



Ingénierie & systèmes automatisés

## COMPTE RENDU STAGE

*Année 2023-2024*

### SEMAINE 2

09/01/2023 – 13/01/2023

FONCTION	NOM
Étudiant stagiaire	Allan ESCOLANO
Représentante entreprise	Grégory MEUNIER
Tuteur en entreprise	Jean-François DANCKAERT
Tuteur professionnel	Jean-François DANCKAERT
Représentante Lycée	Patricia BUËR
Professeur chargé du suivi	David ROUMANET

## **Sommaire / Summary**

# Table des matières

1.	Présentation de l'entreprise.....	3
2.	Mes missions.....	4
1.1	Simulateur_CEOG.....	4
1.1.1	Documents et maquette.....	4
1.1.2	Modifications graphiques.....	8
1.1.3	Modifications code .....	10
3.	Outils utilisés.....	33
4.	Conclusion.....	34
5.	Niko-Niko.....	35

# 1. Présentation de l'entreprise.

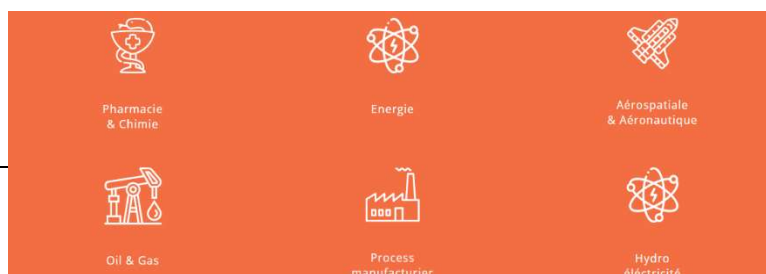


## **ICÔNE AUTOMATISATION**

**Société spécialisée dans la conception, l'intégration, la mise en service et la maintenance de système automatisés et informatisés pour les outils de production et les moyens d'essais.**

Créé en 1985, Icône Automation est spécialisée dans la conception, l'intégration, la mise en service et la maintenance de systèmes automatisés et informatisés pour les outils de production et les moyens d'essais. Icône Automation a développé depuis de nombreuses années son expertise dans la conception et la réalisation de contrôles commandes en milieu sensible.

Capable de prendre en compte les particularités des secteurs d'activités, Icone sait accompagner dans la réalisation des systèmes d'automatisation et d'informatisation de process industriels.



## 2. Mes missions.

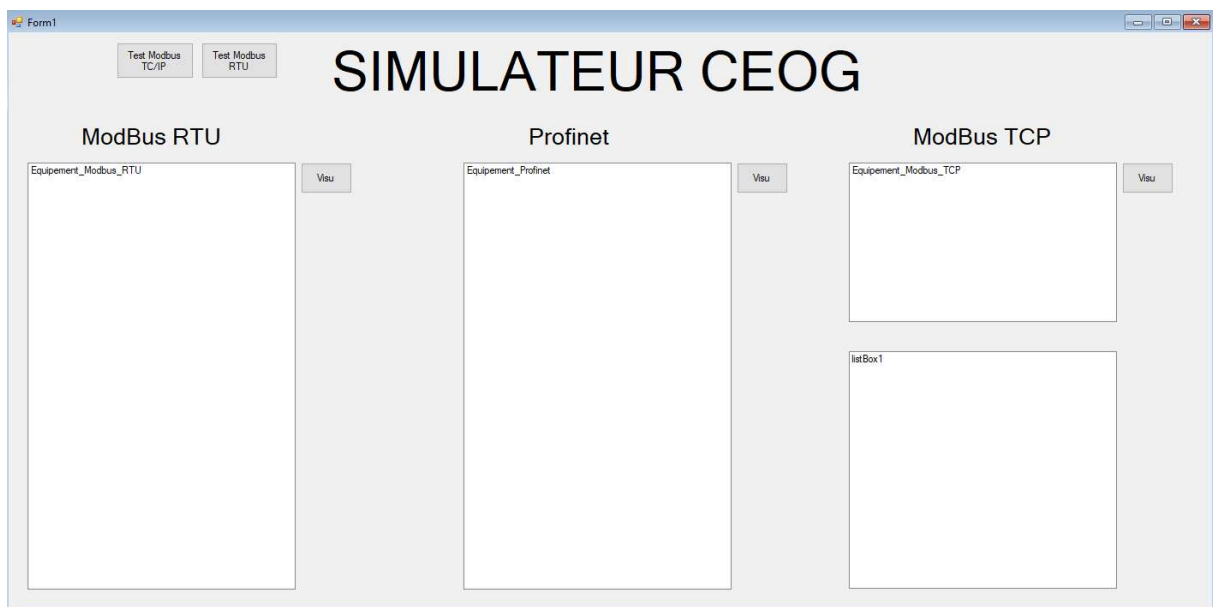
### 1.1 Simulateur CEOG.

#### 1.1.1 Documents et maquette.

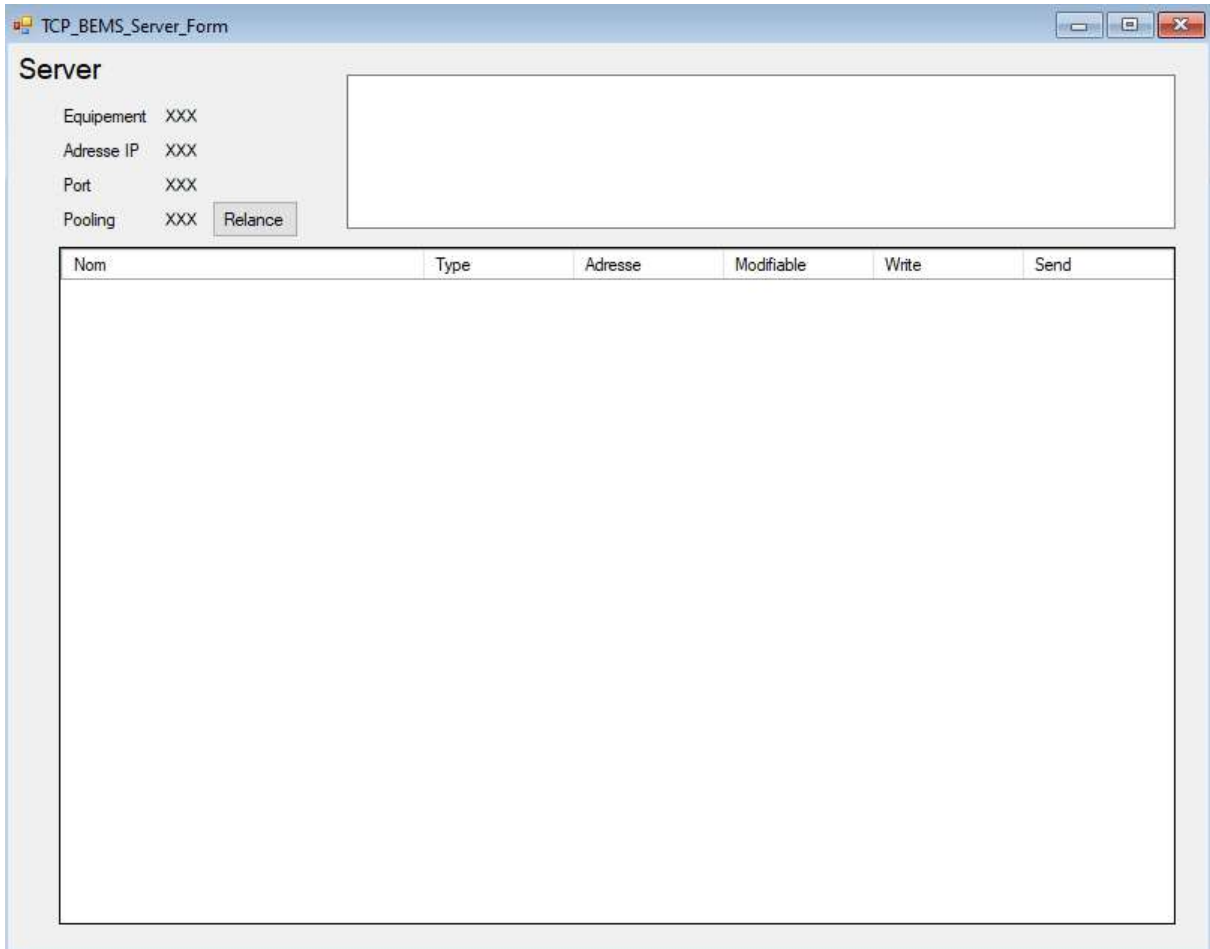
Pour cette semaine, un nouveau projet m'a été affecter.

Le projet se nomme Simlateur\_CEOG et à pour but de faire une application simulation des connexions entre des équipements contenant des protocoles Modbus.

J'ai été fournis pour cela un fichier de documentation du projet ainsi qu'une application C# maquette fait par mon tuteur :



Exemple d'interface d'un Modbus :



Nom	Type	Adresse	Modifiable	Write	Send
-----	------	---------	------------	-------	------

Avant de faire quoi que ce soit, j'ai pris le temps de me renseigner sur les Modbus car je ne connaissais que le nom.

J'y ai vite découvert des informations intéressantes qui seront utiles pour le projet tel que les différents protocoles (TCP/IP et RTU) et comment ils communiquent (Serveur/Client et Master/Slave).

Bien comprendre entièrement leurs fonctionnements m'a pris du temps et j'en ai appris même après avoir commencé la modification du projet.

Ensuite j'ai étudié la maquette et le code ainsi que la DLL (EasyModbus) donnée dans le projet.

Le projet consiste à utiliser des fichiers en .csv pour instancier des classes représentant les équipements à simuler ainsi que leurs interfaces.

Voici un exemple de fichier de configuration (Remplie et choisis par l'utilisateur) :

```




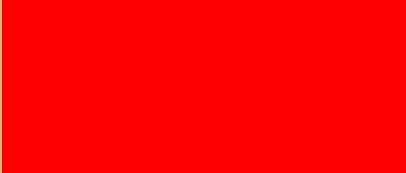




fichier de config.csv - Bloc-notes
Fichier Edition Format Affichage Aide
nom de l'instance;Machine;protocole;Mode;Configuration liaison;fichier echange(Equipement mode.csv)
MonServer1;BEMS;Modbus TCP;Server;172.18.66.81|502|99;ConfigEquipementsCSV\BEMS Server.csv
MonServer2;BEMS;Modbus TCP;Server;172.18.66.81|502|99;ConfigEquipementsCSV\BEMS Server.csv
MonClient1;BEMS;Modbus TCP;Client;172.18.66.81|502|100;ConfigEquipementsCSV\BEMS Client.csv
MonClient2;BEMS;Modbus TCP;Client;172.18.66.81|502|100;ConfigEquipementsCSV\BEMS Client.csv
    
```

Et voici un fichier de configuration d'un équipement (définie par le créateur de l'application) :

```

BEMS Client.csv - Bloc-notes
Fichier Edition Format Affichage Aide
Nom;Type;Adresse;Read/Write
U16 RW;U16;1;RW
U32 RW;U32;2;RW
I16 RW;I16;4;RW
I32 RW;I32;5;RW
float RW;Float;7;RW
U16 W;U16;9;W
U32 W;U32;10;W
I16 W;I16;12;W
I32 W;I32;13;W
float W;Float;15;W
U16 R;U16;17;R
U32 R;U32;18;R
I16 R;I16;20;R
I32 R;I32;21;R
float R;Float;23;R
    
```

Après avoir regarder le fonctionnement de l'application,  
J'y ai vite remarquer plusieurs problèmes :

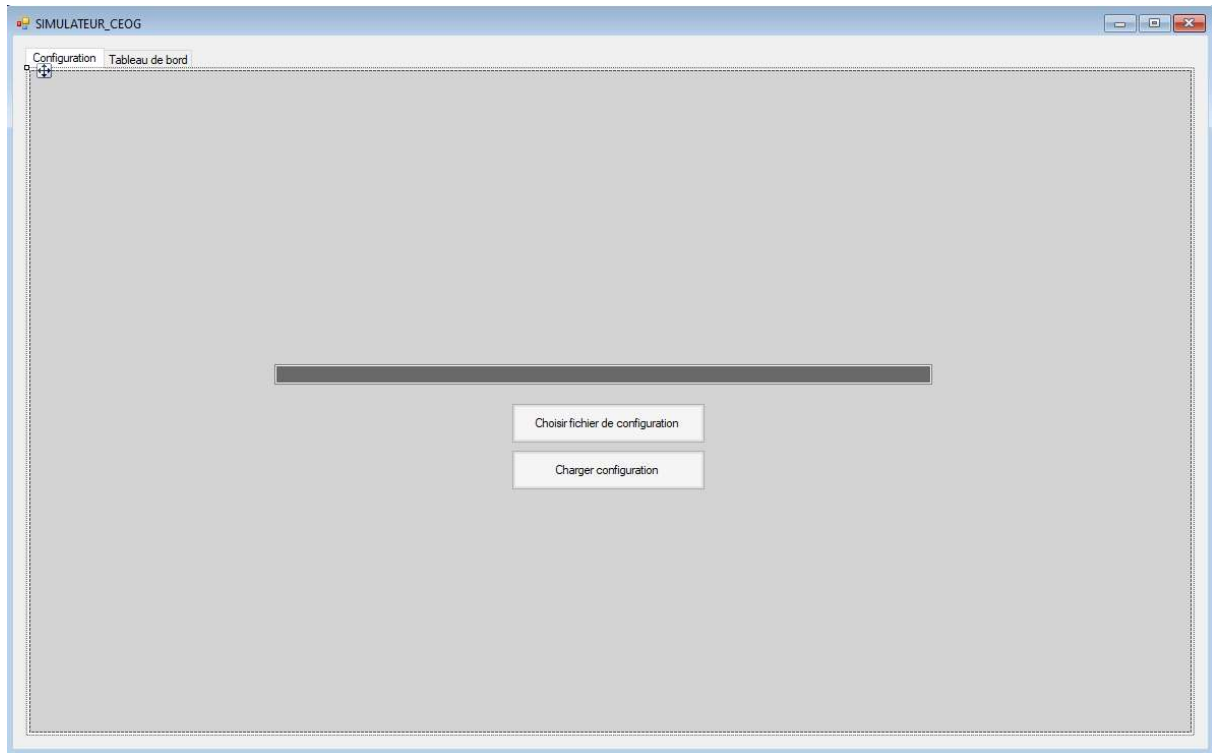
Nom du problème	Description	Gravité
		
<b>Interface non adapté</b>	L'interface n'est pas adaptée pour accueillir toutes les possibilités d'équipements	
<b>Aucun choix de connexions</b>	Le choix des connexions entre les équipements n'est pas disponible	
<b>Choix du fichier de config non disponible</b>	Le choix du fichier de configuration n'est pas présent (le fichier de config est codé en dur)	
<b>Manque d'options et de fonctionnalités</b>	Impossible de changer de fichier de config ou d'ouvrir les configurations des équipements	
<b>Grande Répétition de code</b>	Plusieurs centaines de lignes répétées ainsi qu'un code qui demandera beaucoup de copier-coller pour les différents équipements	
<b>Mauvaise utilisation de classes</b>	Classes pour les équipements vide et non utilisés	
<b>Code non fonctionnel</b>	Une partie du code ne fonctionne pas au lancement de l'application (+ 0 commentaires)	

## 1.1.2 Modifications graphiques.

Suite à ces problèmes, j'ai décidé (sous l'accord de mon tuteur) de reprendre le projet de 0 pour partir sur une base convenable.

Tout d'abord j'y ai apporté des modifications graphiques pour correspondre aux besoins.

Premièrement, une page d'accueil qui force l'utilisateur à choisir son fichier de configuration :







En partant de cette interface plus adaptée, j'ai pu partir sur une base solide pour commencer à coder.

### 1.1.3 Modifications code

Tout d'abord il me fallait une logique/un plan

J'ai décidé de faire les connexions de type Modbus TCP/IP et ai donc créé la classe suivante

Il me faut une classe qui contient :

- Les informations de l'équipement :

```
public class TCP_Class
{
    public string Nom;
    public string Machine;
    public string Protocol;
    public string Configuration_Liaison;
    public string Mode;
    public string Fichier_Echange;

    public string AdressIP;
    public int Port;
    public int Pooling;

    public ModbusClient Modbus_Client;
    public ModbusServer Modbus_Server;

    public TCP_Form Equipement_Form;

    public string ConnexionStatus = "Disconnected";

    public TCP_Class(string Nom, string Machine, string Protocol, string
Configuration_Liaison, string Mode, string Fichier_Echange)
    {
        this.Nom = Nom;
        this.Machine = Machine;
        this.Protocol = Protocol;
        this.Configuration_Liaison = Configuration_Liaison;
        this.Mode = Mode;
        this.Fichier_Echange = AppDomain.CurrentDomain.BaseDirectory +
Fichier_Echange;
        this.AdressIP = Configuration_Liaison.Split('|')[0];
        this.Port = Convert.ToInt32(Configuration_Liaison.Split('|')[1]);
        this.Pooling = Convert.ToInt32(Configuration_Liaison.Split('|')[2]);
    }
}
```

Une fois que nous avons une classe avec toute les informations nécessaires, nous pouvons commencer la lecture du fichier de configuration :

```
private void selectConfigFile(Object sender, EventArgs e)
{
    //permet la selection du fichier de configuration
    if (configFileSelector.ShowDialog() == DialogResult.OK) //Si le fichier
    chemin du fichier de config est bon
    {
        string pathName = configFileSelector.FileName;
        tb_ConfigFileDir.Text = pathName;
    } else
    {
        MessageBox.Show("Erreur lors de la récupération du fichier de
configuration");
    }
}

private void ReadConfigFile(Object sender, EventArgs e)
{
    string pathName = tb_ConfigFileDir.Text;

    try //Si la lecture du fichier de config se passe correctement
    {
        tb_ConfigFileDir.Text = pathName;
        StreamReader reader = new StreamReader(File.OpenRead(pathName));
        List<string[]> fileLines = new List<string[]>(); //liste contenant les
lignes du fichier

        while (!reader.EndOfStream) // tant que la lecture n'est pas au bout
du fichier
        {
            string line = reader.ReadLine();
            string[] values = line.Split(';');
            fileLines.Add(values);
        }
        fileLines.RemoveAt(0); // Enleve la premiere ligne du fichier (les en-
tetes)

        foreach (string[] line in fileLines) // pour chaque ligne du fichier
        {
            #region ALIMENTATION LISTE TCP D'EQUIPEMENTS EN TCP
            if (line[2] == "Modbus TCP")
            {
                //récupération des données et création de nos classes
                string Nom = line[0];
                string Machine = line[1];
                string Protocole = line[2];
                string Mode = line[3];
                string Configuration_Liaison = line[4];
                string Fichier_Echange = line[5];
                TCP_Class newTCP_Equipement = new TCP_Class(Nom, Machine,
Protocole, Configuration_Liaison, Mode, Fichier_Echange);
                listTCP_Equipement.Add(newTCP_Equipement);
            }
            #endregion
        }
    }
}
```

```

        //Changement de page vers le tableau de bord
        tc_Main.SelectedIndex = 1;
    }
    catch //Si erreur pendant la lecture du fichier de config
    {
        MessageBox.Show("Erreur lecture du fichier de configuration");
    }

    #region Ajout form dans les classes
    foreach (TCP_Class TCP in listTCP_Equipement)
    {
        TCP_Form Form = new TCP_Form(TCP);
        TCP.Equipement_Form = Form;
        TCP.Equipement_Form.set_tbStatusText("Disconnected");
        TCP.Equipement_Form.Show(); //Ceci permet de charger les formulaire
    }
    #endregion

    #region Ajout dans les ListBox
    foreach (TCP_Class TCPEquipement in listTCP_Equipement)
    {
        switch (TCPEquipement.Mode)
        {
            case "Client":
                lb_ModBus_TCP_Client.Items.Add(TCPEquipement.Nom);
                break;

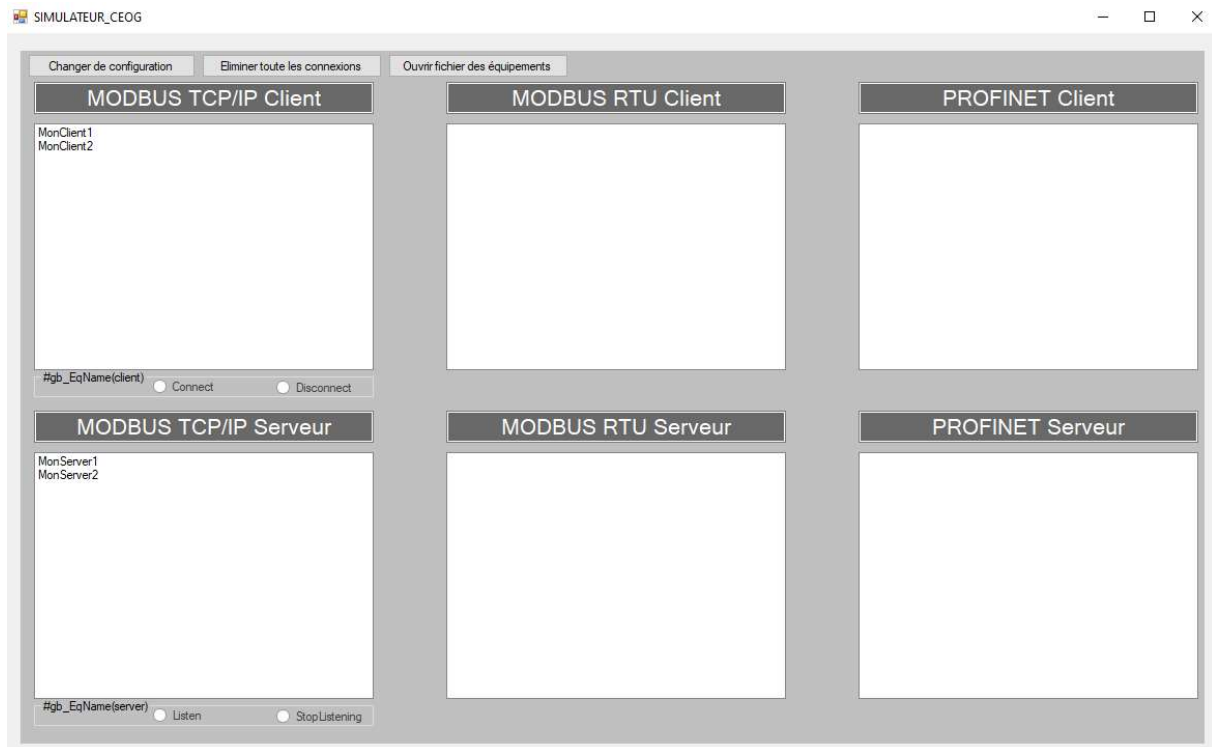
            case "Server":
                lb_ModBus_TCP_Serveur.Items.Add(TCPEquipement.Nom);
                break;
        }
    }
    #endregion

    #region Ajout des modbus Client/Serveur
    foreach (TCP_Class TCPclass in listTCP_Equipement) //Ajout des modbus de
    la DLL dans les classes
    {
        switch (TCPclass.Mode)
        {
            case "Client":
                ModbusClient client = new ModbusClient();
                client.IPAddress = TCPclass.AdressIP;
                client.Port = TCPclass.Port;
                TCPclass.Modbus_Client = client;
                break;

            case "Server":
                ModbusServer server = new ModbusServer();
                IPHostEntry host = Dns.GetHostEntry(Dns.GetHostName());
                server.LocalIPAddress = host.AddressList[3];
                server.Port = TCPclass.Port;
                TCPclass.Modbus_Server = server;
                break;
        }
    }
    #endregion
}

```

Le code précédent permet l'instanciation des classes avec leurs informations ainsi que l'affichage graphique :



Ensuite il fallait coder la possibilité de connecté les équipements entre eux grâce au radio buttons.

Voici la logique que j'ai utilisé :

Dés qu'un Item de la ListBox est choisi

Si le status de l'équipements est connecté, décocher les 2 radios buttons puis Enable le bouton « Disconnect »

Si le status est déconnecté, décocher les 2 radios buttons puis Enable le bouton « Connect »

Il faut aussi changer les boutons quand l'appareil se connecte :

Si un radiobuttons se coche,

Disable les 2 radiosButtons

Connecter l'équipement

## Changer le statut de connexion de l'équipement

Si le statut de l'équipement est connecté, décocher les 2 radios buttons puis Enable le bouton « Disconnect »

Si le statut est déconnecté, décocher les 2 radios buttons puis Enable le bouton « Connect »

Si nous suivons cette logique, nous pouvons connecter nos appareils entre eux de manière libre.

Voici le code qui permet ceci :

```
private void Lb_ModBus_TCP_SelectedIndexChanged(object sender, EventArgs e)
//Quand l'élément choisie de la liste change
{
    #region affichage radiobuttons TCP client quand changement de selection
d'item
    if (sender == lb_ModBus_TCP_Client)
    {
        try
        {
            string equipementName =
lb_ModBus_TCP_Client.SelectedItem.ToString();
            foreach (TCP_Class TCPEquipement in listTCP_Equipement)
            {
                if (TCPEquipement.Nom == equipementName) //recupere la bonne
classe
            {
                gb_TCP_Client_Connexion.Text = TCPEquipement.Nom;
                switch (TCPEquipement.ConnexionStatus)
                {
                    case "Connected": // si l'élément est connecter
                        rb_TCP_Client_Connect.Checked = false;
                        rb_TCP_Client_Disconnect.Checked = false;

                        rb_TCP_Client_Connect.Enabled = false;
                        rb_TCP_Client_Disconnect.Enabled = true;
                        break;

                    case "Disconnected": // si l'élément est deconnecter
                        rb_TCP_Client_Connect.Checked = false;
                        rb_TCP_Client_Disconnect.Checked = false;

                        rb_TCP_Client_Connect.Enabled = true;
                        rb_TCP_Client_Disconnect.Enabled = false;
                        break;
                }
            }
        }
    }
}
catch
{
}
```

```

    }
}
#endregion

#region affichage radiobuttons TCP serveur quand changement de selection
d'item
if (sender == lb_ModBus_TCP_Serveur)
{
    try
    {
        string equipementName =
lb_ModBus_TCP_Serveur.SelectedItem.ToString();
        foreach (TCP_Class TCPequipement in listTCP_Equipement)
        {
            if (TCPequipement.Nom == equipementName)
            {
                gb_TCP_Serveur_Connexion.Text = TCPequipement.Nom;
                switch (TCPequipement.ConnexionStatus)
                {
                    case "Listening":
                        rb_TCP_Serveur_Connect.Checked = false;
                        rb_TCP_Serveur_Disconnect.Checked = false;

                        rb_TCP_Serveur_Connect.Enabled = false;
                        rb_TCP_Serveur_Disconnect.Enabled = true;

                        break;

                    case "Disconnected":
                        rb_TCP_Serveur_Connect.Checked = false;
                        rb_TCP_Serveur_Disconnect.Checked = false;

                        rb_TCP_Serveur_Connect.Enabled = true;
                        rb_TCP_Serveur_Disconnect.Enabled = false;
                        break;
                }
            }
        }
    }
    catch
    {
    }
}
#endregion
}

private void tcpRadioButtons_CheckedChanged(Object sender, EventArgs e)
{
    /*Cette fonction se trouve dans 2 radiobuttons donc pour eviter qu'elle se
    déclanche deux fois (elle se déclanche a chaque changement sur 1 radiobutton)
    C'est pour cela que les radio button ne sont jamais selectioner par le
    code, explication :
        si nous changons de client sélectionner, il faut que les radios button
    change en fonction de son status de connexion
        seulement, si nous changons directement le radiobutton selectioné, cette
    fonction est appelé et va tenter une connexion non voulu
        vu que nous voulons un changement purement graphique, j'utilise l'option
    Enabled des radiobutton qui marche bien pour ce cas
    */
}

```

```

        if (((RadioButton)sender).Checked) // pour ne pas que la fonction se lance
        2 fois (1 seule bouton est .Checked donc elle se lance une fois)
        {
            #region Si RadioButton TCP Client changé
            if (sender == rb_TCP_Client_Connect) //Regarde quel radiobutton appel
            l'event
            {
                string equipementName =
                lb_ModBus_TCP_Client.SelectedItem.ToString();
                foreach (TCP_Class TCPEquipement in listTCP_Equipement)
                {
                    if (equipementName == TCPEquipement.Nom)
                    {
                        connectTCPEquipement(TCPEquipement); //Connecte
                        l'equipement grace a la fonction écrite plus haut
                    }
                }
            }
            if (sender == rb_TCP_Client_Disconnect)
            {
                string equipementName =
                lb_ModBus_TCP_Client.SelectedItem.ToString();
                foreach (TCP_Class TCPEquipement in listTCP_Equipement)
                {
                    if (equipementName == TCPEquipement.Nom)
                    {
                        disconnectTCPEquipement(TCPEquipement);
                    }
                }
            }
        }
        #endregion

        #region Si RadioButton TCP Serveur changé
        if (sender == rb_TCP_Serveur_Connect)
        {
            string equipementName =
            lb_ModBus_TCP_Serveur.SelectedItem.ToString();
            foreach (TCP_Class TCPEquipement in listTCP_Equipement)
            {
                if (equipementName == TCPEquipement.Nom)
                {
                    connectTCPEquipement(TCPEquipement);
                }
            }
        }
        if (sender == rb_TCP_Serveur_Disconnect)
        {
            string equipementName =
            lb_ModBus_TCP_Serveur.SelectedItem.ToString();
            foreach (TCP_Class TCPEquipement in listTCP_Equipement)
            {
                if (equipementName == TCPEquipement.Nom)
                {
                    disconnectTCPEquipement(TCPEquipement);
                }
            }
        }
    }
    #endregion
}
}

```



```
private void connectTCPEquipement(TCP_Class TCPclass)
{
    switch (TCPclass.Mode)
    {
        case "Client": //Connexion client
            try
            {
                if (TCPclass.Modbus_Client.Connected == false) //si le client
n'est pas connecté
                {
                    TCPclass.Equipement_Form.logMessage("Trying connexion at :
IP->" + TCPclass.AdressIP + " / Port->" + TCPclass.Port.ToString());
                    TCPclass.Modbus_Client.Connect();
                    TCPclass.ConnexionStatus = "Connected";
                    TCPclass.Equipement_Form.set_tbStatusText("Connected");
                    TCPclass.Equipement_Form.logMessage("Connexion established
at : IP->" + TCPclass.AdressIP + " / Port->" + TCPclass.Port.ToString());

                    //Ceci permet de disable les radiobutton en fonction de la
connexion (Explication dans la fonction des radiobutton)
                    rb_TCP_Client_Connect.Checked = false;
                    rb_TCP_Client_Disconnect.Checked = false;

                    rb_TCP_Client_Connect.Enabled = false;
                    rb_TCP_Client_Disconnect.Enabled = true;
                }
            }
            catch (Exception e)
            {
                TCPclass.Equipement_Form.logMessage("Connexion failed at : IP-
>" + TCPclass.AdressIP + " / Port->" + TCPclass.Port.ToString());
                rb_TCP_Client_Connect.Checked = false;
                rb_TCP_Client_Disconnect.Checked = false;

                rb_TCP_Client_Connect.Enabled = true;
                rb_TCP_Client_Disconnect.Enabled = false;
                MessageBox.Show("Erreur connexion Client (verifier qu'un
serveur écoute et que les adresse IP sont bien attribuées : " + TCPclass.Nom + " | " +
e.Message);
            }
            break;

        case "Server": //Ecoute serveur
            try
            {
                TCPclass.Equipement_Form.logMessage("Trying listen on : IP->"
+ TCPclass.AdressIP + " / Port->" + TCPclass.Port.ToString());
                TCPclass.Modbus_Server.Listen();
                TCPclass.Modbus_Server.HoldingRegistersChanged += new
ModbusServer.HoldingRegistersChangedHandler(TCPclass.Equipement_Form.registersChanged)
;
                TCPclass.Equipement_Form.logMessage("Listen successful on :
IP." + TCPclass.AdressIP + " Port." + TCPclass.Port.ToString());
                TCPclass.ConnexionStatus = "Listening";
                TCPclass.Equipement_Form.set_tbStatusText("Listening");
                rb_TCP_Serveur_Connect.Checked = false;
                rb_TCP_Serveur_Disconnect.Checked = false;

                rb_TCP_Serveur_Connect.Enabled = false;
                rb_TCP_Serveur_Disconnect.Enabled = true;
            }
            catch (Exception e)
            {
                TCPclass.Equipement_Form.logMessage("Listen failed on : IP->"
+ TCPclass.AdressIP + " / Port->" + TCPclass.Port.ToString());
                rb_TCP_Serveur_Connect.Checked = false;
                rb_TCP_Serveur_Disconnect.Checked = false;

                rb_TCP_Serveur_Connect.Enabled = true;
                rb_TCP_Serveur_Disconnect.Enabled = false;
            }
            break;
    }
}
```

```

    }
    catch (Exception e)
    {
        TCPclass.Equipement_Form.logMessage("Listen failed on : IP->"
+ TCPclass.AdressIP + " / Port->" + TCPclass.Port.ToString());
        rb_TCP_Serveur_Connect.Checked = false;
        rb_TCP_Serveur_Disconnect.Checked = false;

        rb_TCP_Serveur_Connect.Enabled = true;
        rb_TCP_Serveur_Disconnect.Enabled = false;
        MessageBox.Show("Erreur écoute Serveur : " + TCPclass.Nom + "
| " + e.Message);
    }
    break;
}

private void disconnectTCPEquipement(TCP_Class TCPclass)
{
    switch (TCPclass.Mode)
    {
        case "Client": //Déconnexion client
            try
            {
                if (TCPclass.Modbus_Client.Connected)
                {
                    TCPclass.Modbus_Client.Disconnect();
                    TCPclass.Equipement_Form.logMessage("Client successfully
disconnected from : IP->" + TCPclass.AdressIP + " / Port->" +
TCPclass.Port.ToString());
                    TCPclass.ConnexionStatus = "Disconnected";
                    TCPclass.Equipement_Form.set_tbStatusText("Disconnected");
                    rb_TCP_Client_Connect.Checked = false;
                    rb_TCP_Client_Disconnect.Checked = false;

                    rb_TCP_Client_Connect.Enabled = true;
                    rb_TCP_Client_Disconnect.Enabled = false;
                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Erreur déconnexion Client : " + TCPclass.Nom
+ " | " + e.Message);
            }
            break;

        case "Server": // Arrêt écoute serveur
            try
            {
                TCPclass.Modbus_Server.StopListening();
                TCPclass.Equipement_Form.logMessage("Server successfully
stopped listening on : IP->" + TCPclass.AdressIP + " / Port->" +
TCPclass.Port.ToString());
                TCPclass.ConnexionStatus = "Disconnected";
                TCPclass.Equipement_Form.set_tbStatusText("Disconnected");
                rb_TCP_Serveur_Connect.Checked = false;
                rb_TCP_Serveur_Disconnect.Checked = false;

                rb_TCP_Serveur_Connect.Enabled = true;
                rb_TCP_Serveur_Disconnect.Enabled = false;
            }
            catch (Exception e)
            {
                MessageBox.Show("Erreur arrêt écoute serveur : " + TCPclass.Nom
+ " | " + e.Message);
            }
            break;
    }
}

```

```

    }
    catch (Exception e)
    {
        //MessageBox.Show("Erreur stop écoute Serveur : " +
TCPclass.Nom + " | " + e.Message);
    }
    break;
}
}
}

```

Ensuite il faut pouvoir ouvrir l'interface des équipements :

```

private void Lb_ModBus_TCP_DoubleClick(object sender, EventArgs e)
{
    //affiche le form de la classe sélectionné

    #region formulaire TCP client
    if (sender == lb_ModBus_TCP_Client)
    {
        try
        {
            string selectedEquipmentName =
lb_ModBus_TCP_Client.SelectedItem.ToString();
            foreach (TCP_Class TCPclass in listTCP_Equipement)
            {
                if (selectedEquipmentName == TCPclass.Nom) //récupération de
la bonne classe
                {
                    TCPclass.Equipement_Form.Show();
                }
            }
        }
        catch // si aucun item sélectionné, ne rien faire
        {
        }
    }
}

#endregion

#region formulaire TCP serveur
if (sender == lb_ModBus_TCP_Serveur)
{
    try
    {
        string selectedEquipmentName =
lb_ModBus_TCP_Serveur.SelectedItem.ToString();
        foreach (TCP_Class TCPclass in listTCP_Equipement)
        {
            if (selectedEquipmentName == TCPclass.Nom)
            {
                TCPclass.Equipement_Form.Show();
            }
        }
    }
}
}

```

```

        catch
        {
        }
    }
}
#endregion
}

```

Après cela est venu la partie la plus compliqué, c'est faire marcher et communiquer les équipements et leurs interfaces.

Voici les fonctions graphique de chaque interface ainsi que quelques fonctions moins importantes :

```

string Mode;
public string equipementName;
string AdressIP;
int Port;
int Pooling;
string Fichier_echange;
public DateTime Old_date = DateTime.Now;
public DateTime New_date = new DateTime();
public int AdresseMax = 0;
TCP_Class TCPEquipement;

public TCP_Form(TCP_Class TCPEquipement) //Constructeur
{
    this.Mode = TCPEquipement.Mode;
    this.equipementName = TCPEquipement.Nom;
    this.AdressIP = TCPEquipement.AdressIP;
    this.Port = TCPEquipement.Port;
    this.Pooling = TCPEquipement.Pooling;
    this.Fichier_echange = TCPEquipement.Fichier_Echange;
    this.TCPEquipement = TCPEquipement;

    InitializeComponent();

    logMessage("Initialisation complete.");

    StreamReader streamReader = File.OpenText(this.Fichier_echange);
    string headerLine = streamReader.ReadLine();
    int count = 0;
    while (!streamReader.EndOfStream)
    {
        string line = streamReader.ReadLine();
        string[] values = line.Split(';');
        string Nom = values[0];
        string Type = values[1];
        string Adresse = values[2];
        string ReadWrite = values[3];
        dgv_dataHolder.Rows.Add(Nom, 0, Type, Adresse, true);
    }
}

```

```

//MessageBox.Show(Nom + " / " + Type + " / " + Adresse + " / " +
ReadWrite);
switch (ReadWrite)
{
    case "R":
        dgv_dataHolder.Rows[count].Cells["Write"].ReadOnly = true;
        dgv_dataHolder.Rows[count].Cells["Write"].Style.BackColor =
Color.FromArgb(255, 110, 50, 50);
        dgv_dataHolder.Rows[count].Cells["Write"].Style.ForeColor =
Color.White;
        dgv_dataHolder.Rows[count].Cells["Write"].Value = "Lecture
seule pour | " + Nom;
        break;

    case "W":
        dgv_dataHolder.Rows[count].Cells["Valeur"].Value = "Ecriture
seule pour | " + Nom;
        dgv_dataHolder.Rows[count].Cells["Valeur"].Style.BackColor =
Color.FromArgb(255, 110, 50, 50);
        dgv_dataHolder.Rows[count].Cells["Valeur"].Style.ForeColor =
Color.White;
        break;
}
count += 1;
}
AdresseMax = Convert.ToInt32(dgv_dataHolder.Rows[count -
1].Cells["Adresse"].Value);
}

private void BEMS_TCP_Form_Load(object sender, EventArgs e)
{
    lb_Equipement_Mode.Text = Mode;
    lb_Equipement_Name.Text = equipementName;
    lb_Equipement_IP.Text = AdressIP;
    lb_Equipement_Port.Text = Port.ToString();
    lb_Equipement_Pooling.Text = Pooling.ToString();
    this.Text = this.equipementName;
    timer1.Interval = this.Pooling;

    foreach(DataGridViewRow row in dgv_dataHolder.Rows)
    {
        if (Convert.ToInt32(row.Cells[3].Value) > AdresseMax)
        {
            AdresseMax = Convert.ToInt32(row.Cells[3].Value);
        }
    }
}

public void set_tbStatusText(string text)
{
    tb_Status.Text = text;
}

public void logMessage(string msg) // Permet de faire un message de log
{
    string now = DateTime.Now.ToString();
    string line= now + " ---> " + msg + "\n";
    string historique = tb_Logs.Text;
    tb_Logs.Text = line + "\r\n" + historique;
}

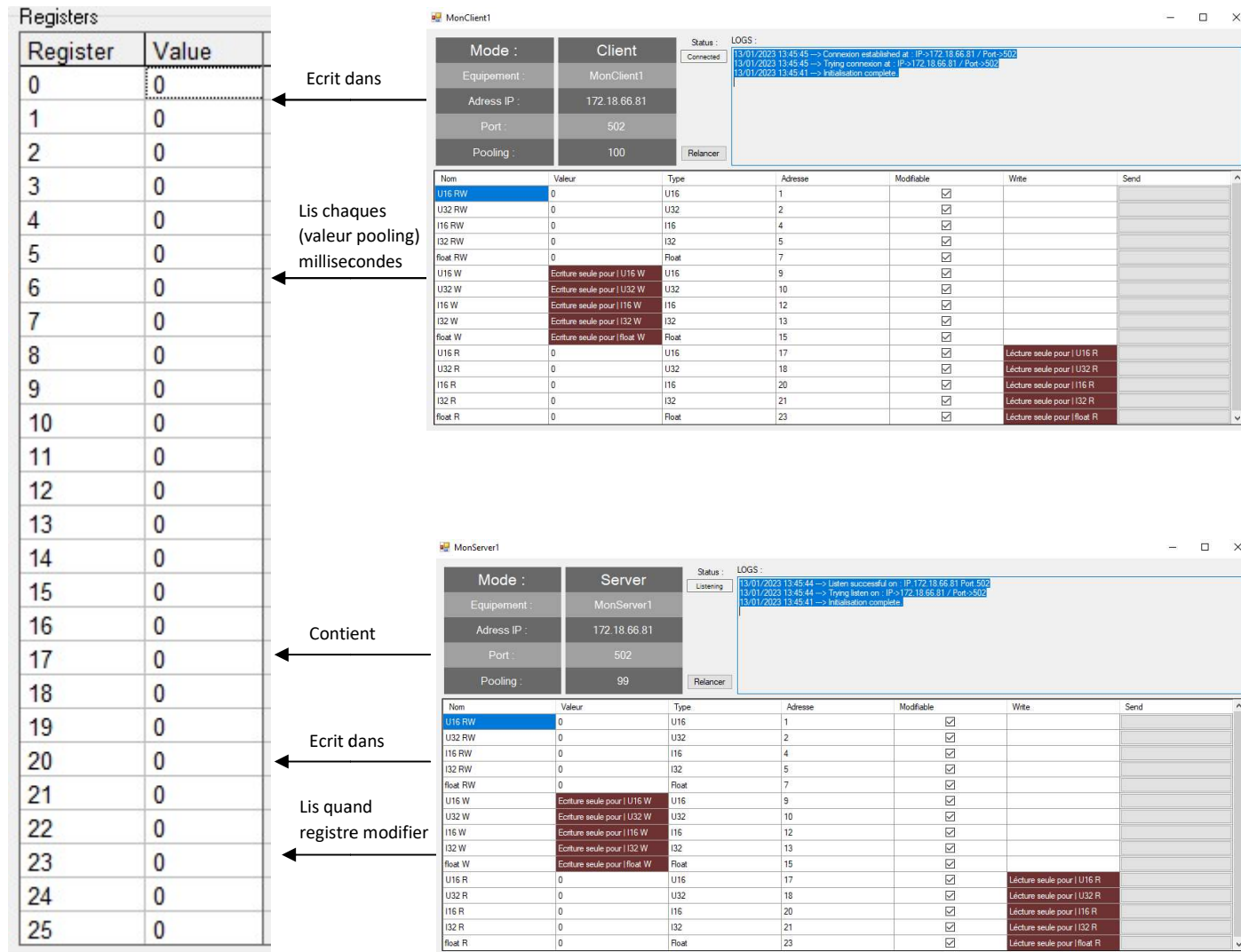
```

```
private void BEMS_TCP_Form_FormClosing(object sender, FormClosingEventArgs e)
//empêche le form de se supprimer mais juste de se cacher
{
    Hide();
    e.Cancel = true;
}
```

Ensuite est venu la partie la plus compliqué, l'écriture entre Serveur/Client

Pour cela il faut comprendre le fonctionnement.

Voici un schéma représentant le fonctionnement attendu :



Choses à savoir :

- Chaque registre de la table est un Int16 donc 2 Bytes par adresse
- Les registres de vrai Modbus TCP/IP Serveur sont des Int16 qui peuvent accepter les UInt16 (Int16 Min/Max = -32 768 à 32 767 et UInt16 Min/Max = 0 à 65 535)
- Les UInt32/Int32 ou Int prennent 2 registres ou 4 Bytes
- Les floats prennent 2 registres ou 4 Bytes
- Les long ou Int64 m'on pas été demander d'être codé
- Les UInt16/Int16 prennent 1 registre ou 2 Bytes

Avec ces informations, la transmission des valeurs semble simple sauf pour les UInt16/UInt32.

Car un UInt16 prend 1 registre ou 2 Bytes et à pour Min/Max->0/65535  
Alors qu'un registre (Int16) prend 2 Bytes et à pour Min/Max->-32768/32767

Nous devons manipuler les valeurs pour plusieurs cas possible :

Le UInt16 est entre [0 ; 32767], on peut envoyer la valeur car elle rentre dans un Int16

Le UInt16 est entre ]32767 ; 65535], on ne peut envoyer la valeur dans un Int16, j'ai donc manipulé la valeur :

Prenons comme exemple le maximum 65535.

On fait la valeur(65535) – (MaxInt16 + 1)

$65535 - (65535 + 1) = -1$

On met donc -1 dans un int16 et on l'envoie au serveur

Le serveur reçoit -1 et fait :

Si la valeur < 0 alors

Valeur = 65535 + (valeur(-1) + 1)

Valeur = 65535 + (-1 + 1)

Valeur = 65535

On retrouve donc notre valeur de base



Maintenant pour expliquer pourquoi je fais le maximum de la donnée + 1  
Si la valeur = 0, quand le serveur reçoit la valeur, celle-ci ne sera jamais plus petite que 0 donc le calcul ne sera jamais effectué et la valeur restera à 0.

Après cette longue explication, voici le code d'écriture entre les Modbus  
(Chaque fonction se ressemble mais est différente et ne peut être convertie en une seule et même fonction) :

### Fonction d'écriture client/serveur (non terminée pour envoi serveur)

```
//Les 3 fonctions qui suivent qui sont un peu collées sont les fonctions d'envois, de
réceptions et de conversion des valeurs
//1 . le client envoie la valeur (Si serveur envoie la valeur, il écrit juste
dans ses registres et passe au 3.)
//2 . Le serveur la reçoit et reconvertit la valeur
//3 . le client récupère les valeurs du serveur et les reconvertit
private void sendValueHandler(object sender, DataGridViewCellEventArgs e)
{
    if (e.ColumnIndex == 6 &&
dgv_dataHolder.CurrentRow.Cells[5].Style.BackColor != Color.FromArgb(255, 110, 50,
50)) //Si c'est le bouton qui est cliqué
    {
        int adresse =
Convert.ToInt32(dgv_dataHolder.CurrentRow.Cells[3].Value); //l'adresse
        int value = 0; //la valeur int du datagridview (16/32)
        int[] valueTwoRegisters = new int[2]; //conteneur de 2 registre (pour
tout /32 ou float)
        Single valueFloat = 0; //la valeur float du datagridview
        try
        {
            switch (TCPEquipement.Mode)
            {
                case "Client": //Si c'est un client
                    switch
(dgv_dataHolder.CurrentRow.Cells[2].Value.ToString())
                    {
                        case "U16":
                            value =
Convert.ToUInt16(dgv_dataHolder.CurrentRow.Cells[5].Value);
                            Int16 valInt16 = 0;
                            if (value > 32767)
                            {
                                valInt16 = Convert.ToInt16(value - 65536);
                            } else
                            {
                                valInt16 = Convert.ToInt16(value);
                            }
                        }
                    }
                    TCPEquipement.Modbus_Client.WriteSingleRegister(adresse, valInt16);
                    logMessage("Envoi de la valeur : " +
value.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
                    break;
                    case "U32":
```

```

        UInt32 val =
Convert.ToInt32(dgv_dataHolder.CurrentRow.Cells[5].Value);
        if (val > 2147483647)
        {
            value = Convert.ToInt32(val - 4294967296L);
//Le L spécifie qu'il s'agit d'un chiffre de type long (car sinon erreur)
        } else
        {
            value = Convert.ToInt32(val);
        }
        valueTwoRegisters =
EasyModbus.ModbusClient.ConvertIntToRegisters(value);

TCPEquipement.Modbus_Client.WriteMultipleRegisters(adresse, valueTwoRegisters);
        logMessage("Envoie de la valeur : " +
val.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
        break;

        case "I16": //Aucun besoin de conversion pour de l'I16
            value =
Convert.ToInt16(dgv_dataHolder.CurrentRow.Cells[5].Value); //la valeur du datagridview

TCPEquipement.Modbus_Client.WriteSingleRegister(adresse, value);
            logMessage("Envoie de la valeur : " +
value.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
            break;

        case "I32":
            value =
Convert.ToInt32(dgv_dataHolder.CurrentRow.Cells[5].Value);
            valueTwoRegisters =
EasyModbus.ModbusClient.ConvertIntToRegisters(value);

TCPEquipement.Modbus_Client.WriteMultipleRegisters(adresse, valueTwoRegisters);
            logMessage("Envoie de la valeur : " +
value.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
            break;

        case "Float":
            valueFloat =
Convert.ToSingle(dgv_dataHolder.CurrentRow.Cells[5].Value);
            valueTwoRegisters =
EasyModbus.ModbusClient.ConvertFloatToRegisters(valueFloat);

TCPEquipement.Modbus_Client.WriteMultipleRegisters(adresse, valueTwoRegisters);
            logMessage("Envoie de la valeur : " +
valueFloat.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
            break;
        }
        break;

        case "Server": //Si c'est un Serveur
            switch
(dgv_dataHolder.CurrentRow.Cells[2].Value.ToString()) //Important ! Pour X raisons,
l'adresse doit 1 fois plus haute que l'adresse d'envoi (raison du +1/+2)

```

```

{
    case "U16": // Non fonctionnel
        /*value =
Convert.ToUInt16(dgv_dataHolder.CurrentRow.Cells[5].Value);
        valueTwoRegisters =
EasyModbus.ModbusClient.ConvertIntToRegisters(value);

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 1] =
(short)valueTwoRegisters[0];

*/TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 2] =
(short)valueTwoRegisters[1];
        logMessage("Envoie de la valeur : " +
value.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
        break;
    case "U32": // Non fonctionnel
        UInt32 val2 =
Convert.ToUInt32(dgv_dataHolder.CurrentRow.Cells[5].Value);
        /*valueFourRegisters =
EasyModbus.ModbusClient.ConvertLongToRegisters(val2);

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 1] =
(short)valueFourRegisters[0];

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 2] =
(short)valueFourRegisters[1];

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 3] =
(short)valueFourRegisters[2];

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 4] =
(short)valueFourRegisters[3];

        */logMessage("Envoie de la valeur : " +
val2.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
        break;

    case "I16":
        Int16 val =
Convert.ToInt16(dgv_dataHolder.CurrentRow.Cells[5].Value); //la valeur du datagridview

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 1] = val;
        logMessage("Envoie de la valeur : " +
val.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
        break;

    case "I32":
        value =
Convert.ToInt32(dgv_dataHolder.CurrentRow.Cells[5].Value);
        valueTwoRegisters =
EasyModbus.ModbusClient.ConvertIntToRegisters(value);

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 1] =
(short)valueTwoRegisters[0];

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 2] =
(short)valueTwoRegisters[1];

```

```
                logMessage("Envoie de la valeur : " +
value.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
                break;

                case "Float":
                    valueFloat =
Convert.ToSingle(dgv_dataHolder.CurrentRow.Cells[5].Value);
                    valueTwoRegisters =
EasyModbus.ModbusClient.ConvertFloatToRegisters(valueFloat);

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 1] =
(short)valueTwoRegisters[0];

TCPEquipement.Modbus_Server.holdingRegisters.localArray[adresse + 2] =
(short)valueTwoRegisters[1];
                    logMessage("Envoie de la valeur : " +
valueFloat.ToString() + " | à l'adresse : " + adresse.ToString() + " | de type : " +
dgv_dataHolder.CurrentRow.Cells[2].Value.ToString());
                    break;
                }
            }
        } catch (Exception err)
        {
            logMessage("Erreur lors de l'envoi : " + err.Message);
        }
    }
}
```

## Fonction de lecture du serveur quand registre modifier

```
public void registersChanged(int register, int numberOfRegisters)
{
    try
    {
        int[] valueTwoRegisters = new int[2];
        int value = 0;
        Single valueFloat = 0;

        valueTwoRegisters[0] =
TCPEquipement.Modbus_Server.holdingRegisters.localArray[register];
        valueTwoRegisters[1] =
TCPEquipement.Modbus_Server.holdingRegisters.localArray[register+1];
        foreach (DataGridViewRow row in dgv_dataHolder.Rows)
        {
            if (row.Cells[3].Value.ToString() == (register - 1).ToString() &&
row.Cells[1].Style.BackColor != Color.FromArgb(255, 110, 50, 50)) //Car le register
est l'adresse + 1 donc il faut enlever 1
            {
                switch (row.Cells[2].Value.ToString())
                {
                    case "U16":
                        value =
TCPEquipement.Modbus_Server.holdingRegisters.localArray[register];
                        if (value < 0)
                        {
                            value = 65535 + (value + 1);
                        }
                        value = Convert.ToUInt16(value);
                        row.Cells[1].Value = value;

                        tb_Logs.Invoke(new MethodInvoker(delegate
                        {
                            logMessage("Ecriture reçu à l'adresse : " +
(register - 1).ToString() + " | de valeur : " + value.ToString() + " | et de type : "
+ row.Cells[2].Value.ToString());
                        }));
                        break;
                    case "U32":
                        Int32 val =
EasyModbus.ModbusClient.ConvertRegistersToInt(valueTwoRegisters);
                        UInt32 valUInt32 = 0;
                        if (val < 0)
                        {
                            valUInt32 = Convert.ToUInt32(4294967295L + (val +
1));
                        } else
                        {
                            valUInt32 = Convert.ToUInt32(val);
                        }
                        row.Cells[1].Value = valUInt32;

                        tb_Logs.Invoke(new MethodInvoker(delegate
                        {
                            logMessage("Ecriture reçu à l'adresse : " +
(register - 1).ToString() + " | de valeur : " + valUInt32.ToString() + " | et de type
: " + row.Cells[2].Value.ToString());
                        }));
                    }
                }
            }
        }
    }
}
```

```

        break;

        case "I16":
            value =
TCPEquipement.Modbus_Server.holdingRegisters.localArray[register];
            value = Convert.ToInt16(valueTwoRegisters[0]);
            row.Cells[1].Value = value;

            tb_Logs.Invoke(new MethodInvoker(delegate
            {
                logMessage("Ecriture reçu à l'adresse : " +
(register - 1).ToString() + " | de valeur : " + value.ToString() + " | et de type : "
+ row.Cells[2].Value.ToString());
            }));
            break;

        case "I32":
            value =
EasyModbus.ModbusClient.ConvertRegistersToInt(valueTwoRegisters);
            row.Cells[1].Value = value;
            tb_Logs.Invoke(new MethodInvoker(delegate
            {
                logMessage("Ecriture reçu à l'adresse : " +
(register - 1).ToString() + " | de valeur : " + value.ToString() + " | et de type : "
+ row.Cells[2].Value.ToString());
            }));
            break;

        case "Float":
            valueFloat =
EasyModbus.ModbusClient.ConvertRegistersToFloat(valueTwoRegisters);
            row.Cells[1].Value = valueFloat;
            tb_Logs.Invoke(new MethodInvoker(delegate
            {
                logMessage("Ecriture reçu à l'adresse : " +
(register - 1).ToString() + " | de valeur : " + valueFloat.ToString() + " | et de type
: " + row.Cells[2].Value.ToString());
            }));
            break;
    }
}
}
}
catch (Exception err)
{
    logMessage("Erreur de lecture : " + err.Message);
}
}

```

Fonction lecture client des registres du serveur toute les (#ValeurPooling) \*  
millisecondes

```
private void Timer1_Tick(object sender, EventArgs e)
{
    if (TCPEquipement.Mode == "Client")
    {
        int[] registersTable =
TCPEquipement.Modbus_Client.ReadHoldingRegisters(0, AdresseMax + 30);

        foreach (int num in registersTable)
        {
            Console.WriteLine(num.ToString() + ";");
        }
        Console.WriteLine("");
        Console.ReadLine();

        foreach (DataGridViewRow row in dgv_dataHolder.Rows)
        {
            if (row.Cells[1].Style.BackColor != Color.FromArgb(255, 110, 50,
50))
            {
                int adresse = Convert.ToInt32(row.Cells[3].Value);
                string type = row.Cells[2].Value.ToString();
                int value = 0;
                Single valueFloat = 0;
                int[] valueTwoRegisters = new int[2];
                switch (type)
                {
                    case "U16":
                        value = Convert.ToInt16(registersTable[adresse]);
                        if (value < 0)
                        {
                            value = 65535 + (value + 1);
                        }
                        value = Convert.ToUInt16(value);
                        row.Cells[1].Value = value;
                        break;
                    case "U32":
                        valueTwoRegisters[0] = registersTable[adresse];
                        valueTwoRegisters[1] = registersTable[adresse + 1];
                        value =
EasyModbus.ModbusClient.ConvertRegistersToInt(valueTwoRegisters);
                        UInt32 valUInt32 = 0;
                        if (value < 0)
                        {
                            valUInt32 = Convert.ToUInt32(4294967295L + (value
+1));
                        }
                        else
                        {
                            valUInt32 = Convert.ToUInt32(value);
                        }
                        row.Cells[1].Value = valUInt32;
                        break;

                    case "I16":
```

```

        value = Convert.ToInt16(registersTable[adresse]);
        row.Cells[1].Value = value;
        break;

    case "I32":
        valueTwoRegisters[0] = registersTable[adresse];
        valueTwoRegisters[1] = registersTable[adresse + 1];
        value =
EasyModbus.ModbusClient.ConvertRegistersToInt(valueTwoRegisters);
        row.Cells[1].Value = value;
        break;

    case "Float":
        valueTwoRegisters[0] = registersTable[adresse];
        valueTwoRegisters[1] = registersTable[adresse + 1];
        valueFloat =
EasyModbus.ModbusClient.ConvertRegistersToFloat(valueTwoRegisters);
        row.Cells[1].Value = valueFloat;
        break;
    }
}
}
}
} //Pour que le client lis les registers du serveur en fonction du pooling

```

Avec ceci de fait, nous avons presque terminé la communication entre des équipements Modbus TCP/IP, il reste quelque changement et peaufinage à faire puis il faudra faire pareil pour RTU et Profinet



### 3. Outils utilisés

Nom de l'outil	Fonction de l'outil
<b>Visual Studio 2019</b>	Espace de développement avec multiple langage de programmations
<b>VMWare Workstation 16</b>	VMware Workstation est un outil de virtualisation de poste de travail, il sert à mettre en place un environnement de test pour développer de nouveaux logiciels, ou pour tester l'architecture complexe d'un système d'exploitation avant de l'installer réellement sur une machine physique.

## 4. Conclusion.

Pour conclure sur cette semaine, j'ai fait je trouve un travail de malade, je ne me suis jamais senti aussi productif avec une découverte de nouvelle technologie, de fonctionnement et de matériel, j'ai repris depuis le début l'application et j'ai bien planifier mes tâches avant de partir tête baissée.






Grâce à ça j'ai pu avancer vite, vite corriger les bugs et les problèmes même en l'absence de mon tuteur dès le mardi.

Il reste encore plusieurs choses à faire et ceci risque d'être ma tâche principale jusqu'à la fin de mon stage si je ne la finis pas avant.

Sachant que cette application va sûrement être présentée par la suite à des personnes professionnelles, j'ai hâte de continuer et de fournir un travail propre, fonctionnel et qui a de la finalité.

J'attends donc le retour de mon tuteur la semaine prochaine pour son avis sur l'avancement de l'application ainsi que les changements apportés pour pouvoir ensuite trouver un accord sur la continuité du développement de l'application.

## 5. Niko-Niko.

JOUR	RESSENTI	TACHE
Lundi		Nouveau projet + Passage du permis
Mardi		Apprentissage du fonctionnement des Modbus
Mercredi		Découverte du projet + étude de maquette de projet (Brainstorming sur mise en place de modifications et de fonctionnalités)
Jeudi		Création interface from scratch + Fonctionnement graphique + résultat épreuve permis (Obtenue)
Vendredi		Mise en place fonctionnement connexions et communications