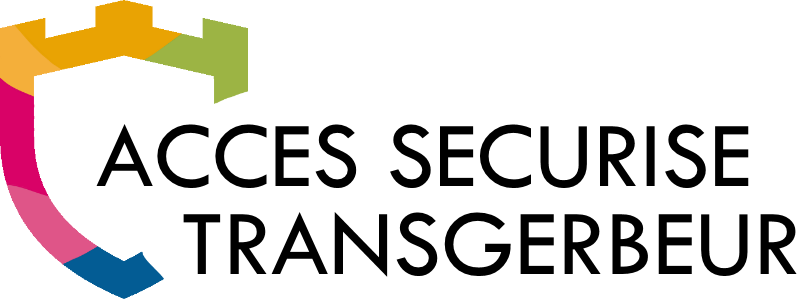
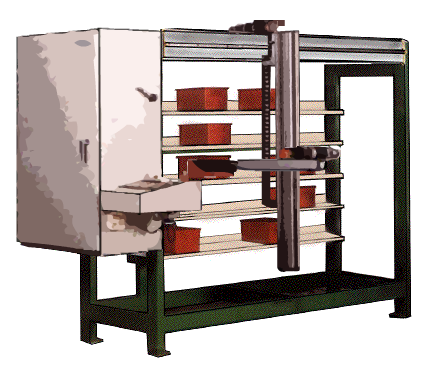
CHANCEREL Nicolas Session 2017

Projet BTS SN | Accès Sécurisé Transgerbeur







Professeur référent : Monsieur HOURDIN (BTS SN)

Monsieur GILBERT (BTS Maintenance)

SOMMAIRE

PRESENTATION DU PROJET 2

RESUME 3

ROLE PERSONNEL 4

SPECIFICATIONS FONCTIONNELLES 4

LECTURE D’UN BADGE 5

GESTION D’ACCES 5

VISUALISATION ETAT DE L’ACCES 5

CONSULTATION HISTORIQUE 6

CREATION BADGE D’ACCES 6

GESTION DES BADGES 6

SPECIFICATIONS NON FONCTIONNELLES 7

CONTRAINTE FINANCIERES 7

CONTRAINTE DE DEVELOPPEMENT 7

CONTRAINTE FIABILITE, SECURITE 7

IHM  7

ETAT DU SYSTEME 7

GESTION DES BADGES 8

Technologies 11

MODBUS TCP 11

WPF 11

LANGAGES 11

Analyse 11

AFFICHAGE ETAT SYSTEME 11

GESRION DES BADGES 11

HISTOTIQUE 11

PRESENTATION DU PROJET

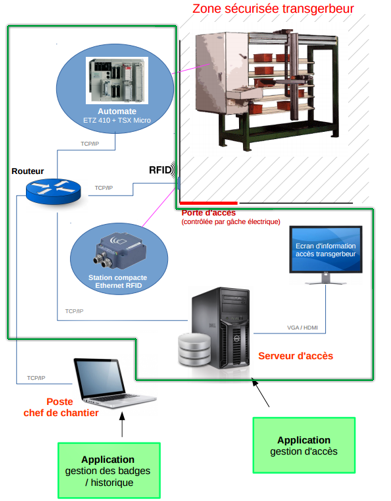
Résume :

Le transgerbeur est un système robotisé sur 3 axes, capable de stocker et déstocker des caisses référencées sur ordre d’un opérateur. Nous souhaitons, à travers ce projet, sécuriser l’accès au transgerbeur grâce à une gestion d’accès par badge.

Les opérateurs pourront y accéder selon leur niveau de qualification et selon son état (Marche | Arrêt | en cycle). Une Serrure électrique sera commandée par l’automate pour contrôler l’accès.

L’enregistrement de toutes les tentatives d’accès réussies ou échouées permettra à une personne responsable d’avoir un suivi.

Synoptique du système avec gestion d’accès



Partie  
 à réaliser

DIAGRAMME

Diagramme de déploiement :

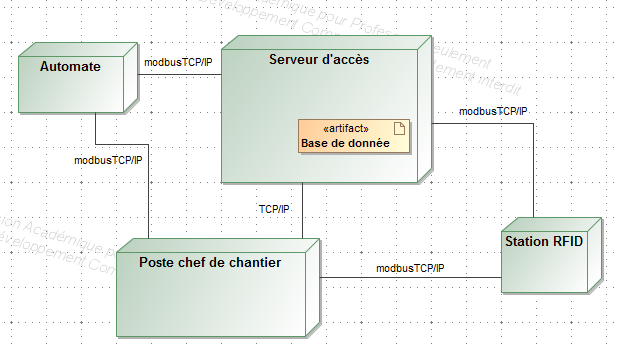


Diagramme de Cas d’utilisation :

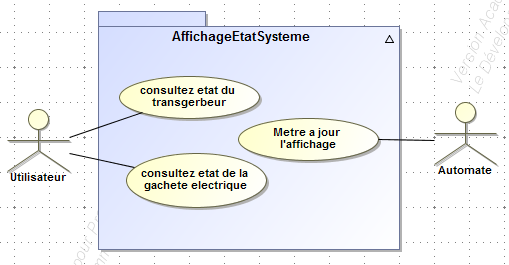
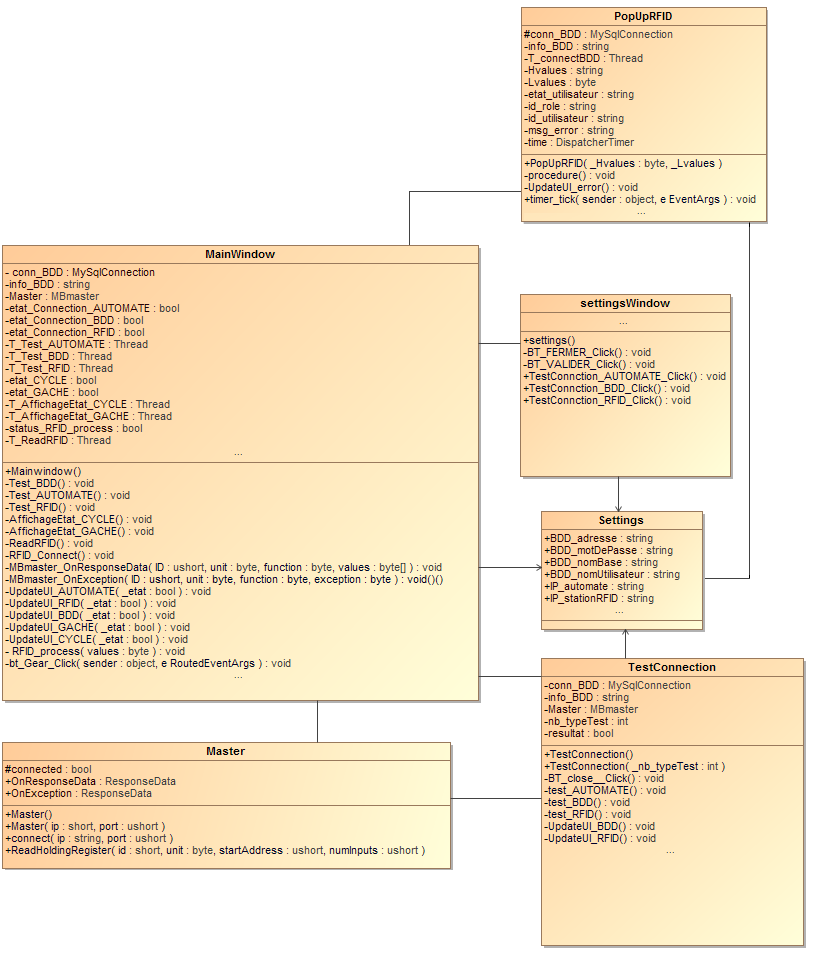
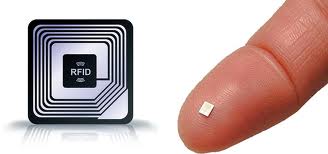


Diagramme de Classe :

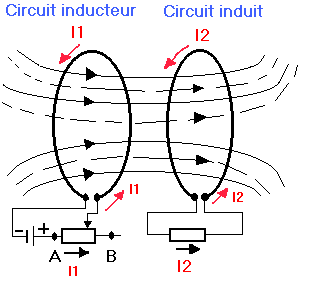


Technologie RFID

La Radio-Identification, le plus souvent désignée par le sigle RFID (Radio Frequency Identification), est une méthode pour mémoriser et récupérer des données à distance en utilisant des marqueurs appelés « Tag RFID ».

Le Lecteur RFID est un dispositif actif qui émet des radiofréquences pour activer une puce qui passe à proximité, et récupérer le tag RFID envoyé par la puce.

La puce RFID est, dans le cadre de ce projet, un dispositif passif, ne nécessitant aucune source d’énergie en dehors de celle fournie par le lecteur au moment de l’interrogation. Elle est composée d’une antenne et d’une puce de silicium.



Modbus TCP

MODBUS est un protocole de communication non-propriétaire, utilisé pour des réseaux d'automates programmables, relevant du niveau 7 (applicatif) du Modèle OSI. En mode TCP il fonctionne sur le mode Client / Serveur. Les clients sont tous actifs, le serveur est complètement passif. Chaque client lit et écrit dans le serveur.

http://www.ac-grenoble.fr/ecole.entreprise/CRGE/cteressources/modbus/Protocole\_Modbus.pdf

Fenêtre principal « MainWindow »

Fonctionnement de la fenêtre :

La fenêtre MainWindow est la fenêtre principale de l’application de Gestion d’accès. L’utilisateur peut grâce à cette fenêtre :

- visualiser l’état de la gâche électrique et le statut du cycle de l’automate en cours

- visualiser l’état de connexion de la base de données, de l’automate et de la station RFID ;

- visualiser l’état de l’application par rapport a une image animées (.GIF) ;

- vérifier la présence d’un badge sur la station RFID puis lancer la procédure de traitement de celui-ci après la récupération des informations nécessaires ;

- ouvrir la fenêtre de configuration de l’application

Interface Home Machine :

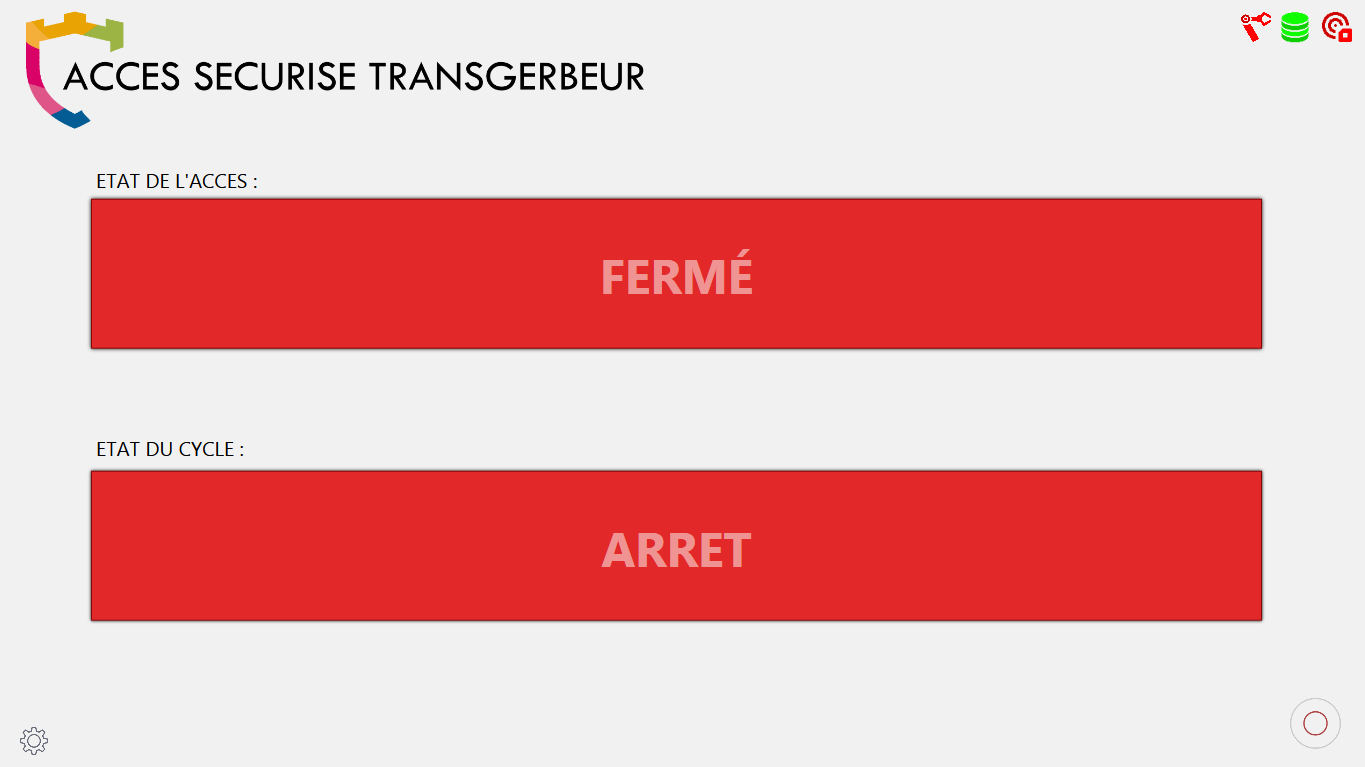
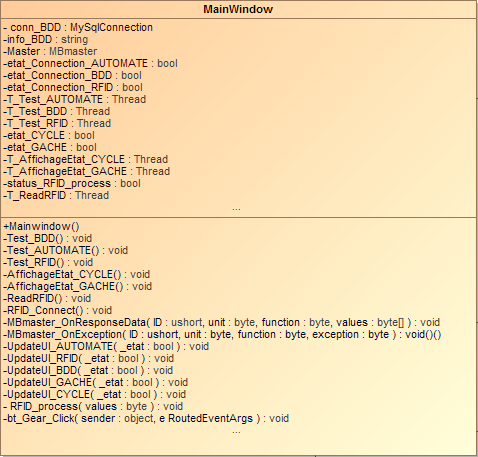


Diagramme de la classe :



Thread de récupération du badge RFID :

Le Thread « T\_ReadRFID » envoi, toute les 0.5s, une requête de lecture du badge a la station RFID, suivant la réponse de la station RFID, le Thread fais appelle au Delegate\* qui exécute le traitement l’identifiant du badge

\*Delegate : variables qui « pointent » vers une méthode.

THREAD de lecture de la station RFID tous les 500ms

#region THREAD de lecture de la sation RFID tout les 500ms

private void ReadRFID()

{

while (true) {

while (etat\_Connection\_RFID)

{

ushort ID = ushort.Parse("3");

byte unit = Convert.ToByte(1);

ushort StartAddress = 0;

byte Length = Convert.ToByte(32);

MBmaster.ReadHoldingRegister(ID, unit, StartAddress, Length); //requete RFID

Task.Delay(500).Wait();

}

}

}

#endregion

Action effectué si aucun Badge n’est présent

#region ACTION EFFECTUE SI UNE CARTE EST LUE  
 private void MBmaster\_OnException(ushort id, byte unit, byte function, byte exception) { }  
 #endregion

Action effectué si un badge est présent

#region ACTION EFFECTUE SI AUCUNE CARTE EST PRESENTE  
 private void MBmaster\_OnResponseData(ushort ID, byte unit, byte function, byte[] values)

{

Dispatcher.Invoke((DEL\_RFID\_process)RFID\_process, values);

}  
 #endregion

#region PROCEDURE DE TRAITEMENT D'UN TAG RFID  
 private void RFID\_process(byte[] values)

{  
 if (!status\_RFID\_process)

{

status\_RFID\_process = true;

PopUpRFID Popup = new PopUpRFID(values[0],values[1])

{

Topmost = true

};

var darkwindow = new Window()

{

Background = Brushes.Black,

Opacity = 0.4,

AllowsTransparency = true,

WindowStyle = WindowStyle.None,

WindowState = WindowState.Maximized

};

darkwindow.Show();

Popup.ShowDialog();

darkwindow.Close();

Popup.Close();

Task.Delay(100).Wait();

status\_RFID\_process = false;

}

}  
 #endregion

Thread de mise à jour d’un indicateur de connexion :=²²

Résultat de recherche d'images pour "red cross .png"http://www.freeiconspng.com/uploads/check-mark-31.png« Test\_RFID() » est une méthode qui effectue une connexion de test toute les seconde, puis met à jours l’indicateurs suivant le retour de la méthode « RFID\_Connect() » ( True : , False : )

#region procedure de test connection Test\_RFID  
 private void Test\_RFID()  
 {

while (true) {

if (RFID\_Connect())

{

etat\_Connection\_RFID = true;

Dispatcher.Invoke(new Action(() => UpdateUI\_RFID(true)));

}

else

{

etat\_Connection\_RFID = false;

Dispatcher.Invoke(new Action(() => UpdateUI\_RFID(false)));

}

Task.Delay(1000).Wait();

}

}

#endregion

#region | RFID

private void UpdateUI\_RFID(bool \_etat)

{

if (\_etat)

{

BitmapImage img\_RFID\_ON = new BitmapImage();

C:\Users\nicolas.chancerel\Documents\GitHub\GestionAcces\DOCUMENTATION\IMAGES\INDICATEURS\RFID_ON.png img\_RFID\_ON.BeginInit();

img\_RFID\_ON.UriSource = new

Uri("/AfficherEtatTransgerbeur;component/img/RFID\_ON.png",

UriKind.Relative);

img\_RFID\_ON.EndInit();

RFID\_status.Source = img\_RFID\_ON;

}

else

{

BitmapImage img\_RFID\_OFF = new BitmapImage();

C:\Users\nicolas.chancerel\Documents\GitHub\GestionAcces\DOCUMENTATION\IMAGES\INDICATEURS\RFID_OFF.png img\_RFID\_OFF.BeginInit();

img\_RFID\_OFF.UriSource = new

Uri("/AfficherEtatTransgerbeur;component/img/RFID\_OFF.png",

UriKind.Relative);

img\_RFID\_OFF.EndInit();

RFID\_status.Source = img\_RFID\_OFF;

}

}

#endregion

Thread de mise à jour d’état :

Les Threads « T\_affichageEtat\_CYCLE » et « T\_affichageEtat\_GACHE » récupère grâce à la méthode « get\_etat\_CYCLE » et « get\_etat\_GACHE » qui utilisent la passerelle réseau de l’automate pour collecter, toute les 25 ms, les valeurs correspondants à l’état du cycle de l’automate et de la gâche.

#region procedure d'Affichage Etat GACHE

private void AffichageEtat\_GACHE()

{

while (get\_etat\_GACHE()) {

Dispatcher.Invoke(new Action(() => UpdateUI\_GACHE(etat\_gache)));

Task.Delay(500).Wait();

}

}

#endregion

Fenêtre principal « MainWindow »

Fonctionnement de la fenêtre :

La fenêtre PopUpRFID est la fenêtre qui effectue le traitement d’un identifiant :

- récupérer les informations de l’utilisateur (nom, prénom, rôle),

- effectuer un archivage du passage du badge,

- envoyer une demande d’ouverture à l’automate.

L’utilisateur pourra visualisez :

- si son badge est affecté,

- si l’accès lui est autorisé ou refusé.

Interface Home Machine :

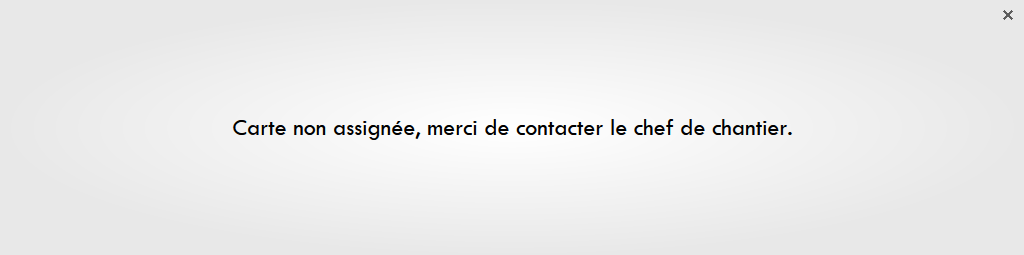
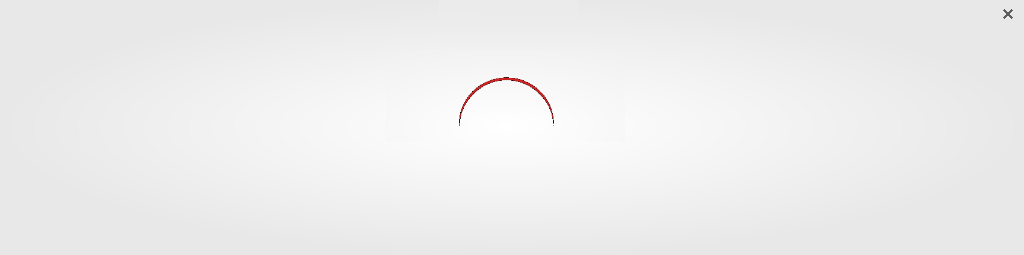
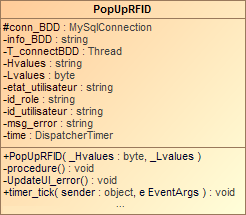


Diagramme de la classe :



Gestion de l’espace de la carte RFID :

Les Cartes utilisées ont 256 octets de mémoire donc nous utilisons 2 octets pour l’identifiant, ce qui donne 216 soit 65536 Identifiant possible.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | < - - - - - - - - - - - - - - - - - - - - - - - - -   **253o**  - - - - - - - - - - - - - - - - - - - - -- - > | | | | |
| **2o** | **1o** |  | **1o** |  | **1o** |  |
| TagRFID | Role | Nom | # | Prénom | $ | null |

Constructeur de la fenêtre « PopUpRFID » :

#region MAIN

public PopUpRFID(byte \_Hvalues, byte \_Lvalues)

{

InitializeComponent();

this.Hvalues = \_Hvalues;

this.Lvalues = \_Lvalues;

T\_connectBDD = new Thread(procedure);

T\_connectBDD.IsBackground = true;

T\_connectBDD.Start();

}

#endregion

Le constructeur récupère en paramètre les deux octets du Tag RFID sous forme Décimal, puis créer et exécute un Thread nommé « T\_connectBDD » .

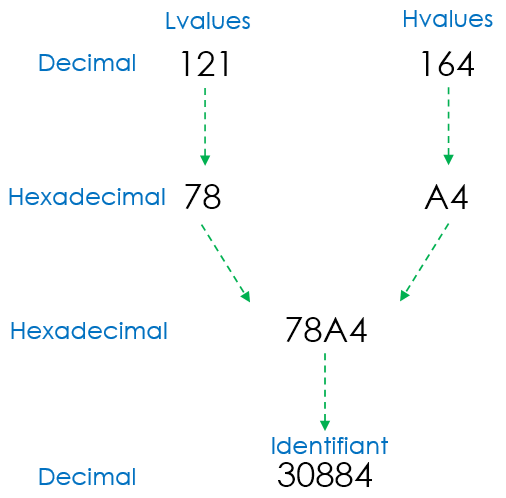
Procédure de traitement d’un Tag RFID :

string myHex = Hvalues.ToString("X") + Lvalues.ToString("X");

int TagRFID = Convert.ToInt32(myHex, 16);

“Hvalues.ToString("X")” sert à convertir une valeur décimal en hexadécimal.

«Convert.ToInt32(myHex, 16)” sert à convertir une valeur hexadécimal en décimal.



Fenêtres « Settings » et TestConnection

Les paramètres d'application permettent de stocker et de récupérer dynamiquement des paramètres de propriété et d'autres informations pour votre application. Ils vous permettent également de conserver les préférences personnalisées d'utilisateur et d'application sur un ordinateur client.

Fonctionnement des fenêtres:

La fenêtre Settings est la fenêtre de configuration de l’application, l’utilisateur pourra y modifier :

- l’adresse de connexion à la base de données,

- le nom de la base de données,

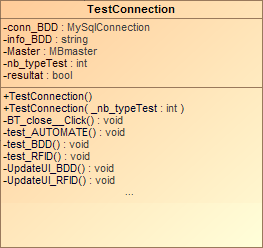
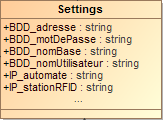
- l’identifiant de l’utilisateur MySql\*,

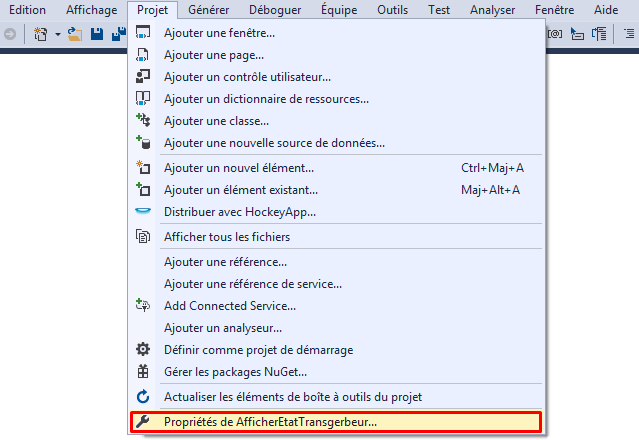
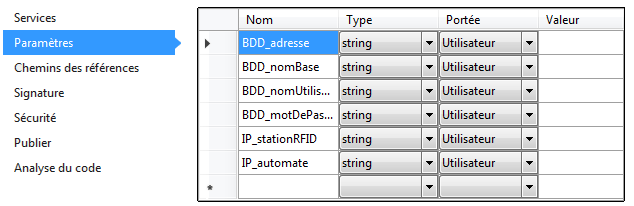
- le mot de passe de l’utilisateur MySql,

- l’adresse de connexion à la station RFID,

-l’adresse de connexion à l’automate.

Puis effectuer des Test de Connexion vers l’automate, la station RFID et la base de données.

Diagrammes de classes :

Utilisation des Paramètres d’applications :

Portée Utilisateur et APPlication ?????

Récupération des valeurs dans le constructeur :

public settings()

{

InitializeComponent();

BDD\_adresse.Text = Properties.Settings.Default.BDD\_adresse;

BDD\_nomBase.Text = Properties.Settings.Default.BDD\_nomBase;

BDD\_nomUtilisateur.Text = Properties.Settings.Default.BDD\_nomUtilisateur;

BDD\_motDePasse.Password = Properties.Settings.Default.BDD\_motDePasse;

IP\_automate.Text = Properties.Settings.Default.IP\_automate;

IP\_stationRFID.Text = Properties.Settings.Default.IP\_stationRFID;

}

“Properties.Settings.Default.BDD\_adresse” par exemple correspond a un accesseur et retourne donc la valeur de « BDD\_adresse » pour l’affecte dans le Textbox « BDD\_adresse ».

Sauvegarde des valeurs entrées :

private void BT\_VALIDER\_Click(object sender, RoutedEventArgs e)

{

Properties.Settings.Default["BDD\_adresse"] = BDD\_adresse.Text;

Properties.Settings.Default["BDD\_nomBase"] = BDD\_nomBase.Text;

Properties.Settings.Default["BDD\_nomUtilisateur"] = BDD\_nomUtilisateur.Text;

Properties.Settings.Default["BDD\_motDePasse"] = BDD\_motDePasse.Password;

Properties.Settings.Default["IP\_automate"] = IP\_automate.Text;

Properties.Settings.Default["IP\_stationRFID"] = IP\_stationRFID.Text;

Properties.Settings.Default.Save();

}

“ Properties.Settings.Default["BDD\_adresse"] ” par exemple correspond a un setteur et pointe donc vers le parametre « BDD\_adresse » pour y affecter la valeur du Textbox « BDD\_adresse ».