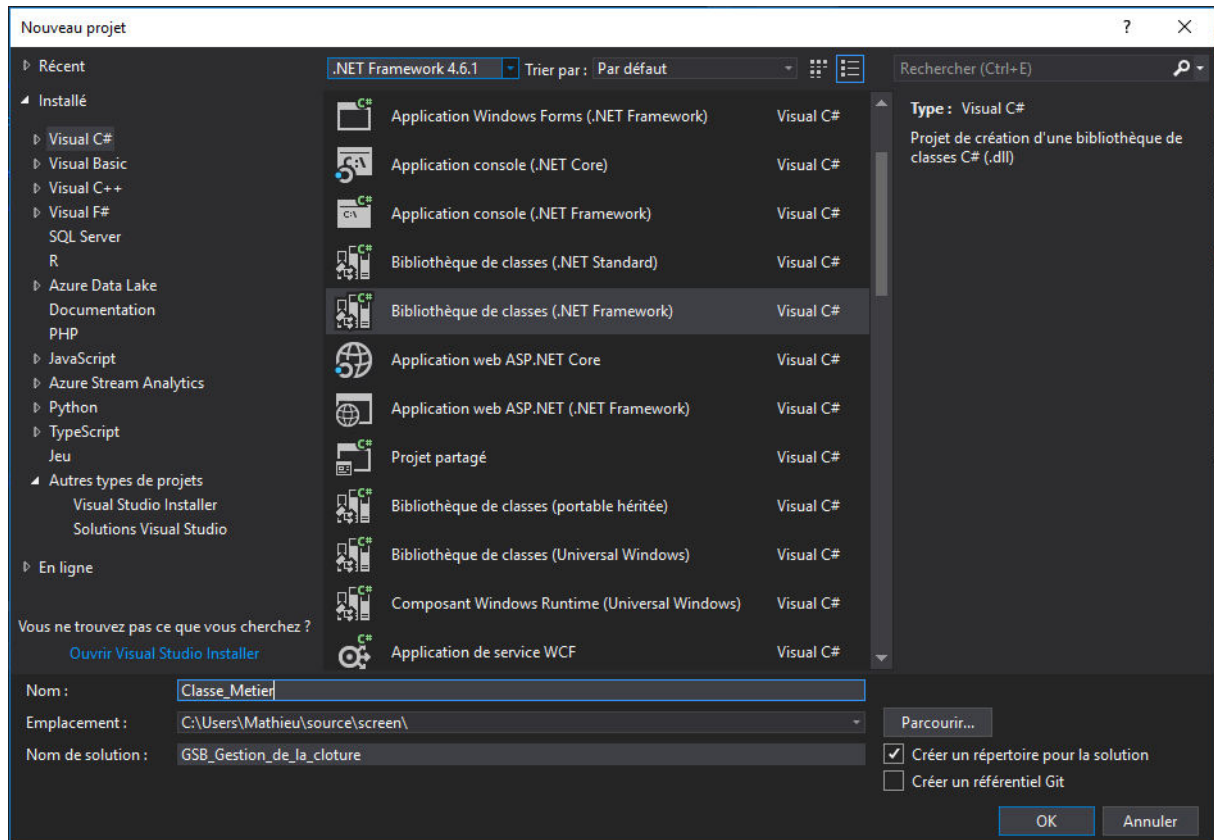


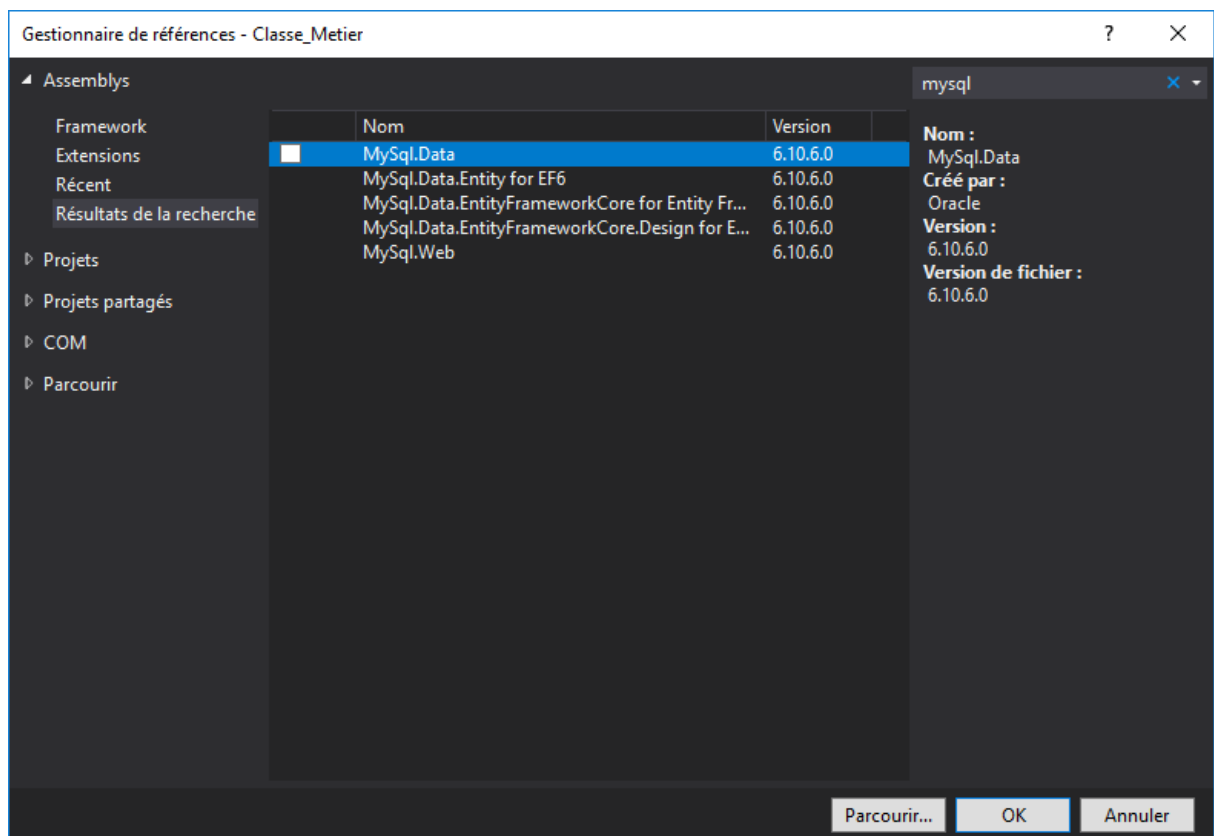
Tâche 1 : Création de la classe d'accès aux données

Le but de cette tâche est de créer les méthodes qui vont nous permettre de communiquer avec la base de donnée, que ce soit avec des requêtes de type SELECT, UPDATE, INSERT,DELETE

On commence donc par créer la bibliothèque de classe qui va comporter ces méthodes



Nous définissons une référence avec la bibliothèque MySql.Data afin de développer les méthodes



Tout d'abord nous avons développé la méthode de Connexion à la base de donnée

```
MySqlConnection cnx;  
MySqlCommand cmd;  
/// Résumé :  
/// <summary>  
///     Méthode utilisée pour connecter son utilisateur à la base de donnée GSB_frais.  
/// </summary>  
/// <returns>  
///     Un System.String contenant un message relatif à la connexion ou non à la base de donnée.  
/// </returns>  
0 références | 0 modification | 0 auteur, 0 modification  
public string ConnexionDB()  
{  
    try  
    {  
        cnx = new MySqlConnection("database=gsb_frais; server=localhost; UID=root; pwd = '');  
        cnx.Open();  
        return "Connexion à la base réussie";  
    }  
    catch (Exception e)  
    {  
        return "Erreur lors de la connexion à la base : " + e;  
    }  
}
```

Puis nous développons la méthode permettant de passer les requêtes SELECT

```
/// Résumé :  
/// <summary>  
///     Méthode utilisée pour executer des requêtes de type SELECT.  
/// </summary>  
///  
/// Paramètres :  
/// <param name="theQuery">  
///     Un System.String qui contient la requête SELECT à executer.  
/// </param>  
///  
/// Retourne :  
/// <returns>  
///     Un MySql.Data.MySqlClient.MySqlDataReader contenant les lignes de la requête sous forme  
///     d'un tableau associatif.  
/// </returns>  
///  
1 référence | 0 modification | 0 auteur, 0 modification  
public MySqlDataReader QuerySelect(string theQuery)  
{  
    cnx.Close();  
    cnx.Open();  
    cmd = cnx.CreateCommand();  
    cmd.CommandText = theQuery;  
    return cmd.ExecuteReader();  
}
```

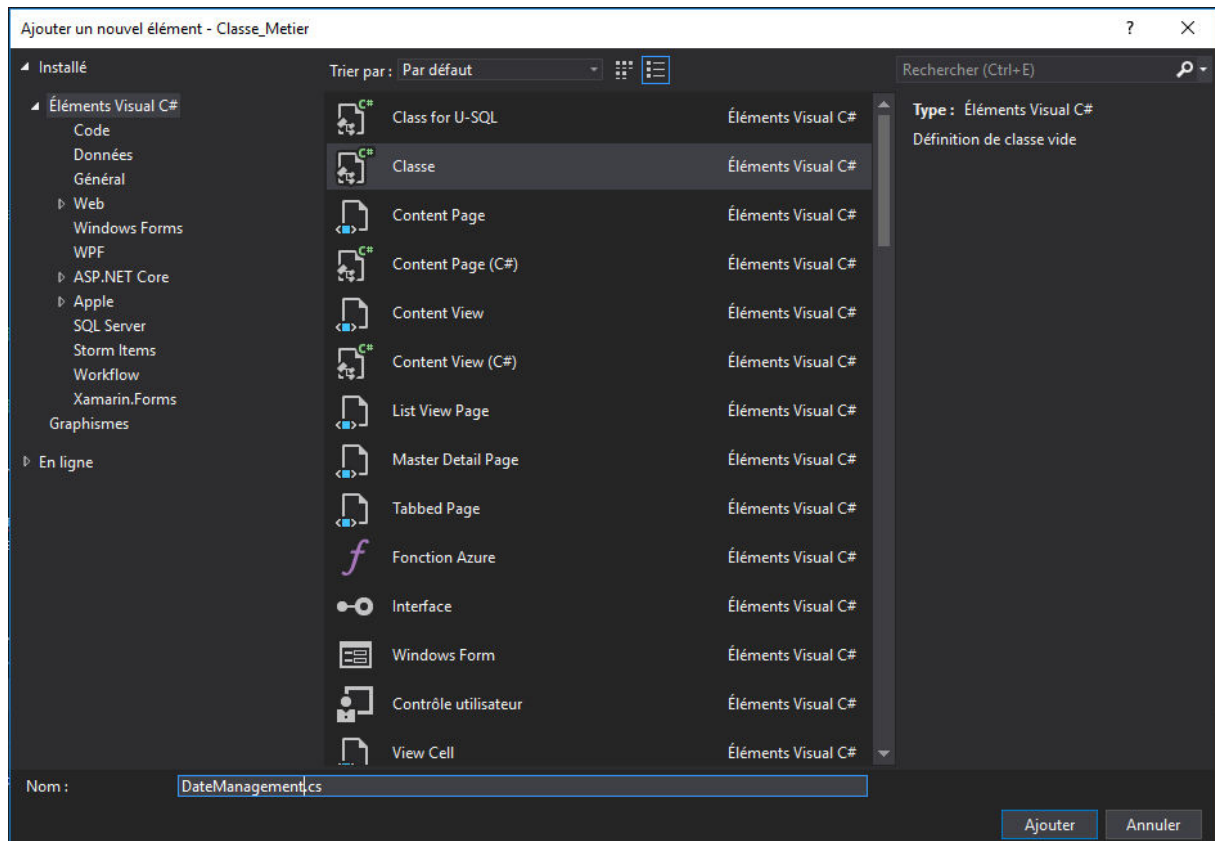
Puis la méthode pour les requêtes d'administration de la base de donnée

```
/// <summary>  
///     Méthode utilisée pour executer des requêtes de type INSERT INTO, UPDATE et DELETE FROM.  
/// </summary>  
///  
/// Paramètres :  
/// <param name="theQuery">  
///     Un System.String qui contient la requête INSERT INTO, UPDATE ou DELETE FROM à executer.  
/// </param>  
///  
/// Retourne :  
/// <returns>  
///     Un System.Int égal à 1 si la requête a bien été executée.  
/// </returns>  
///  
1 référence | 0 modification | 0 auteur, 0 modification  
public int QueryAdministrative(string theQuery)  
{  
    cnx.Close();  
    cnx.Open();  
    cmd = cnx.CreateCommand();  
    cmd.CommandText = theQuery;  
    return cmd.ExecuteNonQuery();  
}
```

Tâche 2 : Création d'une classe de gestion de dates

Le but de cette tâche est de créer une classe qui permettra de gérer les dates, et ainsi ultérieurement l'utiliser pour la sélection des fiches de frais.

On commence par créer la Classe DateManagement



On commence par créer la méthode GetMoisPrécédent() qui possède trois définitions,

```
public abstract class DateManagement
{
    /// <summary>
    ///     Méthode qui retourne le mois précédent.
    /// </summary>
    /// <returns>
    ///     Un System.String des deux chiffres du mois précédent.
    /// </returns>
    0 références | 0 modification | 0 auteur, 0 modification
    public static string GetMoisPrecedent()
    {
        return GetMoisPrecedent(DateTime.Now.Month);
    }

    /// <summary>
    ///     Méthode qui retourne le mois précédent.
    /// </summary>
    /// <param name="date">
    ///     Un System.DateTime qui contient la date à partir de laquelle le mois précédent va être récupéré.
    /// </param>
    /// <returns>
    ///     Un System.String des deux chiffres du mois précédent.
    /// </returns>
    0 références | 0 modification | 0 auteur, 0 modification
    public static string GetMoisPrecedent(DateTime date)
    {
        return GetMoisPrecedent(date.Month);
    }

    /// <summary>
    ///     Méthode qui retourne le mois précédent.
    /// </summary>
    /// <param name="mois">
    ///     Un System.Int qui contient le mois à partir duquel le mois précédent va être récupéré.
    /// </param>
    /// <returns>
    ///     Un System.String des deux chiffres du mois précédent.
    /// </returns>
    2 références | 0 modification | 0 auteur, 0 modification
    public static string GetMoisPrecedent(int mois)
    {
        string moisPrecedent;
        if (mois == 1)
        {
            moisPrecedent = "12";
        }
        else
        {
            if (mois <= 10)
            {
                moisPrecedent = "0" + (mois - 1).ToString();
            }
            else
            {
                moisPrecedent = (mois - 1).ToString();
            }
        }
        return moisPrecedent;
    }
}
```

Puis la méthode `getMoisSuivant()` qui possède elle aussi 3 définitions.

```
/// <summary>
///     Méthode qui retourne le mois suivant.
/// </summary>
/// <returns>
///     Un System.String des deux chiffres du mois suivant.
/// </returns>
0 références | 0 modification | 0 auteur, 0 modification
public static string GetMoisSuivant()
{
    return GetMoisSuivant(DateTime.Now.Month);
}
```

```
/// <summary>
///     Méthode qui retourne le mois suivant.
/// </summary>
/// <param name="date">
///     Un System.DateTime qui contient la date à partir de laquelle le mois suivant va être récupéré.
/// </param>
/// <returns>
///     Un System.String des deux chiffres du mois suivant.
/// </returns>
0 références | 0 modification | 0 auteur, 0 modification
public static string GetMoisSuivant(DateTime date)
{
    return GetMoisSuivant(date.Month);
}
```

```
/// <summary>
///     Méthode qui retourne le mois suivant.
/// </summary>
/// <param name="mois">
///     Un System.Int qui contient le mois à partir duquel le mois suivant va être récupéré.
/// </param>
/// <returns>
///     Un System.String des deux chiffres du mois suivant.
/// </returns>
```

```
2 références | 0 modification | 0 auteur, 0 modification
public static string GetMoisSuivant(int mois)
{
    string moisSuivant;
    if (mois == 12)
    {
        moisSuivant = "01";
    }
    else
    {
        if (mois < 9)
        {
            moisSuivant = "0" + (mois + 1).ToString();
        }
        else
        {
            moisSuivant = (mois + 1).ToString();
        }
    }
    return moisSuivant;
}
```

Et pour terminer la méthode `entre()` qui a elle aussi 3 définitions.

```
/// <summary>
///     Méthode qui vérifie si la date actuelle se situe entre deux jours.
/// </summary>
/// <param name="jour1">
///     Un System.Int qui contient le premier jour de l'entre deux.
/// </param>
/// <param name="jour2">
///     Un System.Int qui contient le dernier jour de l'entre deux.
/// </param>
/// <returns>
///     Un System.Boolean true si la date est entre ces deux jours.
/// </returns>
0 références | 0 modification | 0 auteur, 0 modification
public static bool entre(int jour1, int jour2)
{
    return entre(jour1, jour2, DateTime.Now.Day);
}

/// <summary>
///     Méthode qui vérifie si une date se situe entre deux jours.
/// </summary>
/// <param name="jour1">
///     Un System.Int qui contient le premier jour de l'entre deux.
/// </param>
/// <param name="jour2">
///     Un System.Int qui contient le dernier jour de l'entre deux.
/// </param>
/// <param name="date">
///     Un System.DateTime qui contient la date à vérifier.
/// </param>
/// <returns>
///     Un System.Boolean true si la date est entre ces deux jours.
/// </returns>
0 références | 0 modification | 0 auteur, 0 modification
public static bool entre(int jour1, int jour2, DateTime date)
{
    return entre(jour1, jour2, date.Day);
}
```

```

/// <summary>
///     Méthode qui vérifie si un jour testé se situe entre deux jours.
/// </summary>
/// <param name="jour1">
///     Un System.Int qui contient le premier jour de l'entre deux.
/// </param>
/// <param name="jour2">
///     Un System.Int qui contient le dernier jour de l'entre deux.
/// </param>
/// <param name="jourTest">
///     Un System.Int qui contient le jour à vérifier.
/// </param>
/// <returns>
///     Un System.Boolean true si le jour testé est entre ces deux jours.
/// </returns>

```

2 références | 0 modification | 0 auteur, 0 modification

```

public static bool entre(int jour1, int jour2, int jourTest)
{
    bool res = false;
    if (jour1 <= jourTest && jourTest <= jour2)
    {
        res = true;
    }
    return res;
}

```

Leurs différents usages est explicité dans l'en-tête des méthodes.

Nous allons ensuite créer la classe nous permettant de tester ces méthodes

Créer des tests unitaires

Infrastructure de tests : MSTest [Obtenir des extensions supplémentaires](#)

Projet de test : <Nouveau projet de test>

Format du nom du projet de test : [Project]Tests

Espace de noms : [Namespace].Tests

Fichier de sortie : <Nouveau fichier de test>

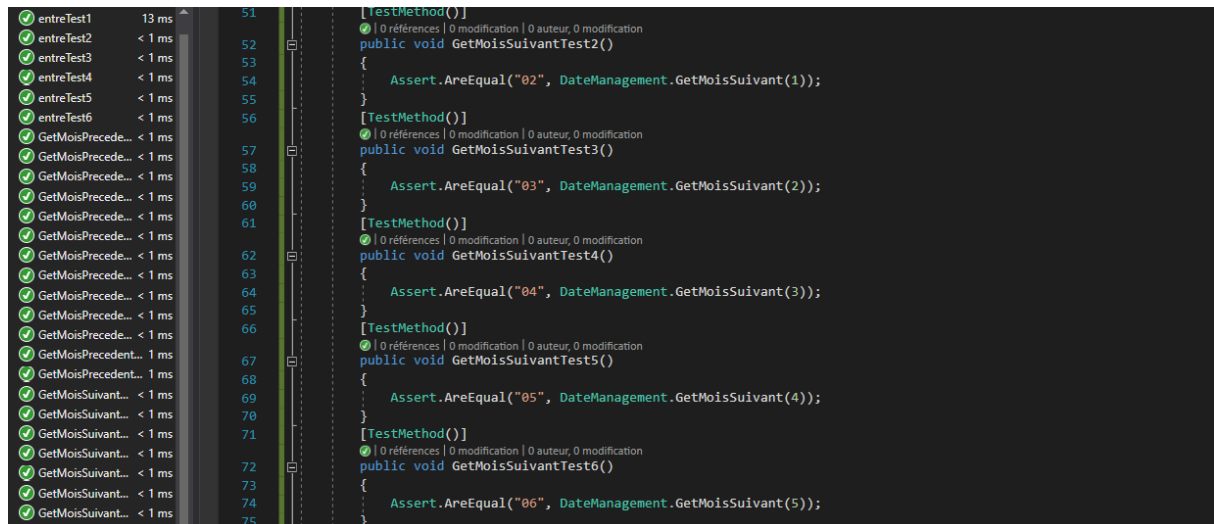
Format du nom de la classe de test : [Class]Tests

Format du nom de la méthode de test : [Method]Test

Code de la méthode de test : Échec d'assertion

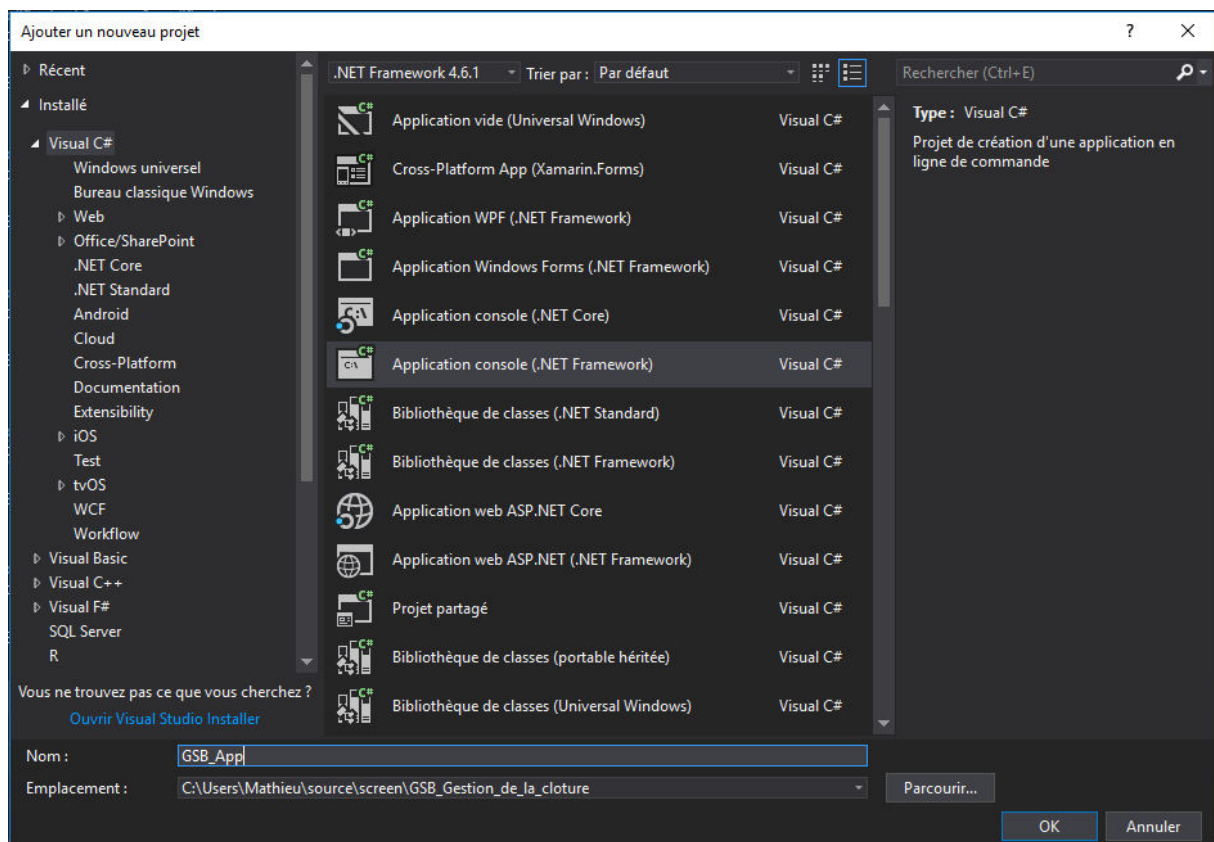
OK Annuler

Une fois cette classe créée nous allons définir tout une batterie de test nous permettant d'analyser le prisme des possibilités de ces méthodes afin de les valider.



Tâche 3 : Création de l'application

Nous allons ainsi créer une application nous permettant la validation et le remboursement automatique des fiches de frais selon la date actuelle.



Les conditions sont les suivantes :

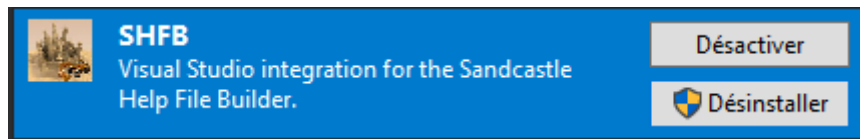
- Récupération des fiches créées du mois N-1 et leur mise à jour, en les mettant à l'état 'CL' ; en supposant que la campagne de validation va se passer entre le 1^{er} et le 10 du mois courant, on va, en comparant les dates, s'assurer que l'on se trouve bien dans cet intervalle-là
- De la même manière, à partir du 20^e jour du mois, on va mettre à jour les fiches validées du mois précédent en les passant à l'état 'RB'

```
0 références | 0 modification | 0 auteur, 0 modification
static void Main(string[] args)
{
    DataAccess da = new DataAccess();
    da.ConnexionDB();
    if (DateManagement.entre(1, 10, DateTime.Now))
    {
        da.ChangerEtat("CR", "CL");
    }
    else
    {
        if (DateManagement.entre(20, 31, DateTime.Now))
        {
            da.ChangerEtat("VA", "RB");
        }
    }
}
```

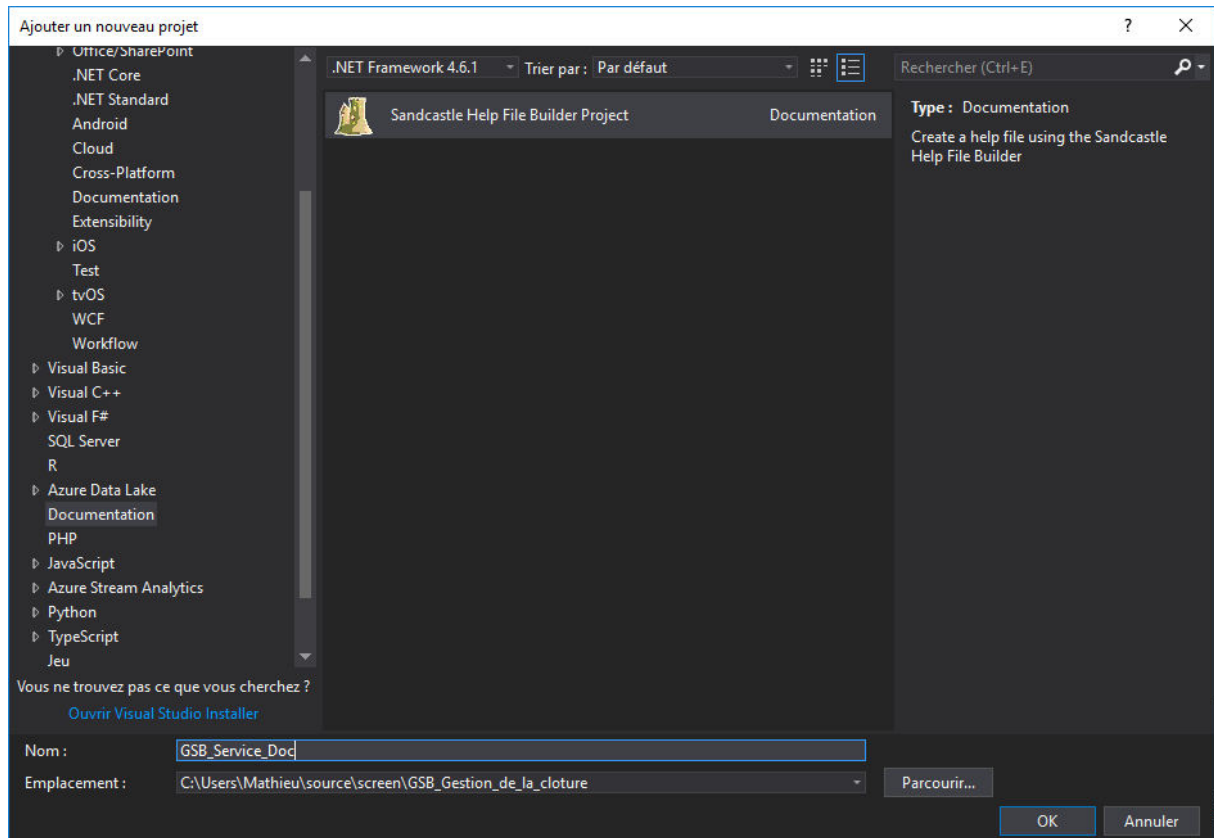
```
/// Résumé :
/// <summary>
///     Change l'état de toutes les fiches de frais du mois précédent à un autre état.
/// </summary>
///
/// Paramètres :
/// <param name="etatAChanger">
///     Un System.String qui contient l'état des fiches de frais à changer.
/// </param>
/// <param name="nouvelEtat">
///     Un System.String qui contient le nouvel état de ces fiches de frais.
/// </param>
4 références | MathieuPerroud, il y a 66 jours | 1 auteur, 1 modification
public void ChangerEtat(string etatAChanger, string nouvelEtat)
{
    string moisPrecedent = DateManagement.GetMoisPrecedent();
    MySqlDataReader reader = QuerySelect(String.Format(
        "SELECT idvisiteur "
        + "FROM fichefrais "
        + "WHERE mois LIKE '{0}' "
        + "AND idetat = '{1}'", moisPrecedent, etatAChanger));
    List<string> lignes = new List<string>();
    while (reader.Read())
    {
        lignes.Add((string)reader["idVisiteur"]);
    }
    foreach (string ligne in lignes)
    {
        QueryAdministrative(String.Format(
            "UPDATE fichefrais "
            + "SET idetat = '{0}' "
            + "WHERE idvisiteur = '{1}' "
            + "AND mois LIKE '{2}'", nouvelEtat, ligne, moisPrecedent));
    }
}
```

Nous allons ensuite générer la documentation technique des méthodes de l'application via SandCastle

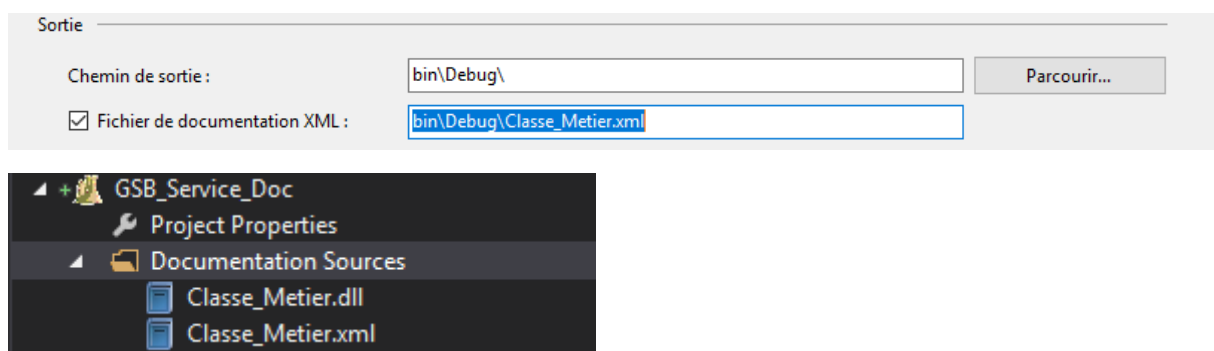
On télécharge l'extension SHFB



Puis on crée une classe Sandcastle



Nous paramétrons le chemin de référence pour la génération technique



Et finalement, voici un exemple d’une documentation technique générée avec SandCastle

Namespace: Classes_Metier

Assembly: Classe_Metier (in Classe_Metier.dll) Version: 1.0.0.0 (1.0.0.0)

The DateManagement type exposes the following members.

Constructors

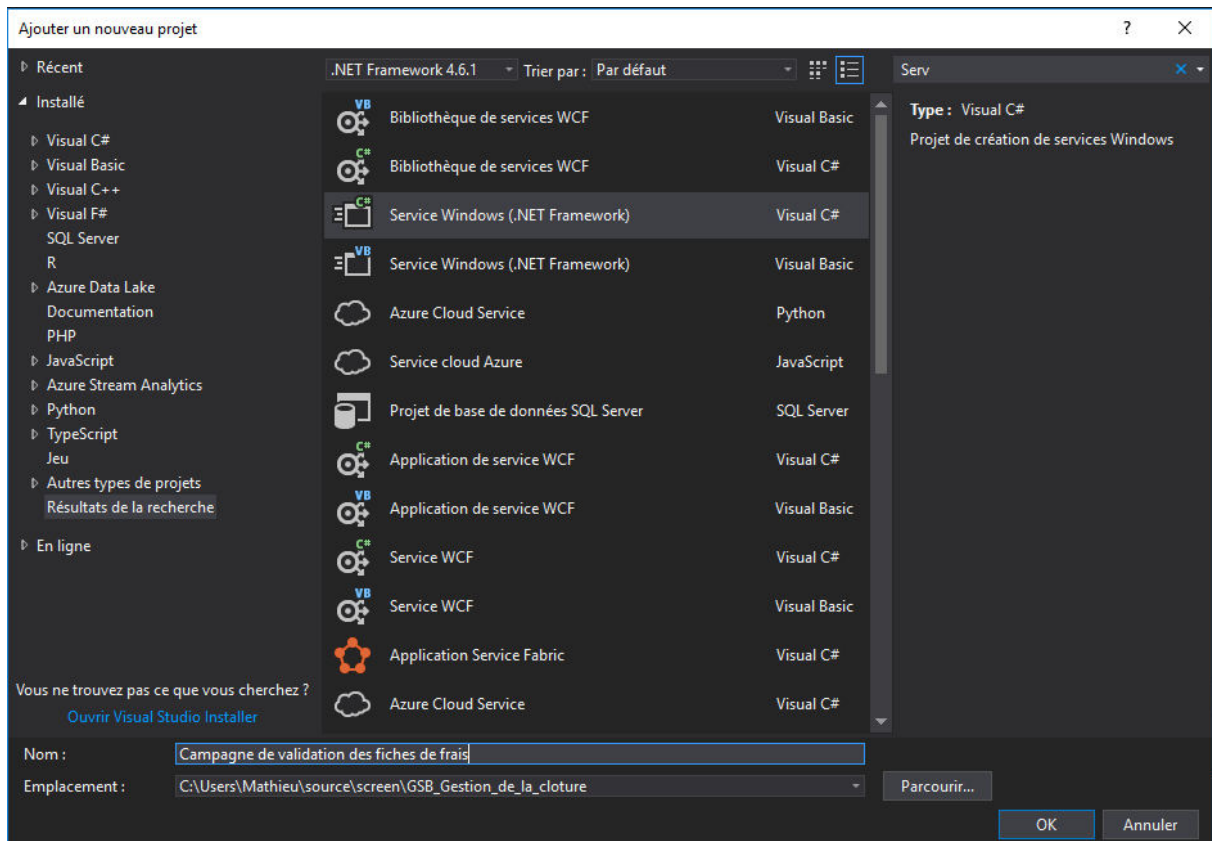
	Name	Description
	DateManagement	Initializes a new instance of the DateManagement class

[Top](#)

Methods

	Name	Description
S	entre(Int32, Int32)	Méthode qui vérifie si la date actuelle se situe entre deux jours.
S	entre(Int32, Int32, DateTime)	Méthode qui vérifie si une date se situe entre deux jours.
S	entre(Int32, Int32, Int32)	Méthode qui vérifie si un jour testé se situe entre deux jours.
	Equals	(Inherited from Object .)
	Finalize	(Inherited from Object .)
	GetHashCode	(Inherited from Object .)
S	GetMoisPrecedent()	Méthode qui retourne le mois précédent.
S	GetMoisPrecedent(DateTime)	Méthode qui retourne le mois précédent.
S	GetMoisPrecedent(Int32)	Méthode qui retourne le mois précédent.
S	GetMoisSuivant()	Méthode qui retourne le mois suivant.
S	GetMoisSuivant(DateTime)	Méthode qui retourne le mois suivant.
S	GetMoisSuivant(Int32)	Méthode qui retourne le mois suivant.
	GetType	(Inherited from Object .)
	MemberwiseClone	(Inherited from Object .)
	ToString	(Inherited from Object .)

Tâche 4 : Création d'un service Windows



Tout d'abord il fallait créer une solution de Service Windows

```
private System.ComponentModel.Container components = null;
Timer timer;
DataAccess da;
1 référence | MathieuPerroud, Il y a 17 heures | 1 auteur, 1 modification
public Service_de_Validation()
{
    InitializeComponent();
}
```

Nous définissons le nom du service à l'initialisation et nous initialisation un objet de la classe DataAccess

```
1 référence | MathieuPerroud, Il y a 17 heures | 1 auteur, 1 modification
private void InitializeComponent()
{
    //
    // Service_de_Validation
    //
    components = new System.ComponentModel.Container();
    this.ServiceName = "Validation des fiches de frais";
    da = new DataAccess();
    EventLog.WriteEntry(da.ConnexionDB());
}
```

Une fois que ceci est fait nous définissons l'événement OnStart() en y affectant un timer ayant une minute d'intervalle (afin de tester aisément le service) auquel on associe la méthode timer_tick qui définit le fonctionnement du Service Windows, et un évènement OnStop() pour procéder l'arrêt.

0 références | MathieuPerroud, il y a 17 heures | 1 auteur, 1 modification
protected override void OnStart(string[] args)

```
{
    timer = new Timer();
    // 1000 = 1 Seconde
    this.timer.Interval = 60000;
    this.timer.Elapsed += new ElapsedEventHandler(this.timer_tick);
    this.timer.Enabled = true;
}
```

1 référence | MathieuPerroud, il y a 17 heures | 1 auteur, 1 modification

private void timer_tick(object sender, ElapsedEventArgs e)

```
{
    if (DateManagement.entre(1, 10, DateTime.Now) == true)
    {
        EventLog.WriteEntry("Nous sommes le : " + DateTime.Now.Day + ". Les fiches de frais sont cloturées et en cours de traitement");
        da.ChangerEtat("CR", "CL");
    }
    else
    {
        if (DateManagement.entre(20, 31, DateTime.Now) == true)
        {
            da.ChangerEtat("VA", "RB");
            EventLog.WriteEntry("Nous sommes le : " + DateTime.Now.Day + ". Les fiches de frais sont Remboursées");
        }
        else
        {
            EventLog.WriteEntry("Nous sommes le : " + DateTime.Now.Day + ". Rien ne se passe");
        }
    }
    EventLog.WriteEntry("Le service s'est bien exécuté.");
}
```

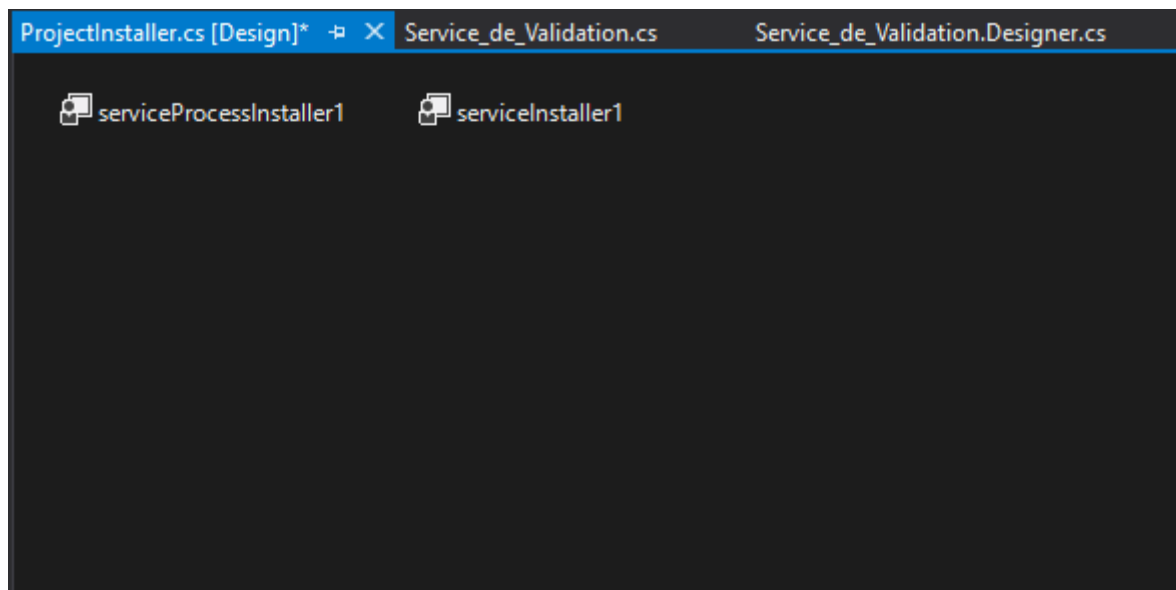
protected override void OnStop()

```
{
    EventLog.WriteEntry("Le service est arrêté");
    timer.Stop();
    timer = null;
}
```

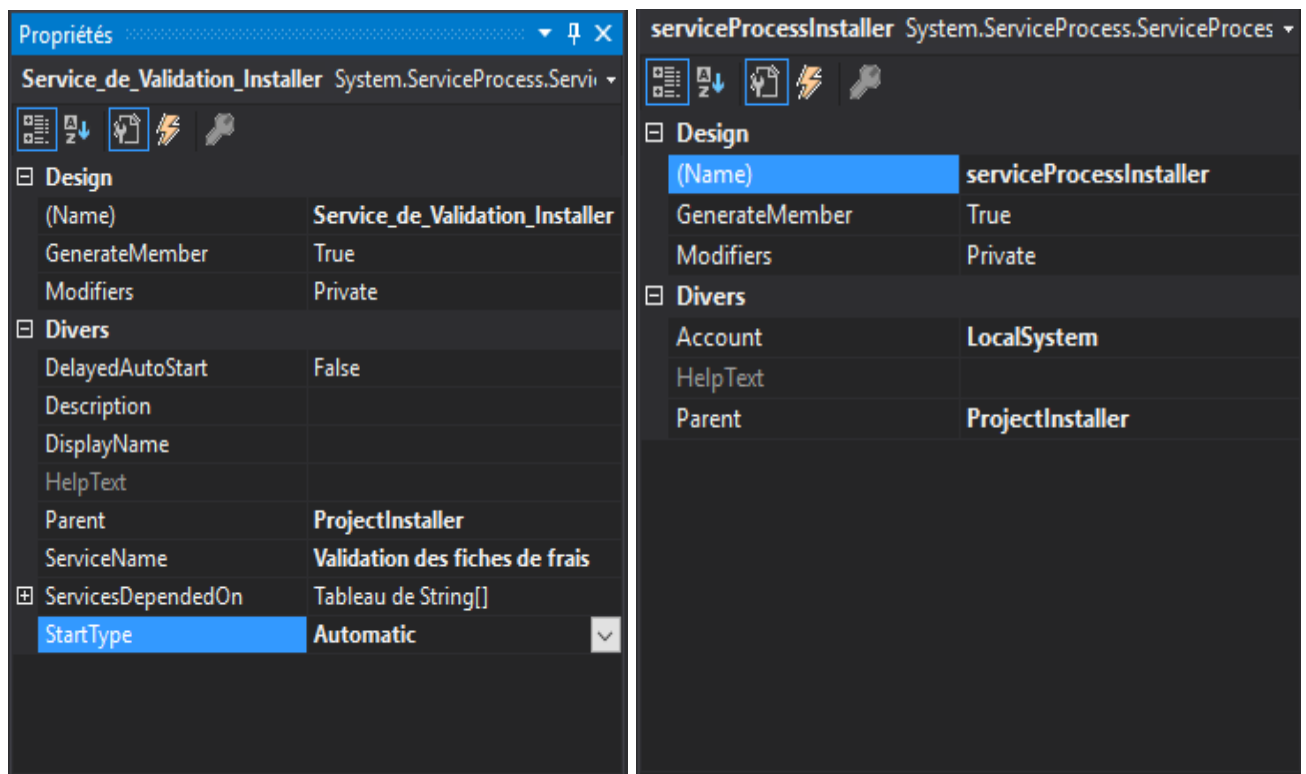
Voici le Main du programme du Service windows où on définit le service à lancer par la classe des méthodes précédemment citées

```
static class Program
{
    /// <summary>
    /// Point d'entrée principal de l'application.
    /// </summary>
    0 références | MathieuPerroud, Il y a 17 heures | 1 auteur, 1 modification
    static void Main()
    {
        ServiceBase[] ServicesToRun;
        ServicesToRun = new ServiceBase[]
        {
            new Service_de_Validation()
        };
        ServiceBase.Run(ServicesToRun);
    }
}
```

Nous procédons ensuite à l'installation du service



On définit bien les paramètres du service



The image displays two side-by-side screenshots of the Visual Studio IDE, specifically the Properties window for service installers.

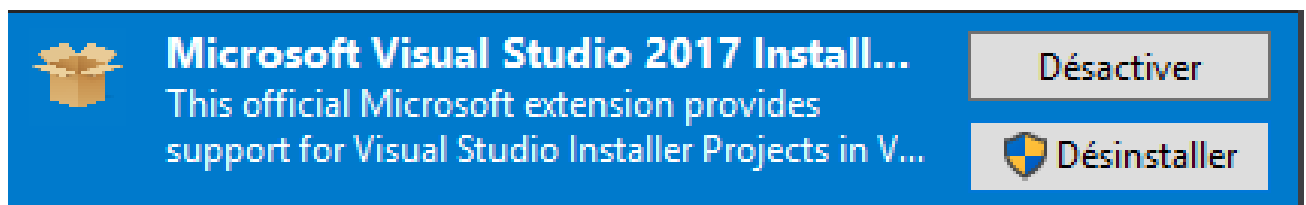
Left Screenshot: Service_de_Validation_Installer

Property	Value
(Name)	Service_de_Validation_Installer
GenerateMember	True
Modifiers	Private
DelayedAutoStart	False
Description	
DisplayName	
HelpText	
Parent	ProjectInstaller
ServiceName	Validation des fiches de frais
ServicesDependedOn	Tableau de String[]
StartType	Automatic

Right Screenshot: serviceProcessInstaller

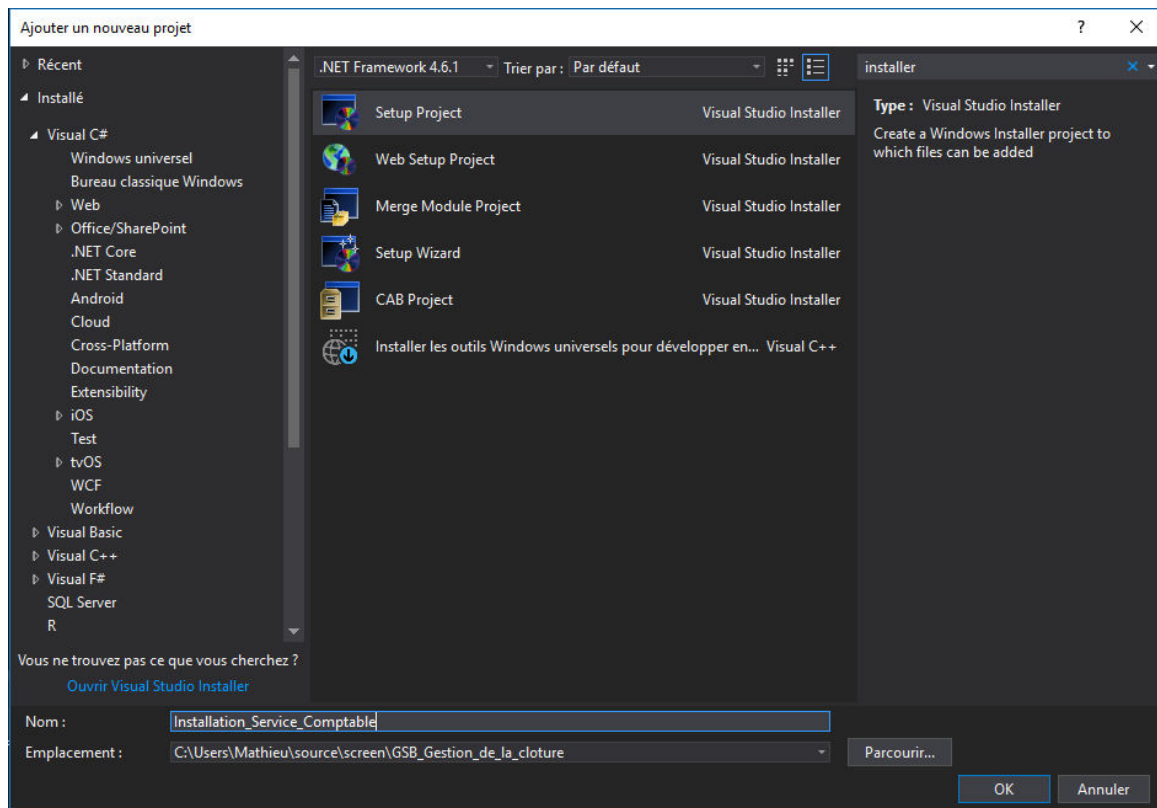
Property	Value
(Name)	serviceProcessInstaller
GenerateMember	True
Modifiers	Private
Account	LocalSystem
HelpText	
Parent	ProjectInstaller

Puis nous installons l'extension Visual Studio 2017 Installer afin de créer l'Installer du service

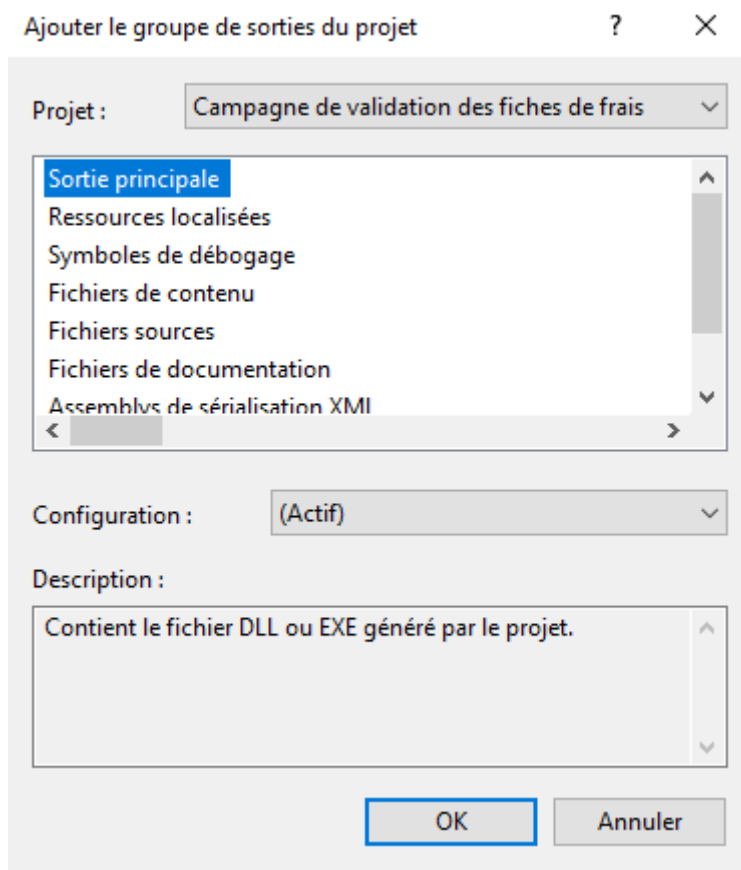


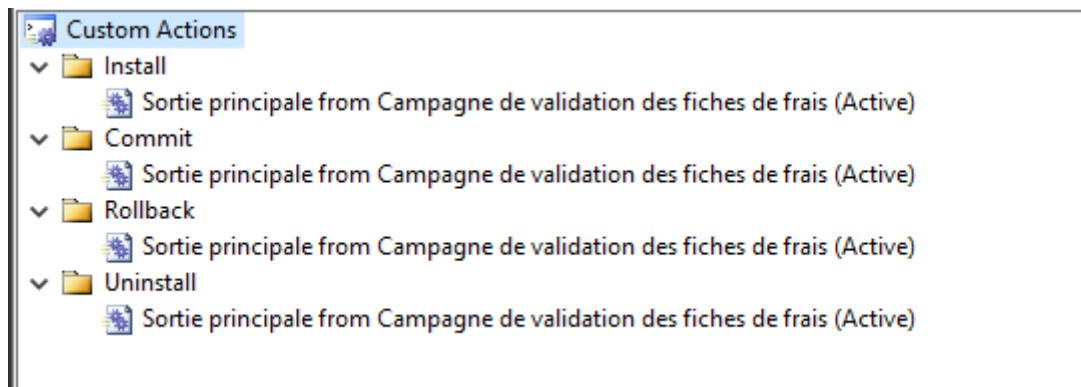
This is a banner for the 'Microsoft Visual Studio 2017 Installer' extension. It features a blue background with a white box on the left containing a yellow box icon and the text 'Microsoft Visual Studio 2017 Install...'. Below this, it says 'This official Microsoft extension provides support for Visual Studio Installer Projects in V...'. On the right, there are two buttons: 'Désactiver' (Disable) and 'Désinstaller' (Uninstall).

Nous créons la solution d'installation

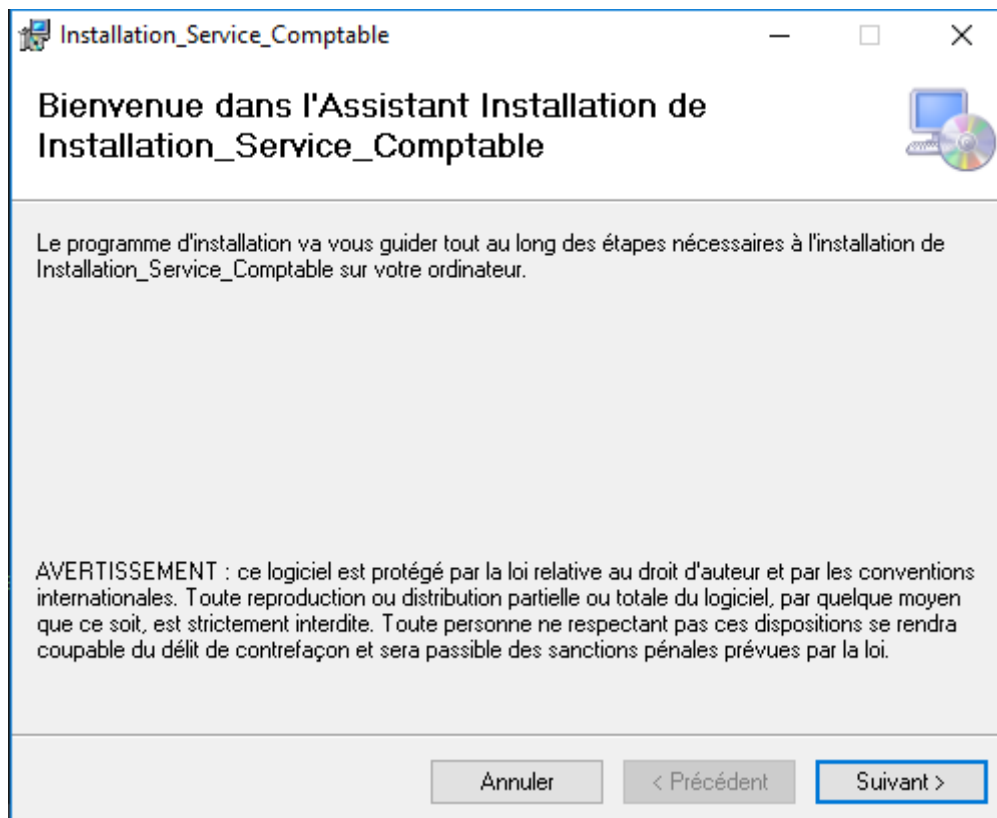


Nous définissons la sortie principale de la solution sur le projet à installer





Après avoir paramétré l'installateur nous pouvons lancer le programme d'installation



Nous constatons ici que le service est bien installé et bien lancé

