4 chiffres et finit par "####". Le reste du message est généralement séparé par des points-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"

5.6 BASE DE DONNÉES

5.6.1 Modèle Conceptuel de Données

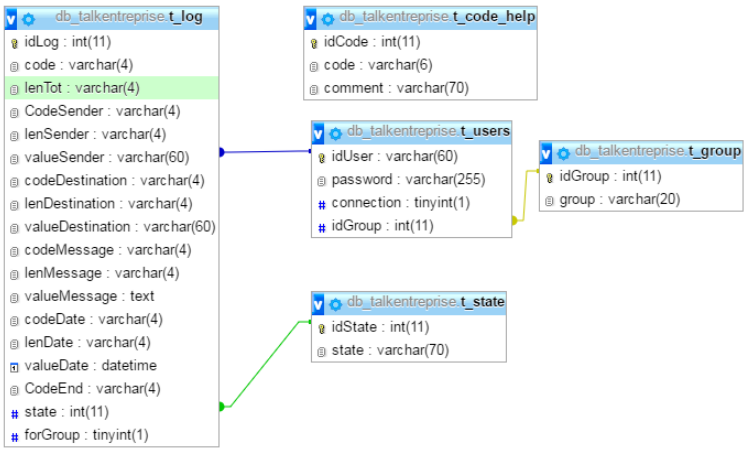


Figure 14 MCD de la base de données

Lors de la création de la base de données, la modification de certaines "Tables" a été faite, ce qui a permis d'éviter le traitement de données inutiles dans l'application.

5.6.1.1 T_code_help

Colonne	Type	Null	Défaut	Commentaires
idCode	int(11)	Non		Identifiant d'un code
code	varchar(6)	Non		Code
comment	varchar(70)	Non		Explication des différents codes

5.6.1.2 T_group

Colonne	Type	Null	Défaut	Commentaires
idGroup	int(11)	Non		Identifiant d'un groupe
group	varchar(20)	Non		Noms des différents secteurs présents dans l'entreprise

5.6.1.3 T_log

Colonne	Type	Null	Défaut	Relié à	Commentaires
idLog	int(11)	Non			Id unique d'une action faite par un "Utilisateur"
code	varchar(4)	Non			Code permettant d'identifier les différentes actions
lenTot	varchar(4)	Non			Longueur totale de la trame en hexadécimal
CodeSender	varchar(4)	Non			Code permettant d'identifier qui est l'envoyeur
lenSender	varchar(4)	Non			Longueur totale de l'identifiant de l'envoyeur en hexadécimal
valueSender	varchar(60)	Non		t_users -> idUser	Identifiant de l'envoyeur
codeDestination	varchar(4)	Non			Code permettant d'identifier qui est le receveur
lenDestination	varchar(4)	Non			Longueur totale de l'identifiant du destinataire en hexadécimal
valueDestination	varchar(60)	Non			Identifiant du receveur
codeMessage	varchar(4)	Non			Code permettant d'identifier le message
lenMessage	varchar(4)	Non			Longueur totale de l'identifiant du message en hexadécimal
valueMessage	text	Non			Contenu du message (crypté)
codeDate	varchar(4)	Non			Code permettant d'identifier la date
lenDate	varchar(4)	Non			Longueur totale de la date en hexadécimal
valueDate	datetime	Non			Valeur de la date et de l'heure

CodeEnd	varchar(4)	Non			Fin de la trame
state	int(11)	Non	2	t_state -> idState	Etat des messages
forGroup	tinyint(1)	Non	0		Si le message est destiné à un groupe

5.6.1.4 T_state

Colonne	Type	Null	Défaut	Commentaires
idState	int(11)	Non		
state	varchar(70)	Non		Différents états d'un message

5.6.1.5 T_users

Colonne	Type	Null	Défaut	Relié à	Commentaires
idUser	varchar(60)	Non			Identifiant d'un "Utilisateur"
password	varchar(255)	Non			Mot de passe de l'"Utilisateur"
connection	tinyint(1)	Non	0		Etat de connexion de l'"Utilisateur"
idGroup	int(11)	Non		t_group -> idGroup	Identifiant du groupe auquel l'"Utilisateur" appartient

5.7 DESCRITPION DES MÉTHODES PRINCIPALES DU "CLIENT"

5.7.1 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 n'en citerai que deux.

5.7.1.1 ResetConnection()

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

```
public bool ResetConnection()
        {
            try
            {
                //Instanciation de l'objet TcpClient
                this.ClientSocket = new TcpClient();
                //Instanciation de l'objet NetworkStream
                this.ServerStream = default(NetworkStream);
                //demande de connexion avec une application qui se trouve à l'adresse
                IP "12.0.0.1" et qui se trouve sur le port "8888"
                this.ClientSocket.Connect("127.0.0.1", 8888);
                //récupération du canal de communication entre le serveur et le client
                this.ServerStream = this.ClientSocket.GetStream();
                return true;
            }
                //si lors de l'exécution du code il y a une erreur alors
            catch (Exception)
            {

                return false;
            }


        }
```

5.7.1.2 ConnectionServer(…)

Cette méthode permet de demander au "Serveur" si l'"Utilisateur" se trouve dans la base de données.

```
private bool ConnectionServer(string id, string password)
        {

            Thread.Sleep(10);
            //Instanciation d'un tableau contenant le maximum de caractère que peut
            contenir un message envoyé par le serveur
            byte[] inStream = new byte[10025];
            string toSend = "#0001;" + id + ";" + password + ";####";
            //Encode le texte en tableau de byte pour l'envoi
            byte[] outStream = Encoding.ASCII.GetBytes(toSend);
            //Envoi au serveur les données préalablement encodées
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique d'envoi des messages
            this.ServerStream.Flush();
            //Récupération des informations envoyer par le serveur
            this.ServerStream.Read(inStream, 0, inStream.Length);
            //Transformation de l'information envoyé par le serveur
            bool result = Convert.ToBoolean(Encoding.ASCII.GetString(inStream));
```

```
            if (result)
            {
                //Envoi la connexion avec le serveur et le canal de communication au
                controler
                 this.Ctrl.SetTcpClientAndNetworkStream(this.ClientSocket,
                this.ServerStream);
            }
            return result;
        }
```

5.7.2 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.

5.7.2.1 Init()

```
public void Init()
        {
            //Initialisation de différentes variabls
            this.LstOldMessages = new List<Message>();
            bool stop = false;
            byte[] inStream = new byte[10025];
            string result = string.Empty;
            List<User> lstUsers = new List<User>();
            List<Message> lstMessages = new List<Message>();
            string[] userInfos;
            bool first = true;
            //cette classe est utiliser dans un autre processus. C'est pour cela qu'il
            est possible de faire une boucle ou la condition est toujours bonne
            while (true)
            {
                //Réinitialisation de la liste des Utilisateur
                lstUsers.Clear();
                first = true;
                //Récupération et traitement des informations envoyés par le serveur
                try
                {
                    this.Stream.Read(inStream, 0, inStream.Length);
                    result = Encoding.ASCII.GetString(inStream);
                    result = result.Substring(0, result.IndexOf("####"));
                }
                catch (Exception)
                {
                    stop = true;

                }

                //S'il n'y a pas de problème
                if (stop)
                {
                    break;
                }
```

```
            //Recherche si une condition est égale au code extrait de la variable
              résult
             switch (result.Split(';')[0])
              {

      {code}

                  case "#0008"
                      //permet de savoir si le changement de mot de passe a bien été
                        fait
                     this.Prog.PasswordIsChanged(Convert.ToBoolean(
                     result.Split(';')[1]), result.Split(';')[2]);
                      break;
                  //si aucune condition est rempli
                  default:
                      stop = true;
                      break;
              }
```

5.7.3 Classe FrmProgram

Cette classe permet de gérer la fenêtre de la messagerie.

5.7.3.1 ShowMessages(…)

Cette méthode permet d'afficher les messages à l'"Utilisateur".

```
public void ShowMessages(List<Message> lstNewMessages, string destination, string
iduser, bool isforGroup)
        {
            //sécursation pour pouvoir exécuter les prochaines instructions.
            Invoke(new MethodInvoker(delegate
            {
                //récupération de l'utilisateur sélectionné
                User user = this.lsbEmployees.SelectedItem as User;

                if (user != null)
                {

                    // si les messages reçus sont pour la conversation actuellement
                    visible par l'utilisateur
                    if ((user.GetIdUser() == destination || user.GetIdUser() == iduser
&& user.GetIdUser().Contains("@")) || (isforGroup && lsbEmployees.SelectedIndex == 0))
                    {
                        //variable permettant de stocker les nouveau messages à
                        afficher
                        string messages = string.Empty;
                        //pour chaque nouveau message
                        for (int i = this.NbMessages; i < lstNewMessages.Count; i++)
                        {
                            //récupération des informations relatif au message
                            Message msg = lstNewMessages[i] as Message;
                            //Si le message n'est pas envoyé par le même utilisateur
                            alors afficher l'autheur du messae
                            if (this.LastAuthor != msg.Author)
                            {
                                messages += Environment.NewLine +
String.Format("{0,30}----------------------------------", "") +
msg.GetAuthor().Split('@')[0] + "----------------------------------";
                            }
```

```
                            messages += Environment.NewLine + msg.GetContent() +
Environment.NewLine + String.Format(" {0,130 }Date: ", string.Empty) + msg.GetDate();
                            this.LastAuthor = msg.GetAuthor();
                        }
                        //permet de sauvegarder le nombre de message afficher sur
                        l'écran
                        this.NbMessages = lstNewMessages.Count;
                        // Ajout des nouveaux messages visuellement (l'instruction
                       "Invoke" permet de modifier les composants du programme)
                        tbxMessages.AppendText(messages);
                    }

                }
                else
                {
                    //lancer la méthode qui permet
                    this.ServerClosed();
                }

            }));
        }
```

5.8 DESCRITPION DES MÉTHODES PRINCIPALES DU "SERVEUR"

5.8.1 Classe ClientConnectToServ

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

5.8.1.1 Init()

Cette méthode permet d'enregistrer les différents "Clients" connectés.

```
public void Init()
        {
            //Initialisation du client et initialisation de l'objet permettant de
            recevoir les connexions
            TcpListener serverSocket = new TcpListener(8888);
            TcpClient clientSocket = default(TcpClient);


            serverSocket.Start();


            while ((true))
            {

                //si un client veut se connecter on accepte
                clientSocket = serverSocket.AcceptTcpClient();
                byte[] sendBytedMessage = null;
                //Initialisation d'un tableau qui contient le maximum de byte que le
                serveur peux envoyer
                byte[] bytesFrom = new byte[10025];
                string dataFromClient = null;
                string user = string.Empty;
                string password = string.Empty;
                bool sendToClient = false;
                string sendInfo = string.Empty;
                List<string> userInfo = new List<string>();
```

```
                ///initialisation du flux et récupération des informations envoyé par
                le client
                NetworkStream networkStream = clientSocket.GetStream();
                networkStream.Read(bytesFrom, 0, bytesFrom.Length);
                //encode le tableau de bytes
                dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
                //récupère la valeure envoyée
                if (dataFromClient.Contains("#0001"))
                {
                    dataFromClient = dataFromClient.Substring(0,
                    dataFromClient.IndexOf("####"));
                    user = dataFromClient.Split(';')[1];
                    password = dataFromClient.Split(';')[2];
                }
                //si la base de données connait l'utilisateur
                if (this.validateConnection(user, password))
                {
                    this.Serv.SucessConnectionToServer(user);
                    sendToClient = true;

                    this.Serv.addClientList(user, clientSocket);
                }
                //envoi true ou false au client
                Thread.Sleep(10);
                sendBytedMessage = Encoding.ASCII.GetBytes(sendToClient.ToString());
                networkStream.Write(sendBytedMessage, 0, sendBytedMessage.Length);
                //si la connexion a abouti
                if (sendToClient)
                {
                    //récupération des informations de l'utilisateur dans la bas de
                    données
                    Thread.Sleep(10);
                    userInfo = this.GetInformation(user);
                    sendInfo = "#0004;" + user + ",";
                    foreach (var info in userInfo)
                    {
                        sendInfo += info + ",";
                    }
                    sendInfo += "####";
                    sendBytedMessage = Encoding.ASCII.GetBytes(sendInfo);
                    networkStream.Write(sendBytedMessage, 0, sendBytedMessage.Length);
                    //permet de mettre à jour la liste des threads
                    this.UserUpdate = new Thread(new UpdateUser(clientSocket,
networkStream, sendToClient, this, user).Update);
                    this.Serv.AddThreadList(user, UserUpdate);
                    this.UserUpdate.IsBackground = true;
                    this.UserUpdate.Start();
                }

            }
            //stop toutes les communications
            clientSocket.Close();
            serverSocket.Stop();

        }
```

5.8.2 Classe RequestSQL

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

5.8.2.1 ConnectionDB()

```
public bool ConnectionDB()
        {
            try
            {
                //information de connexion pour accéder à la table spécifier
                string connectionString =
@"server=127.0.0.1;userid=IT;password=Super;database=db_talkEntreprise";
                //Initialisation de la connection avec la base de données
                this.ConnectionUser = new MySqlConnection(connectionString);
                this.ConnectionUser.Open();
                return true;
            }
            catch (MySqlException ex)
            {
                return false;
            }
        }
```

5.8.2.2 ValidateConnectionUser(…)

```
public Boolean ValidateConnectionUser(string user, string password)
        {
            bool result = false;
            //si la connexion a réussi
            if (this.ConnectionDB())
            {
                //Requête permettant de vérifier si l'utilisateur se trouve bien dans
la base de données
                string sql = String.Format("SELECT Count(*) as total FROM t_users
where  idUser  = '{0}' AND Password = '{1}'", user, password);
                //permet de préparer l'exécution de la requête
                MySqlCommand cmd = new MySqlCommand(sql, ConnectionUser);
                //exécute la requête
                cmd.ExecuteNonQuery();
                //Instantiation de l'objet qui permet de lire les informations
retournées pare la base de données
                MySqlDataReader reader = cmd.ExecuteReader();
                //Lecture des informations retournées
                reader.Read();
                if (Convert.ToInt32(reader.GetString("total")) == 1)
                {

                    result = true;

                }

                reader.Close();

                this.ShutdownConnectionDB();

            }

            return result;
        }
```

6 PROTOCOLE DE TESTS

Vous trouverez les tests dans l'annexe 3.

7 CONCLUSION

7.1 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.

7.2 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".
N'ayant jamais utilisé "l'envoi de données via le réseau", bien qu'avec un peu de crainte, j'ai décidé de relever le 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] 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], 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 "Utilisateurs" 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'était 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.

[1] Nom du site dans la bibliographie
[2] Librairie de fichiers ".dll" qui permet l'utilisation de différents types d'objets

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, lié au premier, est que 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 se 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 alors 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.

7.3 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 une autre personne (pour la première fois), il faut cliquer deux fois sur la personne en question pour la sélectionner.

Lors de la connexion, il arrive parfois que l'affichage de la conversation n'apparaisse pas ou qu'une seule partie des messages soit affiché.

Lorsqu'un "Utilisateur" consulte ses messages, il arrive qu'un message consulté apparaisse en tant que "non-lus".

7.4 AMÉLIORATIONS ENVISAGEABLES

- ☞ Optimisation de l'envoi de données entre le "Serveur" et le "Client".
- ☞ Correction des problèmes non résolus.

8 ANNEXES

- Annexe 1 : DiagrammeClient.pdf
- Annexe 2 : DiagrammeServeur.pdf
- Annexe 3 :ProtocoleTests.pdf

9 BIBLIOGRAPHIE

9.1 IMAGSE UTILISÉES

9.1.1 Immeuble

https://pixabay.com/static/uploads/photo/2014/12/14/15/45/building-567929_960_720.png

9.1.2 Bulles

https://pixabay.com/static/uploads/photo/2013/07/12/12/54/speech-bubble-146508_960_720.png

9.1.3 Whatsapp

“Image Gratuite Sur Pixabay - Whatsapp, Smartphone, Cellulaire.” Accessed June 13, 2016. https://pixabay.com/fr/whatsapp-smartphone-cellulaire-1317538/.

9.2 SITES UTILISÉS POUR LA CRÉATION DU PROJET

9.2.1 Utilisation de l'objet "Invoke"

“Changer Texte Label Avec Conflit de Thread [C#] - C#/.NET Managed - Programmation - FORUM HardWare.fr. http://forum.hardware.fr/hfr/Programmation/CNET-managed/changer-conflit-thread-sujet_123171_1.htm.

9.2.2 Convertisseur de décimal en hexadécimal

“Hex - How to Convert Numbers between Hexadecimal and Decimal in C#? - Stack Overflow.” http://stackoverflow.com/questions/74148/how-to-convert-numbers-between-hexadecimal-and-decimal-in-c.

9.2.3 Explication d'envoi de données via le réseau

“How to C# Chat Server Programming.” http://csharp.net-informations.com/communications/csharp-chat-server-programming.htm.

9.2.4 Utilisation de l'objet "Dispatcher"

“Launching a WPF Window in a Separate Thread, Part 1 : Reed Copsey, Jr.” http://reedcopsey.com/2011/11/28/launching-a-wpf-window-in-a-separate-thread-part-1/.

“Multithreading - How Do I Create and Show WPF Windows on Separate Threads? - Stack Overflow http://stackoverflow.com/questions/1111369/how-do-i-create-and-show-wpf-windows-on-separate-threads.

10 BIBLIOGRAPHIE DES FIGURES