```
/*****************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace talkEntreprise_client
{
    public class Message
    {
        //////Champs/////////////
        private string _author;
        private string _content;
        private string _date;
        //////Propriétées/////////////
        public string Author
        {
            get { return _author; }
            set { _author = value; }
        }
        public string Content
        {
            get { return _content; }
            set { _content = value; }
        }
        public string Date
        {
            get { return _date; }
            set { _date = value; }
        }
        //////Constructeur/////////////
        public Message(string user, string valueMessage, string valueDate)
        {
            this.Author = user;
            this.Content = valueMessage;
            this.Date = valueDate;
        }
        public string GetDate()
        {
            return this.Date;
        }
        public string GetAuthor()
        {
            return this.Author;
        }
        public string GetContent()
        {
            return this.Content;
        }
    }
}
```

```
/*******************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*******************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace talkEntreprise_client
{
    public class User
    {
        ///////Champs//////////
        private string _idUser;
        private string _password;
        private int _idGroup;
        private bool _connection;
        private string _groupeName;
        private bool _forGroup;
        private string _admin;

        ///////propriétées///////

        public bool Connection
        {
            get { return _connection; }
            set { _connection = value; }
        }

        public string IdUser
        {
            get { return _idUser; }
            set { _idUser = value; }
        }
        public string Password
        {
            get { return _password; }
            set { _password = value; }
        }
        public int IdGroup
        {
            get { return _idGroup; }
            set { _idGroup = value; }
        }
        private int _messageNotRead;

        public int MessageNotRead
        {
            get { return _messageNotRead; }
            set { _messageNotRead = value; }
        }
        public string GroupeName
        {
            get { return _groupeName; }
            set { _groupeName = value; }
        }
        public string Admin
        {
            get { return _admin; }
            set { _admin = value; }
        }
        public bool ForGroup
        {
            get { return _forGroup; }
            set { _forGroup = value; }
        }
        ////////Constructeur/////////

        public User(string id, string pwd, int group, bool connect, int
        nbMessagesNotRead, string nameGroup)
        {
            SetUser(id, pwd, group, connect, nbMessagesNotRead, nameGroup);
        }

        /////////méthodes/////////
        /// <summary>
        /// permet d'initialiser les informations de l'utilisateur
        /// </summary>
        /// <param name="id">identifiant de connexion</param>
        /// <param name="pwd">mot de passe de l'utilisateur</param>
        /// <param name="group">numéro du groupe de l'utilisateur</param>
        /// <param name="connect">état de connection de l'utilisateur</param>
        /// <param name="nbMessagesNotRead">nombre de message en attente</param>
        /// <param name="nameGroup">nom du groupe de l'utilisateur</param>
        public void SetUser(string id, string pwd, int group, bool connect, int
        nbMessagesNotRead, string nameGroup)
        {
            this.IdUser = id;
            this.Password = pwd;
            this.IdGroup = group;
            this.Connection = connect;
            this.MessageNotRead = nbMessagesNotRead;
            this.GroupeName = nameGroup;

            if (this.IdGroup == 3)
            {
                this.Admin = "Administrateur";
            }
            else
            {
                this.Admin = "";
            }
        }
        //////////méthodes/////////
        /// <summary>
        /// donne le nom du groupe de l'utiliateur
        /// </summary>
        /// <returns>nom du groupe de l'utilisateur</returns>
        public string GetNameGroup()
        {
            return this.GroupeName;
        }
        /// <summary>
        /// donne l'identifiant de connexion de l'utilisateur
        /// </summary>
        /// <returns></returns>
        public string GetIdUser()
        {
            return this.IdUser;
        }
        /// <summary>
        /// donne l'information si l'utilisateur est connecté ou pas
        /// </summary>
        /// <returns>true ou false</returns>
        public bool GetInformationConnection()
        {
            return this.Connection;
        }
        /// <summary>
        /// donne l'identifiant du groupe de l'utilisateur
        /// </summary>
        /// <returns></returns>
        public int GetIdGroup()
        {
            return this.IdGroup;
        }
        /// <summary>
        /// retourne le nombre de message non lu de l'utilisateur
        /// </summary>
        /// <returns>nombre de message non lu</returns>
        public int GetMessagesNotRead()
        {
            return this.MessageNotRead;
        }
        /// <summary>
        /// permet de mettre à jour le nombre de message non lu de l'utilisateur
        /// </summary>
        /// <param name="nbmessagesNotRead">nombre de messages non lu</param>
        public void SetMessagesNotRead(int nbmessagesNotRead)
        {
            this.MessageNotRead = nbmessagesNotRead;
        }
        /// <summary>
        /// met à jour la connexion de l'utilisateur
        /// </summary>
        /// <param name="b"></param>
        public void SetConnection(bool b)
        {
            this.Connection = b;
        }
        /// <summary>
        /// retourne le mot de passe de l'utilisateur
        /// </summary>
        /// <returns>mot de passe de l'utilisateur</returns>
        public string GetPassword()
        {
            return this.Password;
        }
        /// <summary>
        /// si la personne est admin
        /// </summary>
        /// <returns>si admin</returns>
        public string GetAdmin()
        {
            return this.Admin;
        }
        public void SetPassword(string password)
        {
            this.Password = password;
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace talkEntreprise_client
{
    public class Client
    {
        //////////Champs//////////
        private Controler _ctrl;
        private TcpClient _clientSocket;
        private NetworkStream _serverStream;
        ////////////propriétées//////////
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        public TcpClient ClientSocket
        {
            get { return _clientSocket; }
            set { _clientSocket = value; }
        }
        public NetworkStream ServerStream
        {
            get { return _serverStream; }
            set { _serverStream = value; }
        }
        //////////////Constructeur////////////
        public Client(Controler c)
        {
            this.Ctrl = c;
        }
        //////////////méthodes////////////
        /// <summary>
        /// elle permet de savoir si l'utilisateur se trouve dans la base de données
        /// </summary>
        /// <param name="id">identifiant de l'utilisateur</param>
        /// <param name="password">mot de passe de l'utilisateur</param>
        /// <returns>vrais ou faux</returns>
        public bool Connection(string id, string password)
        {
            string cryptedPassword = this.Ctrl.Sha1(password);
            return ConnectionServer(id, cryptedPassword);
        }
        /// <summary>
        /// elle permet d'envoyer l'identifiant et le mot de passe au serveur et
        récupérer la réponse.
        /// </summary>
        /// <param name="id">identifiant de l'utilisateur</param>
        /// <param name="password">mot de passe de l'utilisateur</param>
        /// <returns>vrais ou faux</returns>
        private bool ConnectionServer(string id, string password)
        {
            Thread.Sleep(10);
            byte[] inStream = new byte[10025];
            string toSend = "#0001;" + id + ";" + password + ";####";
            //Encode le texte en tableau de byte
            byte[] outStream = Encoding.ASCII.GetBytes(toSend);
            //Envoie au serveur les données
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique
            this.ServerStream.Flush();
            //Assignation de la valeur envoyée par le serveur(sous forme de tableau de
            bytes) this.ServerStream.Read(inStream, 0, inStream.Length);
            bool result =
            Convert.ToBoolean(Encoding.ASCII.GetString(inStream)); if (result)
            {
                this.Ctrl.SetTcpClientAndNetworkStream(this.ClientSocket, this.ServerStream);
            }
            return result;
        }
        /// <summary>
        /// permet de recréer une connection avec le serveur
        /// </summary>
        public bool ResetConnection()
        {
            try
            {
                this.ClientSocket = new TcpClient();
                this.ServerStream = default(NetworkStream);
                this.ClientSocket.Connect("127.0.0.1", 8888);
                this.ServerStream =
                this.ClientSocket.GetStream(); return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
        /// <summary>
        /// permet d'envoyer un message au serveur pour lui dire de se déconnecter
        /// </summary>
        public void CloseConnection()
        {
            byte[] inStream = new byte[10025];
            ;
            string toSend = "#0002####";
            //Encode le texte en tableau de byte
            byte[] outStream = Encoding.ASCII.GetBytes(toSend + "####");
            //Envoie au serveur les données
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique
            this.ServerStream.Flush();
        }
        /// <summary>
        /// permet de récupérer les informations de l'utilisateur
        /// </summary>
        /// <param name="user"> identifiant user</param>
        /// <returns>utilisateur</returns>
        public User GetInformationUserConnected()
        {
            byte[] inStream = new byte[10025];
            List<string> lstInfo = new List<string>();
            this.ServerStream.Read(inStream, 0, inStream.Length);
            string result = Encoding.ASCII.GetString(inStream);
            result = result.Substring(0, result.IndexOf("####"));
            result = result.Split(';')[1];
            foreach (string info in result.Split(','))
            {
                lstInfo.Add(info);
            }
            return new User(lstInfo[0], lstInfo[3], Convert.ToInt32(lstInfo[1]), true,
            0, lstInfo[2]);
        }
        /// <summary>
        /// permet d'envoyer le message ua serveur
        /// </summary>
        /// <param name="message">message</param>
        public bool SendMessage(string user, string destination, string message, bool
        forGroup)
        {
            string sendMessage = "#0003;" + user + "-" + destination + "-"
            + this.Ctrl.EncryptMessage(message) + "-" + forGroup + "#####";
            byte[] inStream = new byte[10025];
            try
            {
                //Encode le texte en tableau de byte
                byte[] outStream = Encoding.ASCII.GetBytes(sendMessage);
                //Envoie au serveur les données
                this.ServerStream.Write(outStream, 0, outStream.Length);
                //Efface l'historique
                this.ServerStream.Flush();
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
        /// <summary>
        /// permet d'envoyer le message au serveur
        /// </summary>
        /// <param name="message">message</param>
        public void SendMessageGroup(string user, string Alldestination, string
        message, bool forGroup)
        {
            string sendMessage = "#0003;" + user + "-" + Alldestination + "-"
            + this.Ctrl.EncryptMessage(message) + "-" + forGroup + "#####";
            byte[] inStream = new byte[10025];
            //Encode le texte en tableau de byte
            byte[] outStream = Encoding.ASCII.GetBytes(sendMessage);
            //Envoie au serveur les données
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique
            this.ServerStream.Flush();
        }
        /// <summary>
        /// permet d'afficher la conversation de l'utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire du message</param>
        /// <param name="forGroup">si c'est pour le groupe</param>
        public void GetConversation(string user, string destination, bool forGroup)
        {
            string sendMessage = "#0004;" + user + "-" + destination + "-" + forGroup +
            "#####";
            byte[] inStream = new byte[10025];

            try
            {
                //Encode le texte en tableau de byte
                byte[] outStream = Encoding.ASCII.GetBytes(sendMessage);
                //Envoie au serveur les données
                this.ServerStream.Write(outStream, 0, outStream.Length);
                //Efface l'historique
                this.ServerStream.Flush();

            }
            catch (Exception)
            {

            }
        }
        /// <summary>
        /// met à jour la liste des employés
        /// </summary>
        /// <param name="nameGroupe">nom du groupe de l'utilisateur</param>
        /// <param name="user"> identifiant de l'utilisateur</param>
        /// <param name="idGroup">id du groupe de l'utilisateur</param>
        public void UpdateUsers(string nameGroupe, string user, int idGroup)
        {
            string sendMessage = "#0005;" + nameGroupe + ";" + user + ";" + idGroup +
            "#####"; byte[] inStream = new byte[10025];


            //Encode le texte en tableau de byte
            byte[] outStream = Encoding.ASCII.GetBytes(sendMessage);
            //Envoie au serveur les données
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique
            this.ServerStream.Flush();
        }
        /// <summary>
        /// permet de mettre à jour l'état des messages
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire</param>
        /// <param name="isForGroup">pour un groupe</param>
        public void UpdateStateMessages(string user, string destination, bool
        isForGroup, string nameGroup, int idGroup, string userSecure)
        {
            string sendMessage = "#0006;" + user + ";" + destination + ";" + isForGroup +
            ";" + nameGroup + ";" + idGroup + ";" + userSecure +
            "#####"; byte[] inStream = new byte[10025];
            //Encode le texte en tableau de byte
            byte[] outStream = Encoding.ASCII.GetBytes(sendMessage);
            //Envoie au serveur les données
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique
            this.ServerStream.Flush();
        }
        /// <summary>
        /// permet de récupérer les anciens messages
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire</param>
        /// <param name="forGroup">pour le groupe</param>
        /// <param name="nbDays">jour avant aujourd'huit</param>
        public void GetOldMessages(string user, string destination, bool forGroup, int nbDays)
        {
            string sendMessage = "#0007;" + user + ";" + destination + ";" + forGroup + ";"
            + nbDays + "#####";
            byte[] inStream = new byte[10025];
            //Encode le texte en tableau de byte
            byte[] outStream = Encoding.ASCII.GetBytes(sendMessage);
            //Envoie au serveur les données
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique
            this.ServerStream.Flush();
        }
        /// <summary>
        /// permet de modifier le mot de passe actuelle
        /// </summary>
        /// <param name="password"></param>
        public void ChangePassword(string user, string password)
        {
            string sendMessage = "#0008;" + user + ";" + password +
            "#####"; byte[] inStream = new byte[10025];
            //Encode le texte en tableau de byte
            byte[] outStream = Encoding.ASCII.GetBytes(sendMessage);
            //Envoie au serveur les données
            this.ServerStream.Write(outStream, 0, outStream.Length);
            //Efface l'historique
            this.ServerStream.Flush();
        }
    }
}
```

```
/******************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*******************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Security.Cryptography;
using System.Windows.Threading;
using System.Threading;
using System.Net.Sockets;
namespace talkEntreprise_client
{
    public class Controler
    {
        ////////////////Champs/////////////
        private Client _client;
        private FrmConnection _connect;
        private TcpClient _tClient;
        private NetworkStream _stream;
        private Thread _frmProg;
        private User _userInformation;
        private ManageMessages _manMessage;

        public ManageMessages ManMessage
        {
            get { return _manMessage; }
            set { _manMessage = value; }
        }
        /////////////propriétées//////////
        public Client Client
        {
            get { return _client; }
            set { _client = value; }
        }
        public FrmConnection Connect
        {
            get { return _connect; }
            set { _connect = value; }
        }
        public TcpClient TClient
        {
            get { return _tClient; }
            set { _tClient = value; }
        }
        public NetworkStream Stream
        {
            get { return _stream; }
            set { _stream = value; }
        }
        public Thread FrmProg
        {
            get { return _frmProg; }
            set { _frmProg = value; }
        }
        public User UserInformation
        {
            get { return _userInformation; }
            set {  userInformation = value; }
        }
        //////////////Constructeur//////////

        public Controler(FrmConnection c)
        {
            this.Connect = c;
            this.Client = new Client(this);
            this.ManMessage = new ManageMessages(this);
        }
        //////méthodes Générales///////
        /// <summary>
        /// permet de coder le mot de passe de l'utilisateur
        /// </summary>
        /// <param name="password">mot de passe de l'utilisateur</param>
        /// <returns></returns>
        public string Sha1(string password)
        {
            //créer une instance sha1
            SHA1 sha1 = SHA1.Create();
            //convertit le texte en byte
            byte[] hashData = sha1.ComputeHash(Encoding.Default.GetBytes(password));
            //créer une instance StringBuilder pour sauver les
            hashData StringBuilder returnValue = new StringBuilder();
            //transform un tableau en string
            for (int i = 0; i < hashData.Length; i++)
            {
                returnValue.Append(hashData[i].ToString());
            }

            // return hexadecimal string
            return returnValue.ToString();
        }
        /// <summary>
        /// elle permet de lancer le programme principal
        /// </summary>
        public void CreateProgram(string user)
        {
            //création d'un nouveau processus
            this.FrmProg = new Thread(new ThreadStart(ThreadProgram));
            this.FrmProg.SetApartmentState(ApartmentState.STA);

            //lancer le processus
            this.FrmProg.Start();
        }
        /// <summary>
        /// permet de créer la fenêre FrmProgram dans un aute processus
        /// </summary>
        public void ThreadProgram()
        {
            FrmProgram prog = new FrmProgram(this);
            prog.FormClosed += (s, e) =>
            Dispatcher.CurrentDispatcher.BeginInvokeShutdown(DispatcherPriority.Background);
            prog.Show();

            //permet de garder la fenêtre
            ouverte Dispatcher.Run();
        }
        /// <summary>
        /// permet de sauvegarder la connexion existente au serveur
        /// </summary>
        /// <param name="t">connexion du client</param>
        /// <param name="s">flux d'information entre le client et le serveur</param>
        public void SetTcpClientAndNetworkStream(TcpClient t, NetworkStream s)
        {
            this.TClient = t;
            this.Stream = s;
        }
        ///////////////méthodes Client /////////////////7
        /// <summary>
        /// elle permet de savoir si l'utilisateur peut se connecter
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="password">mot de passe de l'utilisateur</param>
        /// <returns></returns>
        public bool Connection(string user, string password)
        {
            return this.Client.Connection(user, password);
        }

        /// <summary>
        /// permet d'avertire le serveur que l'utilisateur se déconnecte
        /// </summary>
        public void CloseConnection()
        {
            this.Client.CloseConnection();
        }
        /// <summary>
        /// permet de réinitialiser la connexion avec le server
        /// </summary>
        public bool ResetConnection()
        {
            return this.Client.ResetConnection();
        }
        /// <summary>
        /// permet d'envoyer le message ua serveur
        /// </summary>
        /// <param name="message">message</param>
        public bool SendMessage(string user, string destination, string message, bool
        forGroup)
        {
            return this.Client.SendMessage(user, destination, message, forGroup);
        }
        /// <summary>
        /// met à jour la liste des employés
        /// </summary>
        /// <param name="nameGroupe">nom du groupe de l'utilisateur</param>
        /// <param name="user"> identifiant de l'utilisateur</param>
        /// <param name="idGroup">id du groupe de l'utilisateur</param>
        public void UpdateUsers(string nameGroupe, string user, int idGroup)
        {
            this.Client.UpdateUsers(nameGroupe, user, idGroup);
        }
        /// <summary>
        /// permet d'envoyer le message au serveur
        /// </summary>
        /// <param name="message">message</param>
        public void SendMessageGroup(string user, string Alldestination, string
        message, bool forGroup)
        {
            this.Client.SendMessageGroup(user, Alldestination, message, forGroup);
        }
        /// <summary>
        /// permet d'afficher la conversation de l'utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire du message</param>
        /// <param name="forGroup">si c'est pour le groupe</param>
        public void GetConversation(string user, string destination, bool forGroup)
        {
            this.Client.GetConversation(user, destination, forGroup);
        }
        /// <summary>
        /// permet de mettre à jour l'état des messages
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire</param>
        /// <param name="isForGroup">pour un groupe</param>
        public void UpdateStateMessages(string user, string destination, bool
        isForGroup, string nameGroup, int idGroup, string userSecure)
        {
            this.Client.UpdateStateMessages(user, destination, isForGroup,
            nameGroup, idGroup, userSecure);
        }
        /// <summary>
        /// permet de modifier le mot de passe actuelle
        /// </summary>
        /// <param name="password"></param>
        public void ChangePassword(string user, string password)
        {
            this.Client.ChangePassword(user, password);
        }
        /// <summary>
        /// permet de récupérer les anciens messages
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire</param>
        /// <param name="forGroup">pour le groupe</param>
        /// <param name="nbDays">jour avant aujourd'huit</param>
        public void GetOldMessages(string user, string destination, bool forGroup, int nbDays)
        {
            this.Client.GetOldMessages(user, destination, forGroup, nbDays);
        }
        ////////////méthodes ManageMessages////////////
        /// <summary>
        /// permet de décrypter un message
        /// </summary>
        /// <param name="message">message de l'utilisateur</param>
        /// <returns>message codé</returns>
        public string EncryptMessage(string message)
        {
            return this.ManMessage.EncryptMessage(message);
        }

        /// <summary>
        /// permet de décoder le message
        /// </summary>
        /// <param name="message">message codé</param>
        /// <returns>message original</returns>
        public string DecryptMessage(string message)
        {
            return this.ManMessage.DecryptMessage(message);
        }
        ////////////méthodes FrmConnection/////////////
        /// <summary>
        /// permet de modifier la visibilité de la vue
        /// </summary>
        public void VisibleChange(bool isAbort)
        {
            this.Connect.VisibleChange();
            if (isAbort)
            {
                this.TClient.Close();
                this.Stream.Close();
            }
        }
        ////////////méthodes spécifique au controler////
        /// <summary>
        /// permet de donner la connexion du client
        /// </summary>
        /// <returns>connexion du client</returns>
        public TcpClient GetTcpClient()
        {
            return this.TClient;
        }
        /// <summary>
        /// permet de donner le flux d'information du client
        /// </summary>
        /// <returns>flux d'information du client</returns>
        public NetworkStream GetNetStream()
        {
            return this.Stream;
        }

        public void SetUserConnected()
        {
            this.UserInformation = this.Client.GetInformationUserConnected();
        }
        public User GetUserConnected()
        {
            return this.UserInformation;
        }
    }
}
```

```
/*******************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*******************************************/
using System;
using System.Collections.Generic;
using System.ComponentModel; using
System.Drawing;
using System.Linq;
using System.Reflection; using
System.Threading.Tasks; using
System.Windows.Forms;

namespace talkEntreprise_client
{
    partial class FrmAbout : Form
    {
        public FrmAbout()
        {
            InitializeComponent();
            this.Text = String.Format("À propos de {0}", AssemblyTitle);
            this.labelProductName.Text = AssemblyProduct;
            this.labelVersion.Text = String.Format("Version {0}",
            AssemblyVersion); this.labelCopyright.Text = AssemblyCopyright;
            this.labelCompanyName.Text = AssemblyCompany;
            this.textBoxDescription.Text = AssemblyDescription;
        }

        #region Accesseurs d'attribut de l'assembly

        public string AssemblyTitle
        {
            get
            {
                object[] attributes =
                Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyTitleAttrib
                ute), false);
                if (attributes.Length > 0)
                {
                    AssemblyTitleAttribute titleAttribute =
                    (AssemblyTitleAttribute)attributes[0];
                    if (titleAttribute.Title != "")
                    {
                        return titleAttribute.Title;
                    }
                }
                return
                System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().Cod
                eBase);
            }
        }

        public string AssemblyVersion
        {
            get
            {
                return Assembly.GetExecutingAssembly().GetName().Version.ToString();
            }
        }

        public string AssemblyDescription
        {
            get
            {
                object[] attributes =
                Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescription
                Attribute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyDescriptionAttribute)attributes[0]).Description;
            }
        }

        public string AssemblyProduct
        {
            get
            {
                object[] attributes =
                Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyProductAttr
                ibute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyProductAttribute)attributes[0]).Product;
            }
        }

        public string AssemblyCopyright
        {
            get
            {
                object[] attributes =
                Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCopyrightAt
                tribute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyCopyrightAttribute)attributes[0]).Copyright;
            }
        }

        public string AssemblyCompany
        {
            get
            {
                object[] attributes =
                Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCompanyAttr
                ibute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyCompanyAttribute)attributes[0]).Company;
            }
        }
        #endregion
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.ComponentModel; using
System.Data;
using System.Drawing;
using System.Linq; using
System.Text; using
System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace talkEntreprise_client
{
    public partial class FrmConnection : Form
    {
        ///////Champs///////////
        private Controler _ctrl;
        private bool _reConnection;
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        ///////propriétées//////////
        public bool ReConnection
        {
            get { return _reConnection; }
            set { _reConnection = value; }
        }
        ///////méthodes//////////
        public FrmConnection()
        {
            InitializeComponent();
            this.Ctrl = new Controler(this);
            this.ReConnection = false;
        }
        private void btnConnect_Click(object sender, EventArgs e)
        {
            if (this.Ctrl.ResetConnection())
            {
                if (this.Ctrl.Connection(tbxId.Text, tbxPassword.Text))
                {
                    tbxPassword.Clear();
                    tbxPassword.Focus();
                    this.Visible = !this.Visible;
                    this.Ctrl.SetUserConnected();
                    Thread.Sleep(40);
                    this.Ctrl.CreateProgram(tbxId.Text);
                }
                else
                {
                    MessageBox.Show("Identifiant ou mot de passe incorrecte. ", "Connexion
                    non valide", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }
            }
            else
            {
                MessageBox.Show("Le serveur est inaccessible pour le moment réessayé
                ultérieurement", "Serveur injoignable", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
            }
        }

        private void btnQuit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
        /// <summary>
        /// permet de modifier la visibilité de la vue
        /// </summary>
        public void VisibleChange()
        {
            Invoke(new MethodInvoker(delegate
            {
                 this.Visible = !this.Visible;
            }));
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.ComponentModel; using
System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace talkEntreprise_client
{
    public partial class FrmExit : Form
    {
        public FrmExit()
        {
            InitializeComponent();
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.ComponentModel; using
System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Threading;
using talkEntreprise_client.classThread;

namespace talkEntreprise_client
{
    public partial class FrmProgram : Form
    {
        private Controler _ctrl;
        private Thread _updateLstUser;
        private List<User> _lstUser;
        private User _userConnected;
        private string _lastAuthor;
        private int _nbMessages;
        private User _lastSelectedUser;
        private bool _serverError;
        private const int IDADMINISTRATOR = 3;
        private int _dayOldMessages;
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        public Thread UpdateLstUser
        {
            get { return _updateLstUser; }
            set { _updateLstUser = value; }
        }
        public List<User> LstUser
        {
            get { return _lstUser; }
            set { _lstUser = value; }
        }
        public User UserConnected
        {
            get { return _userConnected; }
            set { _userConnected = value; }
        }
        public int NbMessages
        {
            get { return _nbMessages; }
            set { _nbMessages = value; }
        }
        public string LastAuthor
        {
            get { return _lastAuthor; }
            set { _lastAuthor = value; }
        }
        public User LastSelectedUser
        {
            get { return _lastSelectedUser; }
            set { _lastSelectedUser = value; }
        }
        public bool ServerError
        {
            get { return _serverError; }
            set { _serverError = value; }
        }
        public int DayOldMessages
        {
            get { return _dayOldMessages; }
            set { _dayOldMessages = value; }
        }
        /////////Constructeur////////
        public FrmProgram(Controler c)
        {
            InitializeComponent();
            this.Ctrl = c;
            this.ServerError = false;
            this.UpdateLstUser = new Thread(new UpdateUser(this,
            this.Ctrl).Init); this.UpdateLstUser.IsBackground = true;
            this.UpdateLstUser.Start();
            this.UserConnected = this.Ctrl.GetUserConnected();
            this.tbxUser.Text = Environment.NewLine +
            this.UserConnected.GetIdUser().Split('@')[0];
            this.NbMessages = 0;
            this.DayOldMessages = 0;
            // this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
            this.UserConnected.GetIdUser(), true);
        }
        ////méthodes de la fenêtre//////////
        private void FrmProgram_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (!this.ServerError)
            {
                DialogResult answer;
                FrmExit exit = new FrmExit();
                answer = exit.ShowDialog();
                if (answer == DialogResult.Cancel)
                {
                    // empêche la fermeture de la
                    fenêtre e.Cancel = true;
                }
                else
                {
```

```
                if (answer == DialogResult.OK)
                {
                    try
                    {
                        this.Ctrl.CloseConnection();
                        Process.GetCurrentProcess().Kill();
                    }
                    catch (Exception)
                    {
                        Process.GetCurrentProcess().Kill();
                    }
                }
                else
                {
                    try
                    {
                        this.Ctrl.CloseConnection();
                        this.Ctrl.VisibleChange(true);
                    }
                    catch (Exception)
                    {
                        this.Ctrl.VisibleChange(true);
                    }
                }
            }
        }
        else
        {
            try
            {
                this.Ctrl.CloseConnection();
                this.Ctrl.VisibleChange(true);
            }
            catch (Exception)
            {
                this.Ctrl.VisibleChange(true);
            }
        }
    }
    private void lsbEmployees_DrawItem(object sender, DrawItemEventArgs e)
    {
        //repris d'un exercice fait avec
        M.Beney if (e.Index < 0) return;
        User userDrawing = this.lsbEmployees.Items[e.Index] as User;
        // Si l'état de l'élément est sélectionné alors change la couleur de sélection...
        if ((e.State & DrawItemState.Selected) == DrawItemState.Selected)
            e = new DrawItemEventArgs(e.Graphics,
                                      e.Font,
                                      e.Bounds,
                                      e.Index,
                                      e.State ^ DrawItemState.Selected, // ^ --> XOR
                                      logique
                                      e.ForeColor,
                                      Color.LightBlue);//Couleur de son choix
        e.DrawBackground();
        // Définition du pinceau par défaut en noir...
        Brush myBrush = Brushes.Black;
```

```
            Pen myPen = new Pen(Color.Black);
            myPen.Width = 2;
            if (userDrawing.GetInformationConnection())
                myBrush = Brushes.Green;
            else
                myBrush = Brushes.Red;
            // Dessine un Cercle rouge ou vert
            e.Graphics.FillEllipse(myBrush, e.Bounds.Left + 8, e.Bounds.Top + 15, 12, 12);
            e.Graphics.DrawEllipse(myPen, e.Bounds.Left + 8, e.Bounds.Top + 15, 12, 12);
            myBrush = Brushes.Black;
            e.Graphics.DrawString(userDrawing.GetIdUser().Split('@')[0], new Font("Arial",
            10, FontStyle.Bold), myBrush, e.Bounds.Left + 30, e.Bounds.Top + 15,
            StringFormat.GenericTypographic);
            // Dessine un Rectangle gris autour de chaque
            éléments myBrush = Brushes.Red;
            e.Graphics.DrawString(userDrawing.GetAdmin(), new Font("Arial", 8,
            FontStyle.Bold), myBrush, e.Bounds.Left + 30, e.Bounds.Top + 32,
            StringFormat.GenericTypographic);
            myPen.Color = Color.LightGray; myPen.Width =
            1; e.Graphics.DrawRectangle(myPen,
            e.Bounds); if
            (userDrawing.GetMessagesNotRead() != 0)
            {
                myBrush = Brushes.Yellow;
                myPen.Color = Color.Black;
                myPen.Width = 2;
                e.Graphics.FillEllipse(myBrush, e.Bounds.Left + 105, e.Bounds.Top + 15, 20,
                20);
                e.Graphics.DrawEllipse(myPen, e.Bounds.Left + 105, e.Bounds.Top + 15, 20,
                20); myBrush = Brushes.Black;

                if (userDrawing.GetMessagesNotRead() >= 100)
                {
                    e.Graphics.DrawString(userDrawing.GetMessagesNotRead().ToString(), new
                    Font("Arial", 8, FontStyle.Bold), myBrush, e.Bounds.Left +
                    106, e.Bounds.Top + 18, StringFormat.GenericTypographic);
                }
                else if (userDrawing.GetMessagesNotRead() < 10)
                {

                    e.Graphics.DrawString(userDrawing.GetMessagesNotRead().ToString(), new
                    Font("Arial", 8, FontStyle.Bold), myBrush, e.Bounds.Left +
                    112, e.Bounds.Top + 18, StringFormat.GenericTypographic);
                }
                else
                {
                    e.Graphics.DrawString(userDrawing.GetMessagesNotRead().ToString(), new
                    Font("Arial", 8, FontStyle.Bold), myBrush, e.Bounds.Left +
                    110, e.Bounds.Top + 18, StringFormat.GenericTypographic);
                }
            }
            // If the ListBox has focus, draw a focus rectangle around the selected
            item. e.DrawFocusRectangle();
        }
        private void btnSend_Click(object sender, EventArgs e)
        {
            try
            {
                string allDestinations = string.Empty;
                bool first = true;
                User destination = lsbEmployees.SelectedItem as User;
                if (tbxWriteMessage.Text.Trim() != "" && destination != null)
                {
                    if (lsbEmployees.SelectedIndex == 0)
                    {

                        foreach (User user in lsbEmployees.Items)
                        {
                            if (user.GetIdGroup() == this.UserConnected.GetIdGroup())
                            {
                                if (first)
                                {
                                    first = false;
                                }
                                else
                                {
                                    allDestinations += user.GetIdUser() + "!";
                                }
                            }
                        }

                        if (this.DayOldMessages != 0)
                        {
                            this.GetOldConversation();
                            Thread.Sleep(200);
                        }
                        this.Ctrl.SendMessageGroup(this.UserConnected.GetIdUser(),
                        allDestinations, this.tbxWriteMessage.Text, true);

                        Thread.Sleep(10);
                        this.UpdateStateMessagesGroup();
                        Thread.Sleep(10);
                        this.Ctrl.UpdateUsers(this.UserConnected.GetNameGroup(),
                        this.UserConnected.GetIdUser(),
                        this.UserConnected.GetIdGroup()); this.tbxWriteMessage.Clear();
                    }
                    else
                    {
                        if (this.DayOldMessages != 0)
                        {
                            this.GetOldConversation();
                            Thread.Sleep(200);
                        }
                        this.Ctrl.SendMessage(this.UserConnected.GetIdUser(),
                        destination.GetIdUser(), this.tbxWriteMessage.Text, false);

                        Thread.Sleep(10);
                        this.UpdateStateMessagesOneUser();
                        Thread.Sleep(10);
                        this.Ctrl.UpdateUsers(this.UserConnected.GetNameGroup(),
                        this.UserConnected.GetIdUser(), this.UserConnected.GetIdGroup());
                        this.tbxWriteMessage.Clear();
                    }
                }
            }
            catch (Exception)
            {
                this.ServerClosed();
            }
        }
        private void lsbEmployees_SelectedIndexChanged(object sender, EventArgs e)
        {
            try
            {
                User user = this.lsbEmployees.SelectedItem as
                User; if (user != null)
                {

                    if (this.LastSelectedUser != null)
                    {
                        if (this.LastSelectedUser.GetIdUser() != user.GetIdUser())
                        {
                            this.NbMessages = 0;
                            this.tbxMessages.Clear();
                            this.LastAuthor = string.Empty;
                            if (lsbEmployees.SelectedIndex != 0)
                            {
                                if (this.DayOldMessages != 0)
                                {
                                    this.GetOldConversation();
                                    Thread.Sleep(3);
                                }

                                this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
                                user.GetIdUser(), false);
                                Thread.Sleep(200);
                                this.UpdateStateMessagesOneUser();
                            }
                            else
                            {
                                if (this.DayOldMessages != 0)
                                {
                                    this.GetOldConversation();
                                    Thread.Sleep(200);
                                }
                                this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
                                user.GetIdUser(), true);
                                Thread.Sleep(20);

                                Thread.Sleep(20);
                                this.UpdateStateMessagesGroup();
                            }

                            this.LastSelectedUser = user;
                        }
                    }
                    else
                    {
                        this.LastSelectedUser = user;
                        foreach (User userInfo in this.lsbEmployees.Items)
                        {
```

```
                        Thread.Sleep(3);
                        this.Ctrl.UpdateStateMessages(userInfo.GetIdUser(),
                        this.UserConnected.GetIdUser(), true,
                        this.UserConnected.GetNameGroup(),
                        this.UserConnected.GetIdGroup(), this.UserConnected.GetIdUser());
                        Thread.Sleep(3);
                        if (this.DayOldMessages != 0)
                        {
                            this.GetOldConversation();
                            Thread.Sleep(200);
                        }

                        this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
                        user.GetIdUser(), true);
                    }
                }

            }

        }
        catch (Exception)
        {
            this.ServerClosed();
        }
    }
    private void tsmiQuit_Click(object sender, EventArgs e)
    {
        this.Close();
    }
    private void tsmiOldMesssage_Click(object sender, EventArgs e)
    {
        try
        {
            this.LastAuthor = "";
            tbxMessages.Clear();
            User user = lsbEmployees.SelectedItem as User;
            ToolStripMenuItem tsmiFocus = sender as ToolStripMenuItem;
            foreach (ToolStripMenuItem tsmi in this.tsmiOldMessages.DropDownItems)
            {

                tsmi.Checked = false;
            }
            tsmiFocus.Checked = true;
            this.DayOldMessages = Convert.ToInt32(tsmiFocus.Tag);
            Thread.Sleep(10);
            if (lsbEmployees.SelectedIndex != 0)
            {
                if (this.DayOldMessages != 0)
                {
                    this.Ctrl.GetOldMessages(this.UserConnected.GetIdUser(),
                    user.GetIdUser(), false, this.DayOldMessages);
                    Thread.Sleep(10);
                    this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
                    user.GetIdUser(), false);
                }
                else
                {
                    this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
                    user.GetIdUser(), false);
                }
            }
            else
            {
                if (this.DayOldMessages != 0)
                {
                    this.Ctrl.GetOldMessages(this.UserConnected.GetIdUser(),
                    user.GetIdUser(), true, this.DayOldMessages);
                    Thread.Sleep(10);
                    this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
                    user.GetIdUser(), true);
                }
                else
                {
                    this.Ctrl.GetConversation(this.UserConnected.GetIdUser(),
                    user.GetIdUser(), true);
                }
            }
            this.NbMessages = 0;
        }
        catch (Exception)
        {

            this.ServerClosed();
        }
    }
    private void tsmiAbout_Click(object sender, EventArgs e)
    {
        FrmAbout about = new
        FrmAbout(); about.ShowDialog();
    }
    private void tsmiOldMessages_Click(object sender, EventArgs e)
    {
    }
    private void tsmiSettings_Click(object sender, EventArgs e)
    {
        DialogResult res = new DialogResult();
        FrmSettings settings = new FrmSettings(this,
        this.UserConnected.GetPassword()); res = settings.ShowDialog();
        if (res == DialogResult.OK)
        {
            try
            {
                if (settings.GetNewPassword() != string.Empty)
                {
                    this.Ctrl.ChangePassword(this.UserConnected.GetIdUser(),
                    settings.GetNewPassword());
                }
                else
                {
                    tsmiSettings_Click(sender, e);
                }
            }
            catch (Exception)
```

```
                {
                    this.ServerClosed();
                }
            }
        }
        private void tsmDateTime_Tick(object sender, EventArgs e)
        {
            tssDate.Text = DateTime.Now.ToLocalTime().ToString() + " (Heure UTC)";
        }
        ////////méthodes///////
        /// <summary>
        /// permet de mettre à jour la liste des employés
        /// </summary>
        /// <param name="listUsers">liste d'employés</param>
        public void SetEmployees(List<User> listUsers)
        {
            this.LstUser = listUsers;
            Invoke(new MethodInvoker(delegate
            {
                try
                {
                    int getLastSelected = lsbEmployees.SelectedIndex;
                    this.LastSelectedUser = lsbEmployees.SelectedItem as
                    User; this.lsbEmployees.DataSource = null;
                    this.lsbEmployees.DataSource = listUsers;

                    if (getLastSelected < 0)
                    {
                        this.lsbEmployees.SelectedIndex = 0;
                    }
                    else
                    {
                        this.lsbEmployees.SelectedIndex = getLastSelected;
                    }
                }
                catch (Exception)
                {

                    this.Close();
                }
            }));
        }
        /// <summary>
        /// permet d'afficher les messages
        /// </summary>
        /// <param name="lstNewMessages">liste des messages</param>
        /// <param name="destination">destinataire</param>
        /// <param name="iduser">envoyeur</param>
        /// <param name="isforGroup">si c'est envoyé au groupe</param>
        public void ShowMessages(List<Message> lstNewMessages, string destination,
        string iduser, bool isforGroup)
        {

            Invoke(new MethodInvoker(delegate
            {
                User user = this.lsbEmployees.SelectedItem as User;
                if (user != null)
                {
                    if ((user.GetIdUser() == destination || user.GetIdUser() == iduser
                    && user.GetIdUser().Contains("@")) || (isforGroup &&
                    lsbEmployees.SelectedIndex == 0))
                    {

                        string messages = string.Empty;
                        for (int i = this.NbMessages; i < lstNewMessages.Count; i++)
                        {

                            Message msg = lstNewMessages[i] as Message;


                            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();
                        }

                        this.NbMessages = lstNewMessages.Count;
                        tbxMessages.AppendText(messages);
                    }
                }
                else
                {
                    this.ServerClosed();
                }
            }));
        }
        /// <summary>
        /// permet de donner la connexion du client
        /// </summary>
        /// <returns>connexion du client</returns>
        public TcpClient GetTcpClient()
        {
            return this.Ctrl.TClient;
        }
        /// <summary>
        /// permet de donner le flux d'information du client
        /// </summary>
        /// <returns>flux d'information du client</returns>
        public NetworkStream GetNetStream()
        {
            return this.Ctrl.Stream;
        }
        /// <summary>
        /// permet de décoder le message
        /// </summary>
        /// <param name="message">message codé</param>
        /// <returns>message original</returns>
        public string DecryptMessage(string message)
        {
            return this.Ctrl.DecryptMessage(message);
        }
        /// <summary>
        /// fait quitter le programme à l'utilisateur
        /// </summary>
        public void ServerClosed()
        {
            this.ServerError = true;
            MessageBox.Show("Le serveur a été éteint. Vous allez être automatiquement
            déconnecté.", "Serveur inaccessible", MessageBoxButtons.OK,
            MessageBoxIcon.Information);

            this.Close();
        }
        /// <summary>
        /// fait quitter le programme à l'utilisateur
        /// </summary>
        public void DatabaseClosed()
        {
            this.ServerError = true;
            MessageBox.Show("La base de données a été éteinte. Vous allez être
            automatiquement déconnecté.", "Base de données inaccessible",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
            Invoke(new MethodInvoker(delegate
            {
                this.Close();
            }));
        }
        /// <summary>
        /// permet de ^mettre à jour les états des messages de son groupe
        /// </summary>
        private void UpdateStateMessagesGroup()
        {

            foreach (User userInfo in this.lsbEmployees.Items)
            {
                if (userInfo.GetIdUser().Contains("@") && userInfo.GetIdGroup()
                == this.UserConnected.GetIdGroup())
                {
                    this.Ctrl.UpdateStateMessages(userInfo.GetIdUser(),
                    this.UserConnected.GetIdUser(), true, this.UserConnected.GetNameGroup(),
                    this.UserConnected.GetIdGroup(), this.UserConnected.GetIdUser());
                    Thread.Sleep(4);
                }
            }
        }
        /// <summary>
        /// permete de mettre à jour les états des messages entre deux utilisateurs
        /// </summary>
        private void UpdateStateMessagesOneUser()
        {
            User user = this.lsbEmployees.SelectedItem as User;
            this.Ctrl.UpdateStateMessages(user.GetIdUser(), this.UserConnected.GetIdUser(),
                false, this.UserConnected.GetNameGroup(), this.UserConnected.GetIdGroup(),
                this.UserConnected.GetIdUser());
        }
        /// <summary>
        /// permet de récupérer les anciennes converstaions présent dans la base de données
        /// </summary>
        private void GetOldConversation()
        {
            foreach (ToolStripMenuItem tsmi in this.tsmiOldMessages.DropDownItems)
            {
                User user = lsbEmployees.SelectedItem as User;

                if (tsmi.Checked)
                {
                    if (lsbEmployees.SelectedIndex != 0)
                    {
                        this.Ctrl.GetOldMessages(this.UserConnected.GetIdUser(),
                        user.GetIdUser(), false, this.DayOldMessages);
                    }
                    else
                    {
                        this.Ctrl.GetOldMessages(this.UserConnected.GetIdUser(),
                        user.GetIdUser(), true, this.DayOldMessages);
                    }

                    if (Convert.ToInt32(tsmi.Tag) != this.DayOldMessages)
                    {
                        tbxMessages.Clear();
                        this.NbMessages = 0;
                    }

                    break;
                }
            }
        }
        // <summary>
        /// permet de coder le mot de passe de l'utilisateur
        /// </summary>
        /// <param name="password">mot de passe de l'utilisateur</param>
        /// <returns></returns>
        public string Sha1(string password)
        {
            return this.Ctrl.Sha1(password);
        }
        /// <summary>
        /// permet de savoir si le mot de passe a bel et bien été enregistré dans la base
        de données
        /// </summary>
        /// <param name="isChanged">si le changement c'est effectué</param>
        /// <param name="password"> le nouveau mot de passe de l'utilisateur </param>
        public void PasswordIsChanged(bool isChanged, string password)
        {
            if (isChanged)
            {
                MessageBox.Show("Votre nouveau mot de passe a été enregistré.", "Le mot
                de passe a été modifié", MessageBoxButtons.OK,
                MessageBoxIcon.Information); this.UserConnected.SetPassword(password);
            }
            else
            {
                MessageBox.Show("Votre nouveau mot de passe n'a pas été enregistré.", "Le
                mot de passe n'a pas pu être modifié", MessageBoxButtons.OK,
                MessageBoxIcon.Warning);
                this.DatabaseClosed();
            }
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.ComponentModel; using
System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace talkEntreprise_client
{
    public partial class FrmSettings : Form
    {
        ////Champs//////
        private  string  _oldPassword;
        private  string  _newPassword;
        private FrmProgram _prog;
        ////propriétées//////
        public string OldPassword
        {
            get { return _oldPassword; }
            set { _oldPassword = value; }
        }
        public string NewPassword
        {
            get { return _newPassword; }
            set { _newPassword = value; }
        }
        public FrmProgram Prog
        {
            get { return _prog; }
            set { _prog = value; }
        }
        public FrmSettings(FrmProgram p, string pwd)
        {
            InitializeComponent();
            this.Prog = p;
            this.OldPassword = pwd;
            this.NewPassword = string.Empty;
        }
        private bool PasswordIsOk()
        {
            if (this.OldPassword == this.Prog.Sha1(tbxOldPassword.Text))
            {
                if (tbxNewPassword.Text.Trim().Length >= 6)
                {
                    return true;
                }
                else
                {
                    MessageBox.Show("Votre nouveau mot de passe est trop court", "Erreur",
                    MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    return false;
                }

            }
            else
            {
                MessageBox.Show("Vous avez tapé le mauvais mot de passe", "Erreur",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return false;
            }
        }
        public string GetNewPassword()
        {
            return this.NewPassword;
        }
        private void btnChange_Click(object sender, EventArgs e)
        {
            if (this.PasswordIsOk())
            {
                this.NewPassword = this.Prog.Sha1(tbxNewPassword.Text);
            }
        }
    }
}
```

```
/******************************************
* Projet : TalkEntreprise_client
* Description : création d'une messagerie instantanée
* Date : juin 2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Security.Cryptography;
using System.IO;

namespace talkEntreprise_client
{
    public class ManageMessages
    {
        //////Champs//////////////
        private Controler _ctrl;
        private DESCryptoServiceProvider _key;
        //////propriétées//////////////
        public DESCryptoServiceProvider Key
        {
            get { return _key; }
            set { _key = value; }
        }
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        //////Constructeur/////////////
        public ManageMessages(Controler c)
        {
            this.Ctrl = c;
            this.Key = new DESCryptoServiceProvider();
            this.Key.Key = new byte[8] { 178, 107, 216, 40, 30, 50, 250, 253 };
            this.Key.IV = new byte[8] { 249, 169, 4, 183, 39, 35, 176, 26 };
        }
        //////méthodes/////////////
        /// <summary>
        /// permet de crypter le message en tableau de byte
        /// </summary>
        /// <param name="strText">message</param>
        /// <param name="key">clée d'encodage</param>
        /// <returns>messages crypté</returns>
        public static byte[] Encrypt(string strText, SymmetricAlgorithm key)
        {
            // Create a memory stream.
            MemoryStream ms = new MemoryStream();
            // Create a CryptoStream using the memory stream and the
            // CSP(cryptoserviceprovider) DES key.
            CryptoStream crypstream = new CryptoStream(ms, key.CreateEncryptor(key.Key,
            key.IV), CryptoStreamMode.Write);
            // Create  a  StreamWriter  to  write  a  string  to  the
            stream. StreamWriter sw = new StreamWriter(crypstream);
            // Write the strText to the stream.
            sw.WriteLine(strText);
            // Close the StreamWriter and
            CryptoStream. sw.Close();
            crypstream.Close();
            // Get  an  array  of  bytes  that  represents  the  memory
            stream. byte[] buffer = ms.ToArray();
            // Close the memory stream.
            ms.Close();
            // Return the encrypted byte
            array. return buffer;
        }
        /// <summary>
        /// permet de décripter le message
        /// </summary>
        /// <param name="encryptText">message encrypté</param>
        /// <param name="key">clée de cryptage</param>
        /// <returns>message décrypté</returns>
        public static string Decrypt(byte[] encryptText, SymmetricAlgorithm key)
        {
            // Create a memory stream to the passed buffer.
            MemoryStream ms = new MemoryStream(encryptText);
            // Create a CryptoStream using  memory stream and CSP DES key.
            CryptoStream crypstream = new CryptoStream(ms,
            key.CreateDecryptor(key.Key, key.IV), CryptoStreamMode.Read);

            // Create  a  StreamReader  for  reading  the  stream.
            StreamReader sr = new StreamReader(crypstream);

            // Read the stream as a string.
            string val = sr.ReadLine();

            // Close the
            streams. sr.Close();
            crypstream.Close();
            ms.Close();

            return val;
        }

/// <summary>
/// permet de décrypter un message
/// </summary>
/// <param name="message">message de l'utilisateur</param>
/// <returns>message codé</returns>
        public string EncryptMessage(string message)
        {
            string msg = string.Empty;
            byte[] EncryptedMessage = Encrypt(message, Key);

            for (int i = 0; i < EncryptedMessage.Length; i++)
            {
                if (i + 1 == EncryptedMessage.Length)
                {
                    msg += EncryptedMessage[i].ToString();
                }
                else
                {
                    msg += EncryptedMessage[i].ToString() + ",";
                }
            }
            return msg;
        }
        /// <summary>
        /// permet de décoder le message
        /// </summary>
        /// <param name="message">message codé</param>
        /// <returns>message original</returns>
        public string DecryptMessage(string message)
        {
            List<byte> EncryptedMessage = new List<byte>();
            foreach (string msgFrag in message.Split(','))
            {
                EncryptedMessage.Add(Convert.ToByte(msgFrag));
            }
            return Decrypt(EncryptedMessage.ToArray(), Key);

        }



    }
}
```