

Livrable 1.0

EasySave version 1.0



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

Sommaire

Introduction	4
Diversité linguistique du logiciel	6
Historique des actions de travaux de sauvegarde	14
Conclusion	16
Bibliographie	16
Bilan personnel	17

Introduction

Notre équipe vient d'être embauchée par ProSoft et nous devons maintenant produire la version 1.0 en mode console du logiciel EasySave.

Les exigences pour la version 1.0 sont les suivantes :

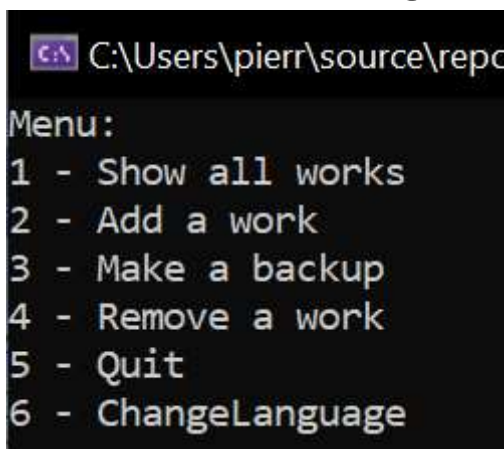
- Le logiciel doit permettre de créer jusqu'à 5 travaux de sauvegarde
- Le logiciel doit être utilisable à minima par des utilisateurs anglophones et Francophones
- L'utilisateur peut demander l'exécution d'un des travaux de sauvegarde ou l'exécution séquentielle de l'ensemble des travaux.
- Fichier Log journalier
- Le logiciel doit enregistrer en temps réel, dans un fichier unique, l'état d'avancement des travaux de sauvegarde.
- Les emplacements des deux fichiers (log journalier et état) devront être étudiés pour fonctionner sur les serveurs des clients.
- Les fichiers (log journalier et état) et les éventuels fichiers de configuration seront au format JSON.

Diversité linguistique du logiciel

Avant de nous intéresser à la partie technique, il faut établir certaines conventions. Ces conventions permettront un passage d'une langue à une autre de manière compréhensible par l'utilisateur.

La première convention est que l'utilisateur peut toujours choisir la langue qu'il souhaite parmi toutes celles disponibles. Le nom d'une langue est toujours dans la langue elle-même, jamais dans celle affichée, car dans le cas de figure où nous sommes, par exemple, en chinois, nous ne pourrions pas identifier quelle langue correspond au français. On utilise jamais de drapeau pour représenter une langue, tout simplement car un drapeau représente un pays et non une langue, exemple le français qui est parlé dans plusieurs pays.

Pour assurer la diversité linguistique du logiciel, il a été décidé de faire au plus simple. Dans le menu principal, l'option numéro 6 permet de passer automatiquement de l'anglais au français et vice-versa :

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Users\pierr\source\repo'. The prompt is 'C:\>'. Below the prompt, the word 'Menu:' is displayed. A list of six options follows: '1 - Show all works', '2 - Add a work', '3 - Make a backup', '4 - Remove a work', '5 - Quit', and '6 - ChangeLanguage'.

```
C:\>
Menu:
1 - Show all works
2 - Add a work
3 - Make a backup
4 - Remove a work
5 - Quit
6 - ChangeLanguage
```

Pour cela, une variable Language a été déclarée :

```
// --- Attributes ---
public Model model;
public View view;
public int Language = 0;
```

Cette variable pourra ensuite être modifiée en sélectionnant l'option ChangeLanguage, permettant ainsi le changement de langue. Il est à noté

que pour la version 1.1, le changement de langue n'a été paramétré que pour le menu principal :

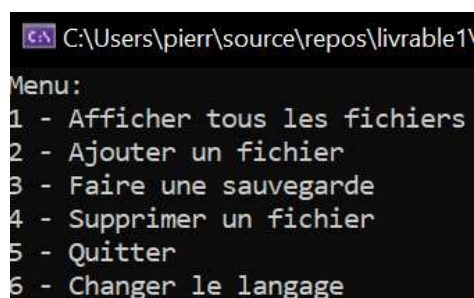
case 6:

```
if (Language == 1)
{
    int Language = 0;
}
else if (Language == 0)
{
    int Language = 1;
}

break;

if (Language == 1)
{
    Console.Clear();
    Console.WriteLine(
        "Menu:" +
        "\n1 - Afficher tous les fichiers" +
        "\n2 - Ajouter un fichier" +
        "\n3 - Faire une sauvegarde" +
        "\n4 - Supprimer un fichier" +
        "\n5 - Quitter" +
        "\n6 - Changer le langage");
}

else if (Language == 0)
{
    Console.Clear();
    Console.WriteLine(
        "Menu:" +
        "\n1 - Show all works" +
        "\n2 - Add a work" +
        "\n3 - Make a backup" +
        "\n4 - Remove a work" +
        "\n5 - Quit" +
        "\n6 - ChangeLanguage");
}
```



C:\Users\pierr\source\repos\livrable1\>

Menu:

- 1 - Afficher tous les fichiers
- 2 - Ajouter un fichier
- 3 - Faire une sauvegarde
- 4 - Supprimer un fichier
- 5 - Quitter
- 6 - Changer le langage

Déroulement des travaux de sauvegarde

Le premier but de notre projet est de créer un logiciel de sauvegarde de fichiers en temps réel d'un fichier source vers un fichier cible.

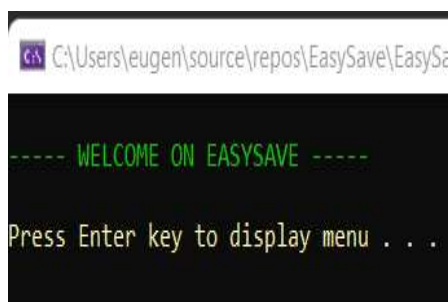
Pour réaliser cela, nous devons respecter les spécificités établies par le client à propos des travaux de sauvegarde à faire.

Ces spécificités sont les suivantes :

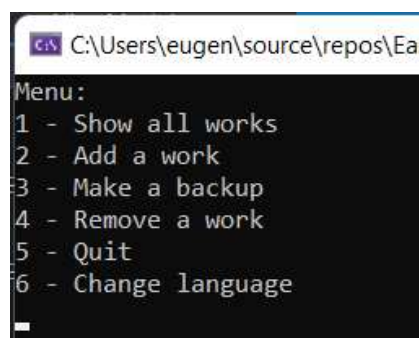
- Le logiciel doit permettre de créer jusqu'à 5 travaux de sauvegarde
- Un travail de sauvegarde est défini par :
 - Une appellation
 - Un répertoire source
 - Un répertoire cible
 - Un type (complet, différentiel)
- L'utilisateur peut demander l'exécution d'un des travaux de sauvegarde ou l'exécution séquentielle de l'ensemble des travaux

Pour cette version du projet, nous avons développé un logiciel en mode console avec Visual Studio 2019.

En lançant le débogueur, l'accueil s'affiche. Après avoir appuyé sur "Enter", le menu s'affiche :



```
C:\Users\eugen\source\repos\EasySave\EasySave>
----- WELCOME ON EASYSAVE -----
Press Enter key to display menu . . .
```



```
C:\Users\eugen\source\repos\Ea
Menu:
1 - Show all works
2 - Add a work
3 - Make a backup
4 - Remove a work
5 - Quit
6 - Change language
```

- Show all works :

Le choix 1 affiche les travaux de sauvegarde existant. En cliquant sur "1" le console affiche un message disant que c'est vide. Ce message est géré dans la méthode "ConsoleUpdate" qui gère les affichages d'informations sur le console; que ce soit une information sur ce qu'il faut taper ou un message d'erreur :

Codes :

```
public void ConsoleUpdate(int _id)
{
    if (_id < 100)
    {
        Console.ForegroundColor = ConsoleColor.Yellow;
        switch (_id)
        {
            //Information message
            case 1:
                Console.WriteLine("\nPress Enter key to display menu . . .");
                Console.ReadLine();
                break;
```

```
            case 204:
                Console.WriteLine("\nWork List is empty.");
                ConsoleUpdate(1);
                break;
```

Console :

```
C:\Users\eugen\source\repos\EasySave\EasySave>
Menu:
1 - Show all works
2 - Add a work
3 - Make a backup
4 - Remove a work
5 - Quit
6 - Change language
1

Work List is empty.

Press Enter key to display menu . . .
```


- Add a work :

Taper sur "Entrer" puis sur "2" va appeler

- une méthode pour inviter à entrer le nom du travail de sauvegarde
- une méthode pour entrer le répertoire source
- une méthode pour entrer le répertoire cible
- une méthode pour choisir le type de sauvegarde à faire si c'est "complet"(full) ou "différentiel"(differential)

Codes :

```
//Add work name
1 référence
public string AddWorkName()
{
    Console.Clear();
    Console.WriteLine("Parameter to add a work:");
    ConsoleUpdate(2);

    Console.WriteLine("\nEnter a name (1 to 20 characters):");
    string name = Console.ReadLine();

    //Check if the name is valid
    while (!CheckName(name))
    {
        name = Console.ReadLine();
    }
    return name;
}
```

```
//Add work source
1 référence
public string AddWorkSrc()
{
    Console.WriteLine("\nEnter directory source. ");
    string src = RectifyPath(Console.ReadLine());

    //Check if the path is valid
    while (!Directory.Exists(src) && src != "")
    {
        ConsoleUpdate(211);
        src = RectifyPath(Console.ReadLine());
    }
    return src;
}
```

```
//Add work destination
1 référence
public string AddWorkDst(string _src)
{
    Console.WriteLine("\nEnter directory destination.");
    string dst = RectifyPath(Console.ReadLine());

    //Check if the path is valid
    while (!CheckWorkDst(_src, dst))
    {
        dst = RectifyPath(Console.ReadLine());
    }
    return dst;
}
```

```
//Add work backup type
1 référence
public int AddWorkBackupType()
{
    Console.WriteLine(
        "\nChoose a type of Backup: " +
        "\n1.Full " +
        "\n2.Differential");
    return CheckChoiceMenu(Console.ReadLine(), 0, 2);
}
```

Message de validation :

```
public void ConsoleUpdate(int _id)
{
    if (_id < 100)
    {
        Console.ForegroundColor = ConsoleColor.Yellow;
        switch (_id)
        {
```

```
        else if (_id < 200)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            switch (_id)
            {
                // Success message from 100 to 199
                case 100:
                    Console.WriteLine("\n----- WELCOME ON EASYSAVE -----");
                    ConsoleUpdate(1);
                    break;

                case 101:
                    Console.WriteLine("\nThe work was added with success!");
                    ConsoleUpdate(1);
                    break;
```

Console :

```
C:\Users\eugen\source\repos\EasySave\EasySave\bin\Debug\net6.0\EasySave.exe
Parameter to add a work:

(Enter 0 to return to the menu)

Enter a name (1 to 20 characters):
Save1

Enter directory source.
C:\Users\eugen\OneDrive\Documents\EasySaveSource

Enter directory destination.
C:\Users\eugen\OneDrive\Documents\EasySaveCible

Choose a type of Backup:
1.Full
2.Differential
1

The work was added with success!

Press Enter key to display menu . . .
```

- Make a backup :

Taper “Entrer” puis choisir “3” : le mode console demande de choisir entre exécuter tout ou sélectionner un travail de sauvegarde.

Codes :

```
//Choose the work to save
1 référence
public int LaunchBackupChoice()
{
    Console.Clear();
    Console.WriteLine(
        "Choose the work to save : " +
        "\n\n1 - all");

    //Display all works
    LoadWorks(2);
    ConsoleUpdate(2);

    //Check if the user's input is a valid integer
    return CheckChoiceMenu(Console.ReadLine(), 0, this.viewModel.model.works.Count + 1);
}
```

```
public void ConsoleUpdate(int _id)
{
    if (_id < 100)
    {
        Console.ForegroundColor = ConsoleColor.Yellow;
        switch (_id)
        {
            //Information message
            case 1:
                Console.WriteLine("\nPress Enter key to display menu . . .");
                Console.ReadLine();
                break;

            case 2:
                Console.WriteLine("\nEnter 0 to return to the menu");
                break;

            case 3:
                Console.Clear();
                Console.WriteLine("\nBackup information :");
                break;

            case 4:
                Console.WriteLine("\nPress Enter key to show more . . .");
                Console.ReadLine();
                break;
        }
    }
}
```

```
2 références
public void DisplayBackupRecap(string _name, double _transferTime)
{
    Console.WriteLine("\n\n" +
        "Backup : " + _name + " finished\n"
        + "\nTime taken : " + _transferTime + " ms\n");
    DisplayProgressBar(100);
}
```

```
//Display message on the console
37 références
public void ConsoleUpdate(int _id)
{
    if (_id < 100)
    {
```

```
case 104:
    Console.WriteLine("\nBackup success !");
    break;
```

Console :

```
C:\Users\eugen\source\repos\EasySave\EasySave\bin\Debug\netcoreapp3.1\EasySave.exe
Choose the work to save :
1 - all
2 - Name: Save1
   Source: c:\users\eugen\onedrive\documents\easyssavesource\
   Destination: c:\users\eugen\onedrive\documents\easyssavecible\
   Type: FULL
(Enter 0 to return to the menu)
```

```
C:\Users\eugen\source\repos\EasySave\EasySave\b
Backup information :
Backup : Save1 finished
Time taken : 131,8779 ms
Progress: [ 100%] [#####]
Backup success !
Press Enter key to show more . . .
```

- Show all works :

En tapant "Entrer" pour afficher le menu puis "1", cela va appeler une méthode qui affichera la liste des travaux existants donc "Save1" pour notre cas actuel :

Codes :

```
1 référence
public void DisplayWorks()
{
    Console.Clear();
    Console.WriteLine("Work list :");

    //Display all works
    LoadWorks(1);
    ConsoleUpdate(1);
}
```

Console :

```
C:\Users\eugen\source\repos\EasySave\EasySave\bin\Debug\netcoreapp3.1\EasySav
Work list :
1 - Name: Save1
  Source: c:\users\eugen\onedrive\documents\easysavesource\
  Destination: c:\users\eugen\onedrive\documents\easysavecible\
  Type: FULL
Press Enter key to display menu . . .
```

- Remove a work :

Taper "Entrer" pour accéder à nouveau au menu puis "4" pour appeler la méthode qui demande de choisir le travail à supprimer :

Codes :

```
//Choose the work to remove
1 référence
public int RemoveWorkChoice()
{
    Console.Clear();
    Console.WriteLine("Choose the work to remove :");

    //Display all works
    LoadWorks(1);
    ConsoleUpdate(2);

    //Check if the user's input is a valid integer
    return CheckChoiceMenu(Console.ReadLine(), 0, this.viewModel.model.works.Count);
}
```

```
public void ConsoleUpdate(int _id)
{
    if (_id < 100)
    {
        Console.WriteLine("1 - Save1\n2 - Save2\n3 - Save3\n4 - Save4\n5 - Save5\n6 - Save6\n7 - Save7\n8 - Save8\n9 - Save9\n0 - Exit");
    }
}
```

```
case 103:
    Console.WriteLine("\nThe work was removed with success!");
    ConsoleUpdate(1);
    break;
```

Console :

```
C:\Users\eugen\source\repos\EasySave\EasySave\bin\Debug\netcoreapp3.1\EasySave.
Choose the work to remove :

1 - Name: Save1
  Source: c:\users\eugen\onedrive\documents\easysavesource\
  Destination: c:\users\eugen\onedrive\documents\easysavecible\
  Type: FULL

(Enter 0 to return to the menu)
1

The work was removed with success!

Press Enter key to display menu . . .
```

Pour vérifier que le travail a bien été supprimé, on va taper “Entrer” puis “1” pour “Show all works” et normalement ça donne un message en rouge disant qu’il n’y a aucun travail :

Console :

```
C:\Users\eugen\source\repos\EasySave\EasySave\bi
Menu:
1 - Show all works
2 - Add a work
3 - Make a backup
4 - Remove a work
5 - Quit
6 - Change language
1

Work List is empty.

Press Enter key to display menu . . .
```


Historique des actions de travaux de sauvegarde

Afin de connaître en temps réel les différentes sauvegardes déjà réalisées, on pourra créer un fichier log, c'est-à-dire un fichier comprenant les informations (logs) enregistrées au niveau des serveurs web lorsqu'une requête de chargement de fichiers est effectuée par un navigateur. Dans notre cas, les informations correspondent à celles liées à une action d'un travail de sauvegarde.

Pour chaque sauvegarde, les informations suivantes sont donc précisées :

- Horodatage
- Appellation du travail de sauvegarde
- Adresse complète du fichier Source (format UNC)
- Adresse complète du fichier de destination (format UNC)
- Taille du fichier
- Temps de transfert du fichier en ms (négatif si erreur)
- Etat de la sauvegarde (réussite ou échec)

On retrouvera d'ailleurs ces attributs dans la classe work de notre code :

```
// --- Attributes ---  
public string name { get; set; }  
public string src { get; set; }  
public string dst { get; set; }  
public BackupType backupType { get; set; }  
public State state { get; set; }  
public string lastBackupDate { get; set; }
```

En l'intégrant à notre architecture MVVM finale, il faudra bien sûr penser à la relier à la partie sur le fichier contenant l'état d'avancement des sauvegardes en cours, afin que, une fois un travail de sauvegarde terminé,

qu'il soit réussi ou non, les informations sur la sauvegarde soient enregistrées sur le fichier log.

L'enregistrement d'une sauvegarde sera géré par la méthode SaveLog :

```
public void SaveLog(DateTime _startDate, string _src, string _dst, long _size, bool isError)
{
    // Prepare times log
    string today = DateTime.Now.ToString("yyyy-MM-dd");
    string startTime = _startDate.ToString("yyyy-MM-dd_HH-mm-ss");
    string elapsedTime = (DateTime.Now - _startDate).ToString();

    if (isError)
    {
        elapsedTime = "-1";
    }

    // Create File if it doesn't exists
    if (!Directory.Exists("./Logs"))
    {
        Directory.CreateDirectory("./Logs");
    }

    // Write log
    File.AppendAllText($"./Logs/{today}.txt", $"{startTime}: {this.name}" +
        $"{'\nSource: {_src}'}" +
        $"{'\nDestination: {_dst}'}" +
        $"{'\nSize (Bytes): {_size}'}" +
        $"{'\nElapsed Time: {elapsedTime}'}" +
        $"{'\n\n'}");
}
```

Fichier de l'état d'avancement des travaux de sauvegarde

Afin de connaître l'état d'avancement des travaux de sauvegarde, on pourra créer un fichier (unique) qui enregistrera, pour chaque travail de sauvegarde, les informations correspondantes.

En particulier, le fichier contiendra :

- l'Appellation du travail de sauvegarde
- l'Horodatage
- l'État du travail de sauvegarde (actif ou non)

De plus, si le l'état est actif, il contiendra aussi :

- le nombre total de fichiers éligibles

- la taille des fichiers à transférer
- la progression
 - ❖ le nombre de fichiers restants
 - ❖ la taille des fichiers restants
 - ❖ l'adresse complète du fichier Source en cours de sauvegarde
 - ❖ l'adresse complète du fichier de destination

On a alors un code comme ci-dessous :

> Le fichier State sera produit au moment du lancement de la Backup

```
4 références
class State
{
    // --- Attributes ---
    1 référence
    public int totalFile { get; set; }
    1 référence
    public long totalSize { get; set; }
    2 références
    public int progress { get; set; }
    1 référence
    public int nbFileLeft { get; set; }
    1 référence
    public long leftSize { get; set; }
    2 références
    public string currentPathSrc { get; set; }
    2 références
    public string currentPathDest { get; set; }
}
```

> On crée les variables correspondants aux informations voulues

```
1 référence
public State(int _totalFile, long _totalSize, string _currentPathSrc, string _currentPathDest)
{
    this.progress = 0;
    this.totalFile = _totalFile;
    this.totalSize = _totalSize;
    this.currentPathSrc = _currentPathSrc;
    this.currentPathDest = _currentPathDest;
}
```

```
// Update State during DoBackup()
1 référence
public void UpdateState(int _progress, int _nbFileLeft, long _leftSize, string _currSrcPath, string _currDestPath)
{
    this.progress = _progress;
    this.nbFileLeft = _nbFileLeft;
    this.leftSize = _leftSize;
    this.currentPathSrc = _currSrcPath;
    this.currentPathDest = _currDestPath;
}
```

> On met à jour les valeurs au cours de la sauvegarde

Lorsque la sauvegarde est terminée, on obtiendra alors, par exemple, un résultat du type :

```
Backup information :  
  
Backup : YOYO finished  
Time taken : 17580,8468 ms  
Progress: [ 100%] [#####]  
  
Backup success !
```

Conclusion

Le cœur du livrable était donc de mettre au point notre toute première version du logiciel EasySave. La version 1.0 permet de créer des travaux de sauvegarde et de les effacer , tout en conservant un trace des actions effectuées et plus de nous renseigner sur l'avancement de la sauvegarde. Le logiciel s'est vu attribuer deux langues d'utilisation différentes.

Bibliographie

[Documentation .NET | Microsoft Learn](#)

[Gérez du code avec Git et GitHub - OpenClassrooms](#)

[Intégration Continue ou Continuous Integration : qu'est-ce que c'est ? \(lebigdata.fr\)](#)

[UML 2 - de l'apprentissage à la pratique \(developpez.com\)](#)

[Architecturer ses applications JS à l'aide du pattern MVVM | \(docdoku.com\)](#)

[The MVVM pattern – Introduction | Diary of a Windows developer \(qmatteoq.com\)](#)

[cs.pdf \(free.fr\)](#)

[Découvrez & formez-vous avec votre Coach C# | Microsoft Learn](#)

Bilan personnel

Milan Sangare : Mes tâches dans la réalisation de ce projet étaient de réaliser le diagramme UML avec mes camarades, ainsi que la mise en place de la diversité linguistique, et j'ai aussi aidé à la réalisation des autres parties.

Pierre Grossin :

Au cours de ce livrable, je me suis chargé de la partie sur l'historique des actions de travaux de sauvegarde ainsi que la réalisation du diagramme de classe. j'ai aussi aidé à la mise en place d'une diversité linguistique.

Eugénie Hariniaina :

Tout au long de ce projet, j'ai participé à la réalisation des diagrammes UML correspondant aux attributs et méthodes de l'application. J'ai implémenté en C# la partie sur les travaux de sauvegarde du logiciel et j'ai participé aux autres parties également.

Alexandre De Jesus Correia :

Au cours du projet, j'ai participé à la création des diagrammes UML, mais j'ai aussi implémenté la partie sur le fichier d'état d'avancement du travail de sauvegarde. J'ai aussi apporté de l'aide quand nécessaire à mes collègues sur leurs différentes parties.