```
/****************************************
* Projet : TalkEntreprise_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* Version : 1.0
* Auteur :Gabriel Strano
*
****************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace talkEntreprise_server
{
    public class Converter
    {
        /////Champs/////
        private Controler _ctrl;
        /////propriétées/////
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        /////Constructeur/////
        public Converter(Controler cont)
        {
            this.Ctrl = cont;
        }
        /////méthodes/////
        /// <summary>
        /// permet de convertire un nombre en hexadécimal
        /// </summary>
        /// <param name="number">nombre à convertire</param>
        /// <returns>nombre en hexadécimal</returns>
        public string NumberToHexadecimal(int number)
        {
            //aide pour la conversion :
            http://stackoverflow.com/questions/74148/how-to-convert-numbers-between-hexadecimal-and-decimal-in-c
            // aide pour le formattage :
            http://stackoverflow.com/questions/11618387/string-format-for-hex
            return string.Format("{0:x4}", number).ToUpper();
        }
        /// <summary>
        /// Convertire de l'héxadécimal en nombre
        /// </summary>
        /// <param name="hexa">nombre hexadécimal à convertire</param>
        /// <returns>nombre</returns>
        public long HexadecimalToNumber(string hexa)
        {
            return Convert.ToInt64(hexa, 16);
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* 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.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using talkEntreprise_server.classThread;

namespace talkEntreprise_server
{
    public partial class FrmConnection : Form
    {
        ////Champs////////
        private Controler _ctrl;
        ////Propriétées///////
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        ////////Constructeur/////////
        public FrmConnection()
        {
            InitializeComponent();
            this.Ctrl = new Controler(this);
            this.Ctrl.SetAllEmployeesDeconnected();
        }
        //////méthodes////
        private void FrmConnection_FormClosing(object sender, FormClosingEventArgs e)
        {
            Application.Exit();
        }
        private void btnConnect_Click(object sender, EventArgs e)
        {
            if (this.Ctrl.ValidateConnectionAdmin(tbxId.Text,
            this.Ctrl.Sha1(tbxPassword.Text)))
            {
                this.IsVisible();
                List<string> lstInfo = this.Ctrl.GetInformation(tbxId.Text);
                User user = new User(tbxId.Text, lstInfo[2], Convert.ToInt32(lstInfo[0]),
                true, 0, lstInfo[1]);
                this.Ctrl.SucessConnectionToServer(tbxId.Text);
                this.Ctrl.CreateProgram(tbxId.Text, user);
            }
```

```
            else
            {
                MessageBox.Show("Identifiant ou mot de passe incorrecte. ", "Connexion non
                valide", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }
        private void btnQuit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
        /// <summary>
        /// changer l'état visuel de la forme de connexion
        /// </summary>
        public void IsVisible()
        {
            Invoke(new MethodInvoker(delegate
            {
                this.Visible = !this.Visible;
            }));

        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* 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_server
{
    public partial class FrmProgram : Form
    {
        //////Champs////
        private Controler _ctrl;
        private User _userConnected;
        //////Propriétées////////
        public User UserConnected
        {
            get { return _userConnected; }
            set { _userConnected = value; }
        }
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        ////////Constructeur////////
        public FrmProgram(Controler c, User u)
        {
            InitializeComponent();
            this.Ctrl = c;
            this.UserConnected = u;
        }

        private void FrmProgram_FormClosing(object sender, FormClosingEventArgs e)
        {
            this.Ctrl.isVisible();
            this.Ctrl.DeconnectionToServer(this.UserConnected.GetIdUser());
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace talkEntreprise_server
{
    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 = this.GetFormatedDate(valueDate);

        }
        //////méthodes//////
        public string GetFormatedDate(string oldDate)
        {
            bool first = true;
            string[] InglobalDateOrHour;
            string res = string.Empty;
            foreach (string globalDateOrHour in oldDate.Split('_'))
            {
                InglobalDateOrHour = globalDateOrHour.Split('-');
                if (first)
                {
```

```
                    res += InglobalDateOrHour[2] + "." + InglobalDateOrHour[1] + "." +
                    InglobalDateOrHour[0] + " ";
                    first = false;
                }
                else
                {
                    res += InglobalDateOrHour[0] + "." + InglobalDateOrHour[1] + "." +
                    InglobalDateOrHour[2];
                }
            }
            return res;
        }
        public string GetDate()
        {
            return this.Date;
        }
        public string GetAuthor()
        {
            return this.Author;
        }
        public string GetContent()
        {
            return this.Content;
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace talkEntreprise_server
{
    public class RequestSQL
    {
        ///////Champs//////
        private const int STATEDEFAULT = 2;
        private const int STATENOTREAD = 3;
        private const int STATEREAD = 4;
        private const int IDADMINISTRATORS = 3;
        private MySqlConnection _connectionUser;
        /////propriétées/////
        public MySqlConnection ConnectionUser
        {
            get { return _connectionUser; }
            set { _connectionUser = value; }
        }
        private Controler _ctrl;

        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }

        ///////Constructeur////

        public RequestSQL(Controler c)
        {
            this.Ctrl = c;
        }
        ///////méhthodes////
        /// <summary>
        /// permet d'initialiser la connexion à la base de données
        /// </summary>
        /// <returns>true false</returns>
        public bool ConnectionDB()
        {
            try
            {
                string connectionString =
                @"server=127.0.0.1;userid=IT;password=Super;database=db_talkEntreprise";
                this.ConnectionUser = new MySqlConnection(connectionString);
```

```
            this.ConnectionUser.Open();
            return true;
        }
        catch (MySqlException ex)
        {
            return false;
        }
    }
    /// <summary>
    /// permet de fermer la connexion à la base de données
    /// </summary>
    public void ShutdownConnectionDB()
    {
        ConnectionUser.Close();
    }
    /// <summary>
    /// Permet de vérifier si les informations entrées par l'utilisateur sont valide ou
    non
    /// </summary>
    /// <param name="user">Identifiant de l'utilisateur</param>
    /// <param name="password"> mot de passe de l'utilisateur</param>
    /// <returns>retourne "true" si l'utilisateur à les bonnes informations de
    connection</returns>
    ///
    public Boolean ValidateConnectionUser(string user, string password)
    {
        bool result = false;
        if (this.ConnectionDB())
        {
            string sql = String.Format("SELECT Count(*) as total FROM t_users where
            idUser  = '{0}' AND Password = '{1}'", user, password);
            MySqlCommand cmd = new MySqlCommand(sql, ConnectionUser);
            cmd.ExecuteNonQuery();
            MySqlDataReader reader = cmd.ExecuteReader();
            reader.Read();
            if (Convert.ToInt32(reader.GetString("total")) == 1)
            {
                result = true;
            }
            reader.Close();

            this.ShutdownConnectionDB();
        }
        return result;
    }
    /// <summary>
    /// permet de récupérer les informations de l'utilisateur demmandé
    /// </summary>
    /// <param name="user">identifiant de l'utilisateu</param>
    /// <returns>list des informations de l'utilisateur</returns>
    public List<string> GetInformation(string user)
    {
        List<string> lstInfoUser = new List<string>();

        if (this.ConnectionDB())
        {
            string sql = String.Format("SELECT u.idGroup, g.group, password FROM t_users
```

```
            u, t_group g where u.idGroup = g.idGroup AND  idUser  = '{0}'", user);
            MySqlCommand cmd = new MySqlCommand(sql, ConnectionUser);
            cmd.ExecuteNonQuery();
            MySqlDataReader reader = cmd.ExecuteReader();
            reader.Read();
            lstInfoUser.Add(reader.GetString("idGroup"));
            lstInfoUser.Add(reader.GetString("group"));
            lstInfoUser.Add(reader.GetString("password"));
        }
        return lstInfoUser;
    }

    /// <summary>
    /// permet de dire que l'utilisateur c'est connecté sur le server --> log de la base
    de données
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    public void SucessConnectionToServer(string user)
    {
        if (this.ConnectionDB())
        {
            string destination = "Host";
            long lastId = 0;

            string sql = string.Format("INSERT INTO t_log
            (`Code`,`lenTot`,`CodeSender`,`lenSender`,`valueSender`,`CodeDestination`,`len
            Destination`,`valueDestination`,`valueDate`,`CodeEnd`) VALUES( '{0}', '{1}'
            , '{2}' , '{3}' , '{4}', '{5}', '{6}', '{7}', '{8}', '{9}')",
                "0001",//0 quel type de message
                "0000", //1 longueur de tout la chaine (depuis idCode sender à idCodeEnd)
                "0011", //2 Code Envoye
                this.Ctrl.NumberToHexadecimal(user.Count()),//3 longueur de l'identifiant
                user, //4 identifiant de la personne
                "0012",//5 Code Destinataire
                this.Ctrl.NumberToHexadecimal(destination.Count()), //6 longueur du
                destinataire
                destination, //7 identifiant du destinataire
                DateTime.Now.ToUniversalTime().ToString("yyyy-MM-dd_HH-mm-ss"),//date de
                la connection (heure universelle)
                "#####" // Code de fin
                );
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            cmd.ExecuteNonQuery();
            lastId = cmd.LastInsertedId;
            cmd.CommandText = String.Format("SELECT * FROM `t_log` where idLog = {0}",
            lastId);
            MySqlDataReader reader = cmd.ExecuteReader();
            reader.Read();
            sql = String.Format("Update `t_log` set lenTot = '{0}' where idLog = {1}",
            this.Ctrl.NumberToHexadecimal(
                reader.GetString("CodeSender").Count() +
                reader.GetString("lenSender").Count() +
                reader.GetString("valueSender").Count()
                + reader.GetString("CodeDestination").Count() +
                reader.GetString("lenDestination").Count() +
                reader.GetString("valueDestination").Count()
                + reader.GetString("CodeDate").Count() +
```

```
                reader.GetString("lenDate").Count() +
                reader.GetString("valueDate").Count()
                ), lastId);
            reader.Close();
            cmd.CommandText = sql;
            cmd.ExecuteNonQuery();
            cmd.CommandText = String.Format("Update `t_users` set Connection = 1 where
            idUser = '{0}'", user);
            cmd.ExecuteNonQuery();
            this.ShutdownConnectionDB();
        }
    }
    /// <summary>
    /// permet de dire que l'utilisateur c'est déconnecté du serveur
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    public void DeconnectionToServer(string user)
    {
        if (ConnectionDB())
        {
            string destination = "Host";
            long lastId = 0;
            string sql = string.Format("INSERT INTO t_log
            (`Code`,`lenTot`,`CodeSender`,`lenSender`,`valueSender`,`CodeDestination`,`len
            Destination`,`valueDestination`,`valueDate`,`CodeEnd`) VALUES( '{0}', '{1}'
            , '{2}' , '{3}' , '{4}', '{5}', '{6}', '{7}', '{8}', '{9}')",
                "0002",//0 quel type de message
                "0000", //1 longueur de tout la chaine (depuis idCode sender à
                idCodeEnd)
                "0011", //2 Code Envoye
                this.Ctrl.NumberToHexadecimal(user.Count()),//3 longueur de
                l'identifiant
                user, //4 identifiant de la personne
                "0012",//5 Code Destinataire
                this.Ctrl.NumberToHexadecimal(destination.Count()), //6 longueur du
                destinataire
                destination, //7 identifiant du destinataire
                 DateTime.Now.ToUniversalTime().ToString("yyyy-MM-dd_HH-mm-ss"),//date
                 de la connection (heure universelle)
                "#####" // Code de fin
                );
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            cmd.ExecuteNonQuery();
            lastId = cmd.LastInsertedId;
            cmd.CommandText = String.Format("SELECT * FROM `t_log` where idLog = {0}",
            lastId);
            MySqlDataReader reader = cmd.ExecuteReader();
            reader.Read();
            sql = String.Format("Update `t_log` set lenTot = '{0}' where idLog = {1}",
            this.Ctrl.NumberToHexadecimal(
                reader.GetString("CodeSender").Count() +
                reader.GetString("lenSender").Count() +
                reader.GetString("valueSender").Count()
                + reader.GetString("CodeDestination").Count() +
                reader.GetString("lenDestination").Count() +
                reader.GetString("valueDestination").Count()
                + reader.GetString("CodeDate").Count() +
```

```
                reader.GetString("lenDate").Count() +
                reader.GetString("valueDate").Count()
                ), lastId);
            reader.Close();
            cmd.CommandText = sql;
            cmd.ExecuteNonQuery();
            cmd.CommandText = String.Format("Update `t_users` set Connection = 0 where
            idUser = '{0}'", user);
            cmd.ExecuteNonQuery();
            this.ShutdownConnectionDB();
        }
    }
    /// <summary>
    /// récupération de la liste des employés
    /// </summary>
    /// <param name="nameGroup">nom du groupe de l'utilisateur</param>
    /// <param name="idGroup">identifiant du groupe de l'utilisateur</param>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <returns>list des employés</returns>
    public List<User> GetUserList(string nameGroup, int idGroup, string user)
    {
        List<User> lsbUsers = new List<User>();
        bool first = true;
        string sql = string.Empty;
        //  bool ConnectedFriend = true;
        //bool NotConnectedFriend = true;
        //  List<string> lsbFriends = new List<string>();
        lsbUsers.Add(new User(nameGroup, "", idGroup, true, 0, nameGroup));
        if (idGroup != IDADMINISTRATORS)
        {
            sql = string.Format("SELECT * From t_users where idGroup = {0} AND idUser !=
            \"{1}\" OR idGroup = {2} AND idUser != \"{3}\" ORDER BY `idUser` ASC",
            IDADMINISTRATORS, user, idGroup, user);
        }
        else
        {
            sql = string.Format("SELECT * From t_users where idGroup = {0} AND idUser !=
            \"{1}\" ORDER BY `idUser` ASC", idGroup, user);
        }

        if (this.ConnectionDB())
        {
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            MySqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                lsbUsers.Add(new User(reader.GetString("idUser"),
                reader.GetString("password"),
                Convert.ToInt32(reader.GetString("idGroup")),
                Convert.ToBoolean(reader.GetString("connection")), 0, nameGroup));
            }
            ShutdownConnectionDB();
        }
        else
        {
            lsbUsers.Add(new User("DB", "", idGroup, false, 0, nameGroup));
        }
```

```
        foreach (User friend in lsbUsers)
        {

            if (first)
            {
                friend.SetMessagesNotRead(GetStatesMessages(user, friend.GetIdUser(),
                true));
                first = false;
            }
            else
            {
                friend.SetMessagesNotRead(GetStatesMessages(user, friend.GetIdUser(),
                false));
            }

        }
        return lsbUsers;
    }
    /// <summary>
    /// récupération de la liste des employés
    /// </summary>
    /// <param name="nameGroup">nom du groupe de l'utilisateur</param>
    /// <param name="idGroup">identifiant du groupe de l'utilisateur</param>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <returns>chaine de caractère contenant la liste des employés</returns>
    public string GetUserListInString(string nameGroup, int idGroup, string user)
    {
        string result = "#0015;";
        List<User> lsbUsers = new List<User>();
        bool first = true;
        string sql;
        lsbUsers.Add(new User(nameGroup, "", idGroup, true, 0, nameGroup));
        if (idGroup != IDADMINISTRATORS)
        {
            sql = string.Format("SELECT * From t_users where idGroup = {0} AND idUser !=
            \"{1}\" OR idGroup = {2} AND idUser != \"{3}\" ORDER BY `idUser` ASC",
            IDADMINISTRATORS, user, idGroup, user);
        }
        else
        {
            sql = string.Format("SELECT * From t_users where idGroup = {0} AND idUser !=
            \"{1}\" ORDER BY `idUser` ASC", idGroup, user);
        }
        if (this.ConnectionDB())
        {
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            MySqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                lsbUsers.Add(new User(reader.GetString("idUser"),
                reader.GetString("password"),
                Convert.ToInt32(reader.GetString("idGroup")),
                Convert.ToBoolean(reader.GetString("connection")), 0, nameGroup));

            }
            ShutdownConnectionDB();
```

```
        }
        else
        {
            lsbUsers.Add(new User("DB", "", idGroup, false, 0, nameGroup));
        }

        foreach (User userInfo in lsbUsers)
        {
            if (first)
            {
                userInfo.SetMessagesNotRead(GetStatesMessages(user,
                userInfo.GetIdUser(), true));
                result += userInfo.GetIdUser() + "," + userInfo.GetPassword() + "," +
                userInfo.GetIdGroup() + "," + userInfo.GetInformationConnection() + ","
                + userInfo.GetMessagesNotRead() + "," + userInfo.GetMessagesNotRead() +
                "," + nameGroup;
                first = false;
            }
            else
            {
                userInfo.SetMessagesNotRead(GetStatesMessages(user,
                userInfo.GetIdUser(), false));
                result += ";" + userInfo.GetIdUser() + "," + userInfo.GetPassword() +
                "," + userInfo.GetIdGroup() + "," + userInfo.GetInformationConnection()
                + "," + userInfo.GetMessagesNotRead() + "," +
                userInfo.GetMessagesNotRead() + "," + nameGroup;
            }
        }
        return result;
    }
    /// <summary>
    /// permet de récupérer les informations relatifs au messages non lu de l'utilisateur
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <param name="userDestination">identifiant d'un autre utilisateur</param>
    /// <param name="forGroup">si le message est envoyé au groupe</param>
    /// <returns>nombre de message eb abscence</returns>
    public int GetStatesMessages(string user, string userDestination, bool forGroup)
    {
        string sql = string.Empty;
        int numberMessages = 0;
        if (forGroup)
        {
            sql = string.Format("SELECT Count( Distinct valueMessage,valueDate) as
            numberMessages FROM t_log"
           + " where  valueDestination=\'{0}\' AND forGroup ={1} AND state={2} OR
           valueDestination=\'{3}\' AND forGroup ={4} AND state={5}  ", user, forGroup,
           STATENOTREAD, user, forGroup, STATEDEFAULT);
        }
        else
        {
            sql = String.Format("SELECT COUNT(*) as numberMessages FROM t_log where
            valueSender = \'{0}\' AND valueDestination = \'{1}\' AND state = {2} AND
            forGroup={3} OR valueSender = \'{4}\' AND valueDestination = \'{5}\' AND
            state= {6} AND forGroup={7} ", userDestination, user, STATEDEFAULT,
            forGroup, userDestination, user, STATENOTREAD, forGroup);
        }
```

```
        if (this.ConnectionDB())
        {
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            MySqlDataReader reader = cmd.ExecuteReader();
            reader.Read();

            numberMessages = reader.GetInt32("numberMessages");
            reader.Close();
            if (forGroup)
            {
                cmd.CommandText = String.Format("Update t_log SET state=\'{0}\' where
                valueDestination = \'{1}\' AND state = {2} AND forGroup={3} ",
                STATENOTREAD, user, STATEDEFAULT, forGroup);
            }
            else
            {
                cmd.CommandText = String.Format("Update t_log SET state=\'{0}\' where
                valueSender = \'{1}\' AND valueDestination = \'{2}\' AND state = {3} AND
                forGroup={4} ", STATENOTREAD, userDestination, user, STATEDEFAULT,
                forGroup);
            }

            cmd.ExecuteNonQuery();
            this.ShutdownConnectionDB();
        }

        return numberMessages;
    }
    /// <summary>
    /// permet  d'enregistrer le message dans la base de données
    /// </summary>
    /// <param name="message">message crypter</param>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <param name="destination">destinataire du message</param>
    /// <param name="forGroup">si c'est pour un groupe</param>
    public void SendMessage(string user, string destination, string message, bool
    forGroup)
    {
        if (this.ConnectionDB())
        {
            long lastInsertId = 0;
            //string date =
            DateTime.Now.ToUniversalTime().AddDays(-1).ToString("yyyy-MM-dd_HH-mm-ss");
            string date = DateTime.Now.ToUniversalTime().ToString("yyyy-MM-dd_HH-mm-ss");
            string sql = String.Format("INSERT INTO t_log
            (`Code`,`lenTot`,`CodeSender`,`lenSender`,`valueSender`,`CodeDestination`,`len
            Destination`,`valueDestination`,`CodeMessage`,`lenMessage`,`valueMessage`,`val
            ueDate`,`CodeEnd`,`forGroup`) VALUES( '{0}', '{1}' , '{2}' , '{3}' , '{4}',
            '{5}', '{6}', '{7}', '{8}', '{9}','{10}','{11}','{12}',{13})",
                "0003",//0 quel type de message
                "0000", //1 longueur de tout la chaine (depuis idCode sender à idCodeEnd)
                "0011", //2 Code Envoye
                this.Ctrl.NumberToHexadecimal(user.Count()),//3 longueur de l'identifiant
                user, //4 identifiant de la personne
                "0012",//5 Code Destinataire
                this.Ctrl.NumberToHexadecimal(destination.Count()), //6 longueur du
```

```
                destinataire
                destination, //7 identifiant du destinataire
                "0020",//8 code pour contenu du message
                this.Ctrl.NumberToHexadecimal(message.Count()),//9 longueur du message
                message, //10 Message à envoyer
                date,//11 date de la connection (heure universelle)
                "#####", //12 Code de fin
                forGroup
                );

            MySqlCommand cmd = new MySqlCommand(sql, ConnectionUser);
            cmd.ExecuteNonQuery();

            lastInsertId = cmd.LastInsertedId;

            cmd.CommandText = String.Format("SELECT * FROM `t_log` where idLog = {0}",
            lastInsertId);

            MySqlDataReader reader = cmd.ExecuteReader();
            reader.Read();

            sql = String.Format("Update `t_log` set lenTot = '{0}' where idLog = {1}",
            this.Ctrl.NumberToHexadecimal(
                reader.GetString("CodeSender").Count() +
                reader.GetString("lenSender").Count() +
                reader.GetString("valueSender").Count()
                + reader.GetString("CodeDestination").Count() +
                reader.GetString("lenDestination").Count() +
                reader.GetString("valueDestination").Count()
                + reader.GetString("CodeMessage").Count() +
                reader.GetString("lenMessage").Count() +
                reader.GetString("valueMessage").Count()
                + reader.GetString("CodeDate").Count() +
                reader.GetString("lenDate").Count() +
                reader.GetString("valueDate").Count()
                ), lastInsertId);
            reader.Close();
            cmd.CommandText = sql;
            cmd.ExecuteNonQuery();

            this.ShutdownConnectionDB();
        }
    }
    /// <summary>
    /// permet de récupérer les messages envoyé par les utilisateur
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <param name="destination">destinataire du messahe</param>
    /// <param name="forGroup">si c'est pour le groupe</param>
    /// <returns></returns>
    public List<Message> GetConversation(string user, string destination, bool forGroup)
    {
        List<Message> lsbMessage = new List<Message>();
        string date = DateTime.Now.ToUniversalTime().ToString("yyyy-MM-dd");
        string LastDay = DateTime.Now.ToUniversalTime().ToString("yyyy-MM-dd");
        string sql = string.Empty;
        if (forGroup)
```

```
        {
            sql = string.Format("SELECT DISTINCT  valueSender, valueMessage,
            DATE_FORMAT(valueDate,'%Y-%m-%d_%H-%i-%S') AS valueDate FROM t_log"
           + " where valueSender=\'{0}\' AND valueMessage!=\"\"  AND forGroup ={1} AND
           valuedate BETWEEN  '{2} 00:00:00' AND '{3} 23:59:59'  OR valueMessage!=\"\"
           AND valueDestination=\'{4}\' AND forGroup ={5} AND valuedate BETWEEN  '{6}
           00:00:00' AND '{7} 23:59:59'"
       , user, forGroup, LastDay, date, user, forGroup, LastDay, date);
        }
        else
        {
            sql = string.Format("SELECT valueSender, valueMessage,
            DATE_FORMAT(valueDate,'%Y-%m-%d_%H-%i-%S') AS valueDate FROM t_log"
          + " where valueSender ='{0}'"
          + "AND valueDestination = '{1}'"
          + " AND forGroup={2} "
           + "AND valuedate BETWEEN  '{3} 00:00:00' AND '{4} 23:59:59'"
          + " OR valueSender ='{5}'"
          + "AND valueDestination = '{6}'"

          + "AND valuedate BETWEEN  '{7} 00:00:00' AND '{8} 23:59:59'"
      + "AND forGroup  ={9} ", user, destination, forGroup, LastDay, date, destination,
      user, LastDay, date, forGroup);
        }

        if (this.ConnectionDB())
        {
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            MySqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                lsbMessage.Add(new Message(reader.GetString("valueSender"),
                reader.GetString("valueMessage"), reader.GetString("valueDate")));
            }
            reader.Close();
            this.ShutdownConnectionDB();
        }
        else
        {
            lsbMessage.Add(new Message("DB", "", "2016-06-09_05-49-44"));
        }
        return lsbMessage;
    }
    /// <summary>
    /// permet de mettre à jour l'état des messages
    /// </summary>
    /// <param name="user"></param>
    /// <param name="destination"></param>
    /// <param name="isforGroup"></param>
    public void UpdateStateMessages(string user, string destination, bool isforGroup)
    {
        string sql = String.Format("UPDATE t_log set state={0} where valueSender =
        \'{1}\' AND valueDestination = \'{2}\' AND state = {3}  AND forGroup={4}  ",
        STATEREAD, user, destination, STATENOTREAD, isforGroup);
        if (this.ConnectionDB())
        {
```

```
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            cmd.ExecuteNonQuery();
            this.ShutdownConnectionDB();
        }
    }
    /// <summary>
    /// permet de mettre à zéro tous les utilisateurs
    /// </summary>
    public void SetAllEmployeesDeconnected()
    {
        string sql = "UPDATE t_users SET connection= false ";
        if (this.ConnectionDB())
        {
            MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
            cmd.ExecuteNonQuery();
            this.ShutdownConnectionDB();
        }
    }
    /// <summary>
    /// permet de récupérer les messages envoyé par les utilisateur
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <param name="destination">destinataire du messahe</param>
    /// <param name="forGroup">si c'est pour le groupe</param>
    /// <returns></returns>
    public List<Message> GetOldConversation(string user, string destination, bool
    forGroup, int nbDays)
    {
        List<Message> lsbMessage = new List<Message>();
        string date = DateTime.Now.ToUniversalTime().AddDays(-1).ToString("yyyy-MM-dd");
        string oldDate =
        DateTime.Now.ToUniversalTime().AddDays(-nbDays).ToString("yyyy-MM-dd");
        string sql = string.Empty;
        if (forGroup)
        {
            sql = string.Format("SELECT DISTINCT  valueSender, valueMessage,
            DATE_FORMAT(valueDate,'%Y-%m-%d_%H-%i-%S') AS valueDate FROM t_log"
           + " where valueSender=\'{0}\' AND valueMessage!=\"\" AND forGroup ={1} AND
           valuedate BETWEEN  '{2} 00:00:00' AND '{3} 23:59:59' OR
           valueDestination=\'{4}\'  AND valuedate BETWEEN  '{5} 00:00:00' AND '{6}
           23:59:59'"
       + "AND forGroup ={7}   ", user, forGroup, oldDate, date, user, oldDate, date,
       forGroup);
        }
        else
        {
            sql = string.Format("SELECT valueSender, valueMessage,
            DATE_FORMAT(valueDate,'%Y-%m-%d_%H-%i-%S') AS valueDate FROM t_log"
          + " where valueSender ='{0}'"
          + "AND valueDestination = '{1}'"
          + " AND forGroup={2} "
           + "AND valuedate BETWEEN  '{3} 00:00:00' AND '{4} 23:59:59'"
          + " OR valueSender ='{5}'"
          + "AND valueDestination = '{6}'"

          + "AND valuedate BETWEEN  '{7} 00:00:00' AND '{8} 23:59:59'"
      + "AND forGroup  ={9} ", user, destination, forGroup, oldDate, date, destination,
```

```
            user, oldDate, date, forGroup);
            }
            if (this.ConnectionDB())
            {
                MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
                MySqlDataReader reader = cmd.ExecuteReader();

                while (reader.Read())
                {
                    lsbMessage.Add(new Message(reader.GetString("valueSender"),
                    reader.GetString("valueMessage"), reader.GetString("valueDate")));
                }
                reader.Close();
                this.ShutdownConnectionDB();
            }
            else
            {
                lsbMessage.Add(new Message("DB", "", "2016-06-09_05-49-44"));
            }
            return lsbMessage;
        }
        /// <summary>
        /// permet de changer le mot de passe de l'utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="password">nouveau mot de passe de l'utilisateur</param>
        /// <returns>réussit ou annuler</returns>
        public bool ChangePassword(string user, string password)
        {
            string sql = string.Format("UPDATE t_users SET password= '{0}' where idUser =
            '{1}' ", password, user);
            if (this.ConnectionDB())
            {
                MySqlCommand cmd = new MySqlCommand(sql, this.ConnectionUser);
                cmd.ExecuteNonQuery();
                this.ShutdownConnectionDB();
                return true;
            }
            else
            {
                return false;
            }
        }
        /// <summary>
        /// Permet de vérifier si les informations entrées par l'administrateur sont valide
        ou non
        /// </summary>
        /// <param name="user">Identifiant de l'administrateur</param>
        /// <param name="password"> mot de passe de l'administrateurn</param>
        /// <returns>retourne "true" si l'administrateur à les bonnes informations de
        connection</returns>
        ///
        public Boolean ValidateConnectionAdmin(string user, string password)
        {
            bool result = false;
            if (this.ConnectionDB())
            {
```

```
                string sql = String.Format("SELECT Count(*) as total FROM t_users where
                idUser  = '{0}' AND Password = '{1}' AND idGroup = {2}", user, password,
                IDADMINISTRATORS);
                MySqlCommand cmd = new MySqlCommand(sql, ConnectionUser);
                cmd.ExecuteNonQuery();
                MySqlDataReader reader = cmd.ExecuteReader();
                reader.Read();
                if (Convert.ToInt32(reader.GetString("total")) == 1)
                {
                    result = true;
                }

                reader.Close();

                this.ShutdownConnectionDB();

            }

            return result;
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using talkEntreprise_server.classThread;

namespace talkEntreprise_server
{
    public class Server
    {
        /////Champs//////
        //tableau contenant les différentes connexions (nom utilisateur, connexion)
        public static Hashtable clientsList = new Hashtable();
        public static Hashtable UserThreadsList = new Hashtable();
        private Controler _ctrl;
        private Thread _firstConnection;
        ///////propriété/////////
        public Controler Ctrl
        {
            get { return _ctrl; }
            set { _ctrl = value; }
        }
        public Thread FirstConnection
        {
            get { return _firstConnection; }
            set { _firstConnection = value; }
        }
        /////Constructeur////
        public Server(Controler c)
        {
            this.Ctrl = c;
            this.FirstConnection = new Thread(new ClientConnectToServ(this).Init);
            this.FirstConnection.IsBackground = true;
            this.FirstConnection.Start();
        }
        /////////méthodes////////
        /// <summary>
        /// permet d'ajouter le nouveau client à la liste de client
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="tcp">connection au client</param>
        public void AddClientList(string user, TcpClient tcp)
        {
            if (!clientsList.ContainsKey(user))
            {
```

```
            clientsList.Add(user, tcp);
        }
    }
    /// <summary>
    /// permet d'ajouter le nouveau processus à la liste de processus
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <param name="t">processus de l'utilisateur</param>
    public void AddThreadList(string user, Thread t)
    {
        if (!UserThreadsList.ContainsKey(user))
        {
            UserThreadsList.Add(user, t);
        }
    }
    /// <summary>
    /// permet de récupérer le processus d'un utilisateur
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <returns>processus de l'utilisateur concerné</returns>
    public Thread GetThreadlist(string user)
    {
        return UserThreadsList[user] as Thread;
    }
    /// <summary>
    /// permet la suppresion de la connection d'un utilisateur
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    public void DelInList(string user)
    {
        UserThreadsList.Remove(user);
        clientsList.Remove(user);
    }
    /// <summary>
    /// permet de savoir su l'utilisateur se trouve dans la base de données
    /// </summary>
    /// <param name="user">identifiant de l'utilisateur</param>
    /// <param name="password">mot de passe de l'utilisateur</param>
    /// <returns></returns>
    public Boolean ValidateConnection(string user, string password)
    {
        return this.Ctrl.ValidateConnection(user, password);
    }
    /// <summary>
    /// permet d'envoyer un message à la base de données pour dir que l'utilisateur
    c'est connectée
    /// </summary>
    /// <param name="user">identifiant de l'utiliseur</param>
    public void SucessConnectionToServer(string user)
    {
        this.Ctrl.SucessConnectionToServer(user);
    }
    /// <summary>
    /// permet d'envoyer l'information à la base de données que l'utilisateur c'est
    déconnecté
    /// </summary>
    /// <param name="user"></param>
```

```
        public void DeconnectionToServer(string user)
        {
            this.Ctrl.DeconnectionToServer(user);
        }
        /// <summary>
        /// permet de récupérer les informations de l'utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <returns></returns>
        public List<string> GetInformation(string user)
        {
            return this.Ctrl.GetInformation(user);
        }
        /// <summary>
        /// récupération de la liste des employés par rapport à l'utilisateur envoyé
        /// </summary>
        /// <param name="nameGroup">nom du groupe des utilisateurs</param>
        /// <param name="idGroup">identifiant du groupe des utilisateurs</param>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <returns></returns>
        public List<User> GetUserList(string nameGroup, int idGroup, string user)
        {
            return this.Ctrl.GetUserList(nameGroup, idGroup, user);
        }
        /// <summary>
        /// permert de récupérer la liste des employés pour l'envoyer au client
        /// </summary>
        /// <param name="nameGroup"> nom du groupe des utilisateur</param>
        /// <param name="idGroup">identifiant du groupe des utilisateur</param>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <returns></returns>
        public string GetUserListInString(string nameGroup, int idGroup, string user)
        {
            return this.Ctrl.GetUserListInString(nameGroup, idGroup, user);
        }
        /// <summary>
        /// récupératuion de la connexion au client par rapport à son identifant de connexion
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <returns>connnexion du client</returns>
        public TcpClient GetTcpClientInClientList(string user)
        {
            return clientsList[user] as TcpClient;
        }
        /// <summary>
        /// vérifie si l'utilisateur est connecté sur le serveur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <returns>connnexion du client</returns>
        public bool IsInClientList(string user)
        {
            return clientsList.ContainsKey(user);
        }

        /// <summary>
        /// permet  d'enregistrer le message dans la base de données
        /// </summary>
```

```
        /// <param name="message">message crypter</param>
        /// <param name="user">nom de l'utilisateur</param>
        /// <param name="destinationUsername">destinataire du message</param>
        /// <param name="forGroup">si c'est pour un groupe</param>
        public void SendMessage(string user, string destinationUsername, string message,
        bool forGroup)
        {
            this.Ctrl.SendMessage(user, destinationUsername, message, forGroup);
        }
        /// <summary>
        /// permet de récupérer les messages envoyé par les utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire du messahe</param>
        /// <param name="forGroup">si c'est pour le groupe</param>
        /// <returns></returns>
        public List<Message> GetConversation(string user, string destination, bool forGroup)
        {
            return this.Ctrl.GetConversation(user, destination, forGroup);
        }
        /// permet de récupérer les anciens messages envoyé par les utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire du messahe</param>
        /// <param name="forGroup">si c'est pour le groupe</param>
        /// <returns></returns>
        public List<Message> GetOldConversation(string user, string destination, bool
        forGroup, int nbDays)
        {
            return this.Ctrl.GetOldConversation(user, destination, forGroup, nbDays);
        }
        /// <summary>
        /// permet de mettre à jour l'état des messages
        /// </summary>
        /// <param name="user"></param>
        /// <param name="destination"></param>
        /// <param name="forGroup"></param>
        public void UpdateStateMessages(string user, string destination, bool isforGroup)
        {
            this.Ctrl.UpdateStateMessages(user, destination, isforGroup);
        }
        /// <summary>
        /// permet de changer le mot de passe de l'utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="password">nouveau mot de passe de l'utilisateur</param>
        /// <returns>réussit ou annuler</returns>
        public bool ChangePassword(string user, string password)
        {
            return this.Ctrl.ChangePassword(user, password);
        }
    }
}
```

```
/*****************************************
* Projet : TalkEntreprise_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* Version : 1.0
* Auteur :Gabriel Strano
*
*****************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace talkEntreprise_server
{
    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;


        public bool ForGroup
        {
            get { return _forGroup; }
            set { _forGroup = value; }
        }
        ///////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; }
        }

        ////////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_server
* Description : création d'une messagerie instantanée
* Date : 15.06.2016
* 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_server
{
    public class Controler
    {
        /////Champs/////
        FrmConnection _frmLogin;
        private Server _serv;
        private RequestSQL _request;
        private Converter _conv;
        private User _admin;
        private Thread _frmProg;
        /////propriétées/////
        public FrmConnection FrmLogin
        {
            get { return _frmLogin; }
            set { _frmLogin = value; }
        }
        public Server Serv
        {
            get { return _serv; }
            set { _serv = value; }
        }

        public RequestSQL Request
        {
            get { return _request; }
            set { _request = value; }
        }
        public Converter Conv
        {
            get { return _conv; }
            set { _conv = value; }
        }
        public User Admin
        {
            get { return _admin; }
            set { _admin = value; }
        }
        public Thread FrmProg
        {
```

```
        get { return _frmProg; }
        set { _frmProg = value; }
    }
    /////Constructeur/////
    public Controler(FrmConnection frm)
    {
        this.FrmLogin = frm;
        this.Request = new RequestSQL(this);
        this.Serv = new Server(this);
        this.Conv = new Converter(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, User u)
    {
        this.Admin = u;
        //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, this.Admin);
        prog.FormClosed += (s, e) =>
        Dispatcher.CurrentDispatcher.BeginInvokeShutdown(DispatcherPriority.Background);
        prog.Show();
        //permet de garder la fenêtre ouverte
        Dispatcher.Run();
```

```
        }
        //////méthodes RequestSQL///
        /// <summary>
        /// Permet de vérifier si les informations entrées par l'administrateur sont valide
        ou non
        /// </summary>
        /// <param name="user">Identifiant de l'administrateur</param>
        /// <param name="password"> mot de passe de l'administrateurn</param>
        /// <returns>retourne "true" si l'administrateur à les bonnes informations de
        connection</returns>
        ///
        public Boolean ValidateConnectionAdmin(string user, string password)
        {
            return this.Request.ValidateConnectionAdmin(user, password);
        }
        /// <summary>
        /// Permet de vérifier si les informations entrées par l'utilisateur sont valide ou
        non
        /// </summary>
        /// <param name="user">Identifiant de l'utilisateur</param>
        /// <param name="password"> mot de passe de l'utilisateur</param>
        /// <returns>retourne "true" si l'utilisateur à les bonnes informations de
        connection</returns>
        ///
        public Boolean ValidateConnection(string id, string password)
        {
            return this.Request.ValidateConnectionUser(id, password);
        }
        /// <summary>
        /// permet de convertire un nombre en hexadécimal
        /// </summary>
        /// <param name="number">nombre à convertire</param>
        /// <returns>nombre en hexadécimal</returns>
        public string NumberToHexadecimal(int number)
        {
            return this.Conv.NumberToHexadecimal(number);
        }
        /// <summary>
        /// permet de dire que l'utilisateur c'est connecté sur le server --> log de la base
        de données
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        public void SucessConnectionToServer(string user)
        {
            this.Request.SucessConnectionToServer(user);
        }
        /// <summary>
        /// permet de dire que l'utilisateur c'est déconnecté du serveur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        public void DeconnectionToServer(string user)
        {
            this.Request.DeconnectionToServer(user);
        }
        /// <summary>
        /// permet de récupérer les informations de l'utilisateur demmandé
        /// </summary>
```

```
        /// <param name="user">identifiant de l'utilisateu</param>
        /// <returns>list des informations de l'utilisateur</returns>
        public List<string> GetInformation(string user)
        {
            return this.Request.GetInformation(user);
        }
        /// <summary>
        /// récupération de la liste des employés
        /// </summary>
        /// <param name="nameGroup">nom du groupe de l'utilisateur</param>
        /// <param name="idGroup">identifiant du groupe de l'utilisateur</param>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <returns>list des employés</returns>
        public List<User> GetUserList(string nameGroup, int idGroup, string user)
        {
            return this.Request.GetUserList(nameGroup, idGroup, user);
        }
        /// <summary>
        /// permet de récupérer les informations relatifs au messages non lu de l'utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="userDestination">identifiant d'un autre utilisateur</param>
        /// <param name="forGroup">si le message est envoyé au groupe</param>
        /// <returns>nombre de message eb abscence</returns>
        public string GetUserListInString(string nameGroup, int idGroup, string user)
        {
            return this.Request.GetUserListInString(nameGroup, idGroup, user);
        }
        /// <summary>
        /// permet  d'enregistrer le message dans la base de données
        /// </summary>
        /// <param name="message">message crypter</param>
        /// <param name="user">nom de l'utilisateur</param>
        /// <param name="destinationUsername">destinataire du message</param>
        /// <param name="forGroup">si c'est pour un groupe</param>
        public void SendMessage(string user, string destinationUsername, string message,
        bool forGroup)
        {
            this.Request.SendMessage(user, destinationUsername, message, forGroup);
        }
        /// <summary>
        /// permet de récupérer les messages envoyé par les utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire du messahe</param>
        /// <param name="forGroup">si c'est pour le groupe</param>
        /// <returns></returns>
        public List<Message> GetConversation(string user, string destination, bool forGroup)
        {
            return this.Request.GetConversation(user, destination, forGroup);
        }
        /// <summary>
        /// permet de récupérer les anciens messages envoyé par les utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="destination">destinataire du messahe</param>
        /// <param name="forGroup">si c'est pour le groupe</param>
```

```
        /// <returns></returns>
        public List<Message> GetOldConversation(string user, string destination, bool
        forGroup, int nbDays)
        {
            return this.Request.GetOldConversation(user, destination, forGroup, nbDays);
        }
        /// <summary>
        /// permet de mettre à jour l'état des messages
        /// </summary>
        /// <param name="user"></param>
        /// <param name="destination"></param>
        /// <param name="forGroup"></param>
        public void UpdateStateMessages(string user, string destination, bool isforGroup)
        {
            this.Request.UpdateStateMessages(user, destination, isforGroup);
        }
        /// <summary>
        /// permet de mettre à zéro tous les utilisateurs
        /// </summary>
        public void SetAllEmployeesDeconnected()
        {
            this.Request.SetAllEmployeesDeconnected();
        }

        /// <summary>
        /// permet de changer le mot de passe de l'utilisateur
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="password">nouveau mot de passe de l'utilisateur</param>
        /// <returns>réussit ou annuler</returns>
        public bool ChangePassword(string user, string password)
        {
            return this.Request.ChangePassword(user, password);
        }
        //////méthodes Server///////
        /// <summary>
        /// permet de quitter la connection
        /// </summary>
        ////Server////
        /// <summary>
        /// permet d'ajouter le nouveau client à la liste de client
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="tcp">connection au client</param>
        public void AddClientList(string user, TcpClient tcp)
        {
            this.Serv.AddClientList(user, tcp);
        }
        /// <summary>
        /// permet d'ajouter le nouveau processus à la liste de processus
        /// </summary>
        /// <param name="user">identifiant de l'utilisateur</param>
        /// <param name="t">processus de l'utilisateur</param>
        public void AddThreadList(string user, Thread t)
        {
            this.Serv.AddThreadList(user, t);
        }
```

```
        //méthode FrmConnection
        /// <summary>
        /// changer l'état visuel de la forme de connexion
        /// </summary>
        public void isVisible()
        {
            this.FrmLogin.IsVisible();
        }
    }
}
```