Livrable 2

EasySave 2.0

Chef de groupe : Milan Sangare

Eugénie Hariniaina

Alexandre De Jesus Correia

Pierre Grossin

Date : 29/11/2022

Ecole d'ingénieur du CESI

Etudiant en Première année du cycle ingénieur informatique

93 Bd de la Seine, 92000 Nanterre

CESI
ÉCOLE D'INGÉNIEURS

CESI
ÉCOLE D'INGÉNIEURS

Sommaire

CESI
ÉCOLE D'INGÉNIEURS

Introduction

Suite à une enquête client, la direction nous demande d'améliorer le logiciel. Les exigences pour la version 2.0 concernent la mise en place d'une interface graphique, le retrait de la limite de sauvegarde et utiliser un logiciel de cryptage de l'entreprise CryptoSoft.

Version 1.1: Fichier log en JSON ou XML

Pour pouvoir répondre aux demandes de nos plus grands clients, une version EasySave 1.1 est exigée : il faut permettre le choix du format des fichiers Log (journalier).

En particulier, ces derniers sont actuellement seulement disponibles au format JSON. Il faut ajouter un nouveau format selon la demande exigée : le format XML. Bien évidemment, cette fonctionnalité sera gardée pour la version 2.0.

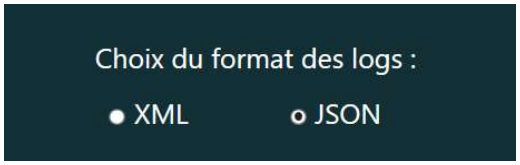
Dans notre ViewModel, on aura alors une mise à jour de la partie concernant les logs (pour la version 2.0, elle sera appelée "LoggingViewModel" pour plus de clarté) :

```
if (Logs.getInstance().logType == "json")
{
    var test = new
    {
        Name = name,
        Source = source,
        Target = target,
        Progress = new
        {
            sizeFiles = size + " bytes",
            startTime = dtDateTime,
            TransfertTime = transferTime + " ms",
            CryptTime = cryptTime+" ms",
        },
        // CryptTime = cryptTime
    };


    string json = JsonConvert.SerializeObject(test, Newtonsoft.Json.Formatting.Indented);



    File.AppendAllText(path, json + ",");
```

```
}
else if (Logs.getInstance().logType == "xml")
{
    var settings = new System.Xml.XmlWriterSettings
    {
        OmitXmlDeclaration = true,
        Indent = true
    };
    using (XmlWriter writer = XmlWriter.Create(path, settings))
    {
        writer.WriteStartElement("Save");
        writer.WriteAttributeString("Name", name);
        writer.WriteElementString("SourceFolder", source);
        writer.WriteElementString("TargetFolder", target);
        writer.WriteStartElement("Progress");
        writer.WriteElementString("sizeFiles", size + " bytes");
        writer.WriteElementString("startTime", XmlConvert.ToString(dtDateTime)); ;
        writer.WriteElementString("TransfertTime", transferTime + " ms");
        writer.WriteEndElement();
    }
```

Cette fonctionnalité sera aussi disponible dans la future interface graphique (View):

```
private void RadioButton_Checked(object sender, RoutedEventArgs e)
{
    if (json.IsChecked == true)
    {
        app.models.Logs.getInstance("json");
    }

    else if (xml.IsChecked == true)
    {
        app.models.Logs.getInstance("xml");
```



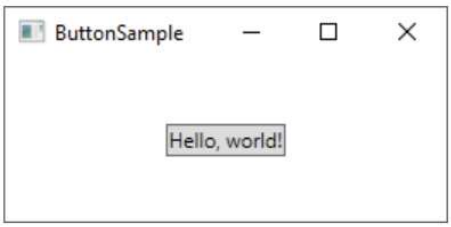
Interface graphique

Contrairement à la première version du livrable ou l'application devait s'afficher sur une Console, on souhaite maintenant le faire apparaître en format WPF, sous .Net Core. Pour rappel, Windows Presentation Foundation (WPF) est la spécification graphique de Microsoft .NET 3.0. Son principal composant est le langage descriptif XAML.

WPF peut fournir tous les éléments d'interface graphique comme les fenêtres, les boutons, les champs de texte, les menus, les listes, etc. La description de l'interface se faisant bien sûr en XAML.

Exemple de configuration d'un bouton en XAML :

```
<Button Name="Test" Background="White" Foreground="Black" Click="Create_Click">
    Hello, World!
</Button>
```

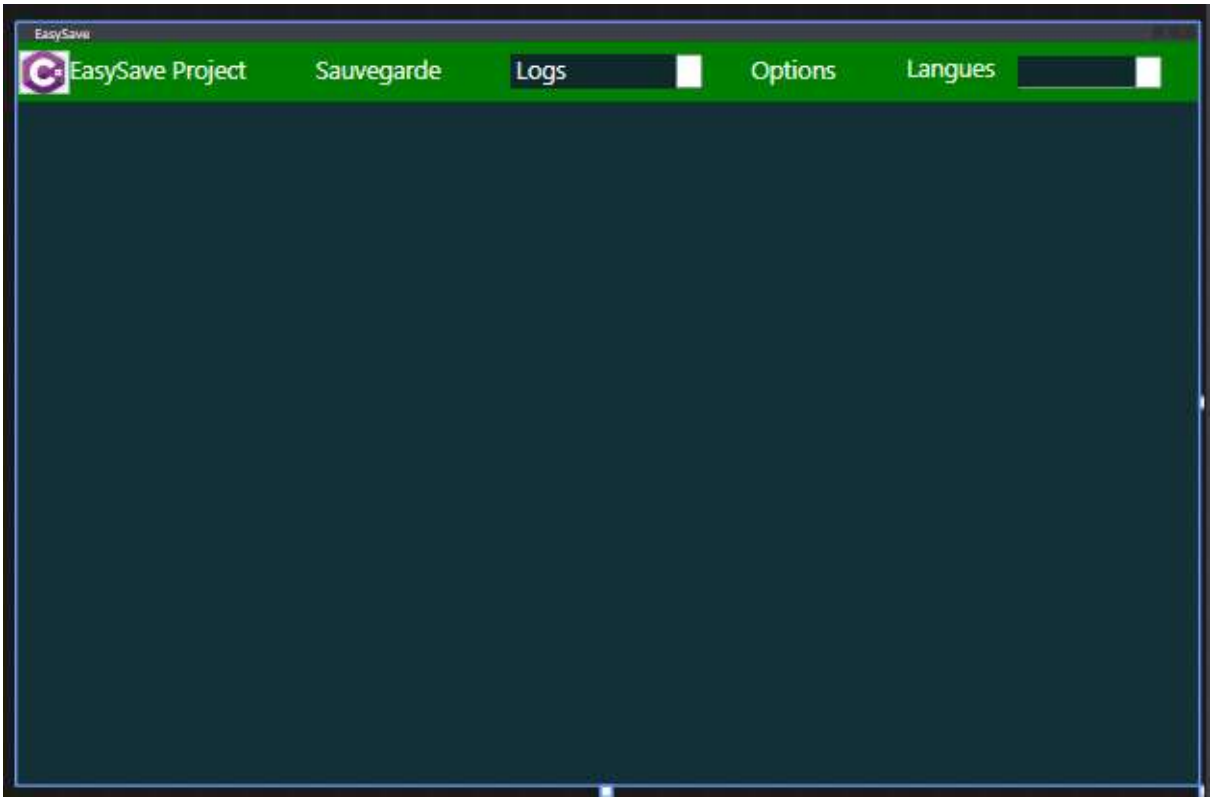


Ici, on a par exemple un bouton qui, lorsqu'il sera cliqué, déclenchera la fonction "Create_Click"

De plus WPF fournit aux développeurs différents moyens de créer leurs propres composants, par agrégation (UserControl) ou dérivation (Custom Control) de composants existants.

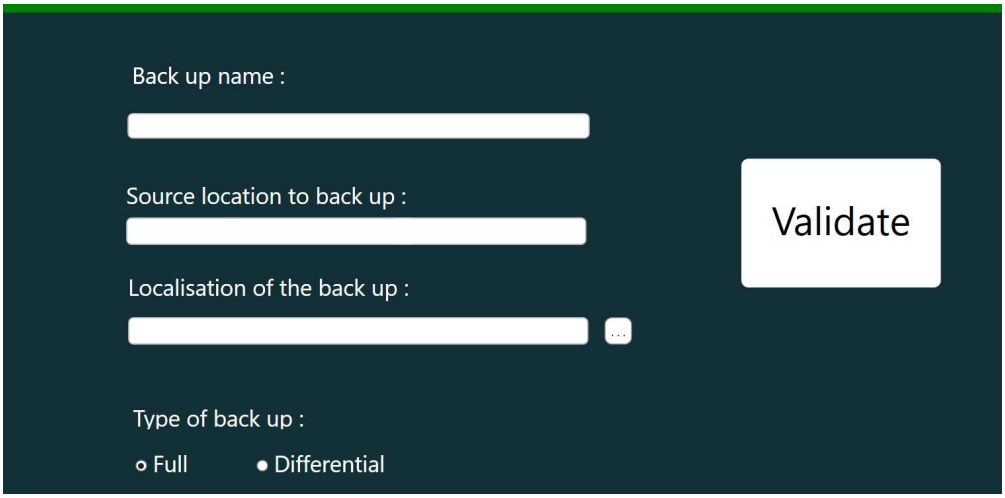
L'une des particularités de WPF est de dissocier le contrôle, au sens "composant" du terme (Entrées/Sorties, Évènements, etc.) de son graphisme. De fait, pour un contrôle donné, créer ou remplacer le graphisme (au sens large du terme, c.a.d en incluant les animations, les sons, etc.) se fait de manière particulièrement aisée. On parle alors de "Template" de Control.

Pour l'interface graphique de ce livrable, l'équipe a décidé de créer une View principale appelée MainWindow.xaml dans laquelle sera paramétrée la barre des tâches ainsi que les dimensions de la fenêtre WPF :



Ensuite, d'autres fichiers View ont été créés afin de permettre la création d'une sauvegarde, l'affichage de celles-ci ainsi que les réglages du logiciel de cryptage CryptoSoft.

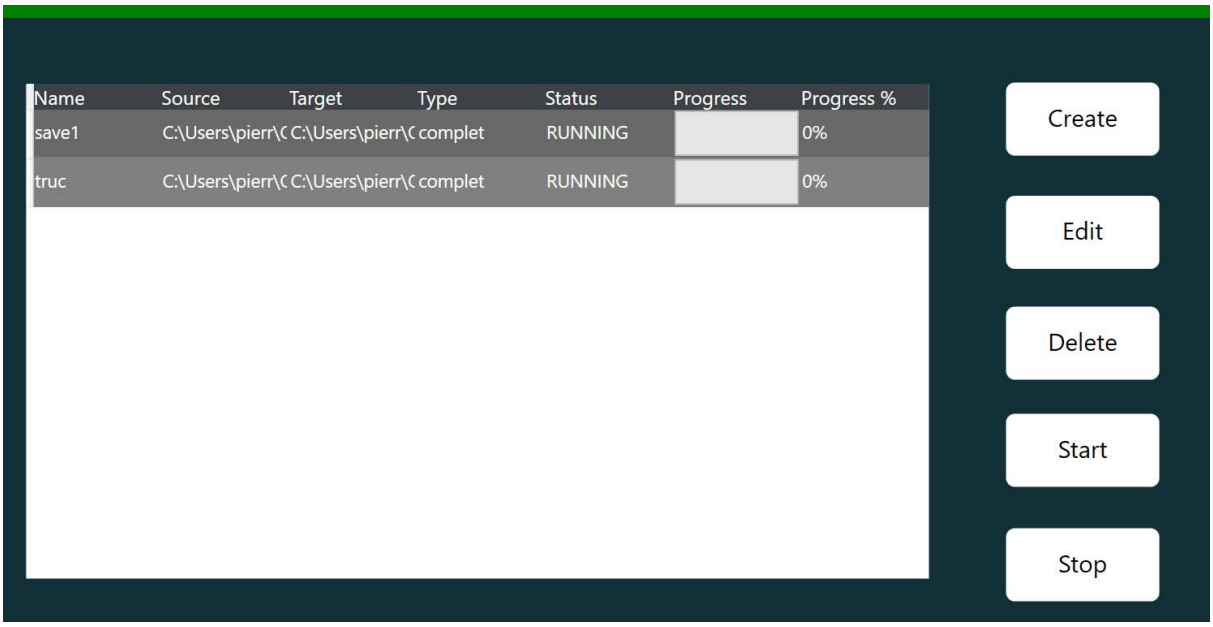
Fenêtre de création d'une sauvegarde :



Pour chaque sauvegarde, on devra bien sûr préciser:

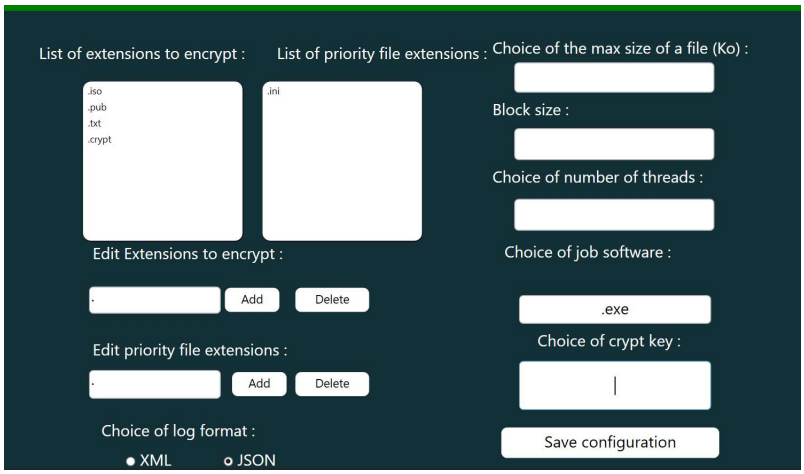
- le nom de celle-ci
- l'emplacement de la source à sauvegarder
- l'Emplacement de la sauvegarde
- le type de sauvegarde (complète ou différentielle)

Fenêtre d'affichage des sauvegardes :

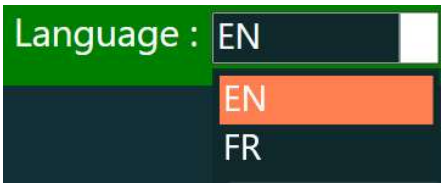


Pour chaque sauvegarde, on peut la modifier, la supprimer, l'activer ou stopper la sauvegarde des fichiers.

Fenêtre de la gestion de CryptoSoft



Enfin, l'application donne la possibilité de changer de langue très facilement, notamment grâce à 2 fichiers EN.json et FR.json contenant tous les champs de texte en anglais et en français :



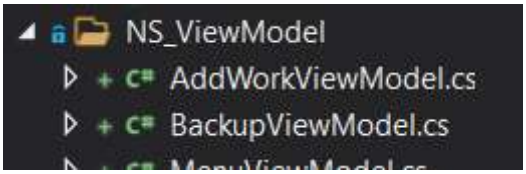

```
{
  "slugManageSave": "Sauvegarde",
  "slugLanguage": "Langue :",
  "slugSettings": "Reglages",
  "slugChoiceOfJobSoftware": "Choix du logiciel metier :",
  "slugChoiceOfExtensions": "Liste des entensions a chiffrer :",
  "slugSaveStatus": "Statut",
  "slugSaveName": "Nom",
  "slugSaveSource": "Source",
  "slugSaveTarget": "Destination",
  "slugSaveType": "Type",
  "slugCreateSave": "Creer",
  "slugEditSave": "Modifier",
  "slugDeleteSave": "Supprimer",
  "slugStartSave": "Lancer",
  "slugStopSave": "Arreter",
  "slugSavingName": "Nom de la sauvegarde :",
```

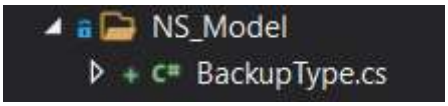
```
{
  "slugManageSave": "Back up",
  "slugLanguage": "Language :",
  "slugSettings": "Settings",
  "slugChoiceOfJobSoftware": "Choice of job software :",
  "slugChoiceOfExtensions": "List of extensions to encrypt :",
  "slugSaveStatus": "Status",
  "slugSaveName": "Name",
  "slugSaveSource": "Source",
  "slugSaveTarget": "Target",
  "slugSaveType": "Type",
  "slugCreateSave": "Create",
  "slugEditSave": "Edit",
  "slugDeleteSave": "Delete",
  "slugStartSave": "Start",
  "slugStopSave": "Stop",
  "slugSavingName": "Back up name :",
```

Retrait de la limite de sauvegarde

À part l'abandon du mode console et donc l'implémentation de l'interface graphique, le nombre de travaux à effectuer n'est plus limité à 5. Le client souhaite que le nombre de travaux effectués soit illimité.

Model et ViewModel contenant les codes initialisant un travail de sauvegarde et gère donc le nombre de travaux à faire :





- AddWorkViewModel.cs : contenant l'attribut "model", le constructeur AddWorkViewModel() et la méthode AddWork qui sera appelé autant de fois qu'il le faut pour ajouter un travail de sauvegarde :

```
// ----- Attributes -----
5 références
public Model model { get; set; }


// ----- Constructor -----
1 référence
public AddWorkViewModel(Model _model)
{
    this.model = _model;
}

1 référence
public void AddWork(string _name, string _src, string _dst, BackupType _backupType, bool _isCrypted)
{
    try
    {
        // Add Work in the program (at the end of the List)
        this.model.works.Add(new Work(_name, _src, _dst, _backupType, _isCrypted));
        this.model.SaveWorks();
    }
    catch
    {
        // Return Error Code
        model.errorMsg?.Invoke("errorAddWork");
    }
```

- BackupViewModel.cs : contenant plusieurs méthodes de manipulation sur les travaux de sauvegarde à faire mais ce qui nous intéresse est celle qui sera appelée pour lancer une sauvegarde :

La méthode **LaunchBackupWork**() qui reçoit en paramètre un entier qui est l'identifiant du travail à sauvegarder :

```
public void LaunchBackupWork(int[] idWorkToSave)
{
    if (this.model.works.Count > 0 && idWorkToSave.Length > 0)
    {
        for (int i = 0; i < idWorkToSave.Length; i++)
        {
            // Get id of one Running Business Software from the List (-1 if none)
            foreach (string businessSoftware in this.model.settings.businessSoftwares)
            {
                if (Process.GetProcessesByName(businessSoftware).Length > 0)
                {
                    for (int j = i; j < idWorkToSave.Length; j++)
                    {
                        currentBackupInfo?.Invoke("error", 0, 0, 0);
                    }
                    // Return Error Code
                    model.errorMsg?.Invoke("businessSoftwareOn");

                    return;
                }
            }
            LaunchBackupType(this.model.works[idWorkToSave[i]]);
        }
    }
    else
    {
        // Return Error Code
        model.errorMsg?.Invoke("noSelectedWork");
    }
}
```

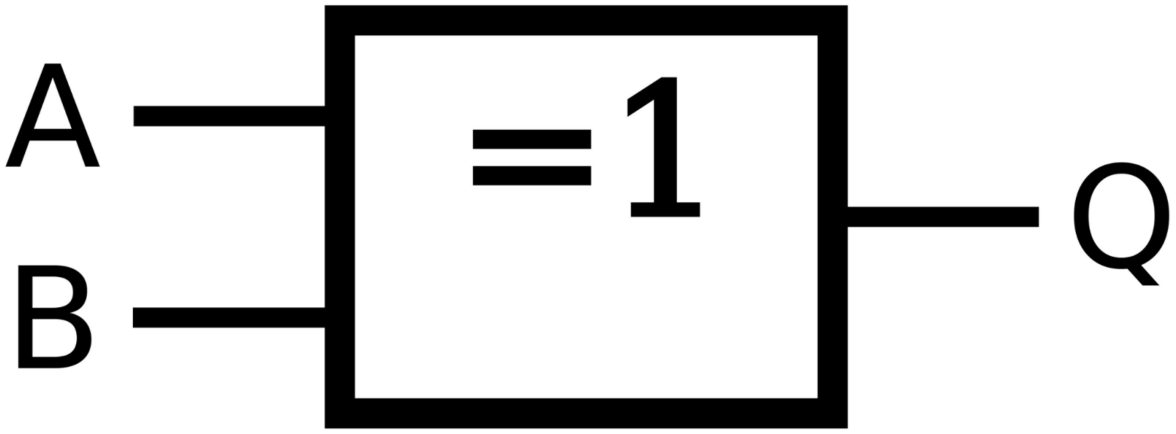
- BackupType.cs : contenant l'énumération des 2 types de backup à faire à savoir : complet ou différentiel

```
namespace EasySave.NS_Model
{
    6 références
    public enum BackupType
    {
        FULL,
        DIFFRENTIAL
    }
}
```

Cryptosoft

Dans le cahier des charges il est imposé que le logiciel devra être capable de crypter les fichiers en utilisant le logiciel CryptoSoft. Seuls les fichiers dont les extensions ont été définies par l'utilisateur dans les paramètres généraux devront être cryptés.

Après quelques recherches, on opte pour un système de cryptage, plutôt simple, le cryptage XOR.



Le cryptage XOR utilise l'opérateur binaire XOR (Ou Exclusif, symbolisé par ⊕) et comme opérandes le texte clair et la clé (préalablement encodés en binaire). XOR s'applique sur des données binaires, une conversion ASCII doit être réalisée sur un texte non binaire..C'est un système de cryptage basique beaucoup été utilisé dans les débuts de l'informatique car il est facile à implémenter.

Commençons par nous établir les dossier source et destination :

```
namespace CryptoSoft
{
    class Program
    {
        static int Main(string[] args)
        {
            int ArgLen = args.Length;
            string source = "";
            string destination = "";

            // seting the source and destination of the files
            for (int i = 0; i < ArgLen; i++)
            {
                if (args[i] == "source" && i + 1 < ArgLen)
                {
                    source = args[i + 1];
                    i++;
                }
                else if (args[i] == "destination" && i + 1 < ArgLen)
                {
                    destination = args[i + 1];
                    i++;
                }
            }
```

puis nous vérifions que le fichier à crypter existe et que les sources et destinations sont bien remplient :

```
// Looking if th destination and the source have been filled
if (source.Length == 0 || destination.Length == 0)
{
    Console.WriteLine("Missing arguments, either the source or the destination is empty.");
    return -1;
}

// Looking if the file we need to crypt exist
else if (!File.Exists(source))
{
    Console.WriteLine("Source file doesn't exist.");
    return -1;
}
```

nous pouvons donc maintenant commencer les opérations de cryptage :

```
try
{
    DateTime startTimeFile = DateTime.Now;

    //we convert the file we need to crypt to bytes and create a the file that will be crypted with the key
    byte[] to_crypt = File.ReadAllBytes(source);
    byte[] crypted = new byte[to_crypt.Length];

    //we set a key by default
    byte[] byteKey = new byte[8] { 9, 122, 23, 35, 75, 42, 166, 200 };
    BitArray bitKey = new BitArray(byteKey);

    //we crypting the file
    for (int i = 0; i < to_crypt.Length; i++)
    {
        crypted[i] = (byte)(to_crypt[i] ^ byteKey[i % key.Length]);
    }

    // we send the file to the selected destination
    File.WriteAllBytes(destination, crypted);

    TimeSpan cryptTime = DateTime.Now - startTimeFile;
    return (int)cryptTime.TotalMilliseconds;

}
catch
{
    Console.WriteLine("Cannot crypt this file.");
    return -1;
}
```

Le XOR se sert de l'ASCII pour effectuer sont chiffrement, on transforme les chaine de caractère en chaîne binaire puis on effectue une opération OU Exclusif, XOR, avec la clef définie.

Pour implémenter cela nous avons donc une clef en binaire et une conversion du document à chiffrer en binaire.

Voici comment nous appelons notre logiciel CryptoSoft dans EasySave :

```
public static void startCryptoSoft(string key, string path)
{
    int cryptedTime;
    try
    {   /// we are starting cryptosoft with the right arg for it to run
        ProcessStartInfo startInfo = new ProcessStartInfo("../../../src/CryptoSoft.exe");
        startInfo.Arguments = $"{key} {path} ";
        var CS_process = Process.Start(startInfo);
        CS_process.WaitForExit();
        cryptedTime = CS_process.ExitCode;
    }
    catch
    {
        // Return Error Code
        cryptedTime = -1;
        ///penser a ajouter un message d'erreur
        ///Invoke("cryptoSoftPathError, please put the full path of the CryptoSoft.exe ");
    }
}
```

On crée donc un nouveau processus pour utiliser le logiciel en bloquant EasySave le temps du cryptage.

Conclusion

Nous pouvons donc affirmer que le but principal du livrable est d'implémenter une interface graphique à notre logiciel avec les boutons, les champs de texte et tous les éléments nécessaires à une application de sauvegarde. Mais également une évolution sur le nombre de travail de sauvegarde qui n'est plus limité. La version 2.0 inclut donc les anciennes fonctionnalités de la version 1.0 et de nouvelles qui ont été détaillées dans ce livrable.

Bilan individuel

Milan SANGARE : Pour la réalisation de ce livrable je devais réaliser le système de cryptage, et j'ai aussi aidé les autres à réaliser certains aspects de leurs parties.

Pierre Grossin :
Au cours de ce livrable, j'ai réalisé la partie interface graphique du livrable. J'ai aussi aidé à la réalisation des autres parties.

Eugénie Hariniaina :
Tout au long de ce projet, j'étais responsable de la mise en place du système de versionning sur git et du déroulement des travaux de sauvegarde. J'ai également aidé à la réalisation des autres parties du projet.

Alexandre De Jesus Correia :
Au cours de ce projet, je me suis chargé du livrable 1.1, correspondant à l'implémentation du choix du format des fichiers logs journaliers (JSON ou XML). De plus, j'ai aussi participé aux autres parties également.

Bibliographie

Cryptographie :

https://www.dcode.fr/chiffre-xor

Travail de sauvegarde :

Model-View-ViewModel | Microsoft Learn

Delegates - C# Programming Guide | Microsoft Learn

A Simple WPF Application Implementing MVVM (c-sharpcorner.com)