

Documentation technique

EasySave V3.0

Introduction :

Suite à la demande de Prosoft, notre équipe de développement a réalisé *EasySave*, un logiciel de sauvegarde pour un de ses clients.

Le document suivant fait suite à ce projet avec la réalisation des différentes versions de EasySave, que ce soit en CLI ou en GUI.

Pour l'installation faite au début du projet avec les outils que l'on envisageait utiliser, il existe un autre document pour configurer l'environnement parmi les livrables; les outils finalement adoptés sont mentionnés un peu plus tard.

Sommaire :

Introduction :	1
Sommaire :	1
Rappel des fonctionnalités et avancement :	2
Avant de commencer le travail sur EasySave :	4
Architecture implémentée :	4
Logiques implémentées :	5
Catégorisation :	5
Multi Threading :	5
Copie complète :	6
Copie différentielle :	6
Gestion des langues :	6
Interface déportée :	6
Mock-up et amélioration pour la future version :	7
Améliorations techniques :	7

Rappel des fonctionnalités et avancement :

Ci-dessous un tableau permettant d'identifier les différentes fonctionnalités demandées ainsi que celle à être implémentée en priorité :

Fonctionnalité	V1.0	V1.1	V2.0	Version 3.0
Interface Graphique	Console	Console	WPF	WPF
Multi-langues	Support de l'anglais et du français			
Travaux de sauvegarde	Limité à 5		Illimité	
Fichier Log journalier	JSON	JSON/XML	JSON/XML + Temps chiffrement	
Pause de 1 ou + taches	Non			Oui
Fichier Etat (state.json)	Oui			
Fonctionnement sauvegarde	Mono ou séquentielle			Parallèle
Arrêt si détection du logiciel métier	Non		Oui (impossible de lancer un travail)	Oui (arrêt de tous les transferts en cours)
Utilisation de cryptosoft	Non		Oui	

Gestion de fichiers Prioritaires	Non		OUI, avec attente des autres tâches
Interdiction de sauvegardes simultanées pour les fichiers volumineux	Non		NON
Interface déportée	Non		Oui
Ligne de commande	Oui	identique version 1.0	
Application Mono-instance	Non		Oui
Surveillance charge Réseau	Non		Pas de réduction automatique des flux

Suite à ce tableau, certaines des fonctionnalités sont mises en rouge car elles n'ont pas été implémentées.

L'utilisation de CryptoSoft est mise en orange car il y a certains points à remettre à jour, notamment au niveau du chiffrement des fichiers contenant des espaces dans leurs noms.

Avant de commencer le travail sur EasySave :

Les outils utilisés pour la gestion de projet et de code sont Azure DevOps avec un dépôt Git contenant des branches pour du développement en parallèle.

Le logiciel a été programmé sur des OS Windows sous JetBrains Rider en général ainsi que Visual Studio 2019 et 2022 pour WPF et cela de manière complémentaire.

Ce qu'il faut faire avant d'entamer la collaboration :

- Bien connaître les concepts de la POO.
- Connaître les bases sur .NET Core et C#.
- Se renseigner sur l'architecture MVVM notamment le Data Binding et l'organisation du code.
- Comprendre les designs pattern et surtout Factory, Singleton, Observer et Strategy.

Les pré-requis pour commencer à collaborer :

- Un accès au projet hébergé sur Azure DevOps
- L'outil de gestion de versions Git
- .NET 5 et .NET Core 3.1
- Un IDE pour C# : Visual Studio 2019/2022 ou JetBrains Rider

Architecture implémentée :

La version 3.0 de EasySave est basée sur une architecture MVVM ainsi que la réutilisation de certaines classes dans un projet librairie qui était de prime abord présente dans la version 1.0, et qui a pour but de mettre en commun les classes pour ne pas avoir à dupliquer du code avec tous les soucis qu'il pourrait y avoir suite à ça. Pour la V3.0, voici les fichiers présents :

MVVM : Parties ViewModel + View

```
+---View
|   AddJobWindow.xaml
|   AddJobWindow.xaml.cs
|   EditJobWindow.xaml
|   EditJobWindow.xaml.cs
|   MainWindow.xaml
|   MainWindow.xaml.cs
|   Option.xaml
|   Option.xaml.cs
|
\---ViewModel
    AddJobViewModel.cs
    EditJobViewModel.cs
    JobsViewModel.cs
    LanguageSettingsViewModel.cs
    MainViewModel.cs
    ViewModelBase.cs
```

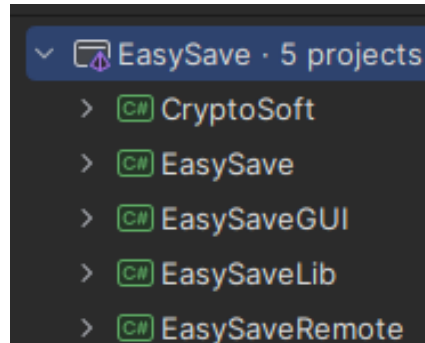
Classes auxiliaires

```
---Assets
    StringResources.en.xaml
    StringResources.fr.xaml
    StringsResources.en.json
    StringsResources.fr.json
---Command
    RelayCommand.cs
---Helper
    CopyController.cs
    CryptoSoftCipher.cs
    JsonResourceDictionary.cs
    ProcessBL.cs
    ProgressBar.cs
```

Projet Libraire : EasySaveLib

```
+---Model
|   ConfigManager.cs
|   IdentityManager.cs
|   Job.cs
|   Logger.cs
|   \---CopyHelper
|       FileGetter.cs
```

Solution contenant les projets



```
EasySave · 5 projects
> [C#] CryptoSoft
> [C#] EasySave
> [C#] EasySaveGUI
> [C#] EasySaveLib
> [C#] EasySaveRemote
```

Il y a suite à cette organisation des dépendances entre projet (utilisation de Model présent dans EasySaveLib dans les ViewModel de EasySaveGUI...)

Toutefois il faut savoir que le modèle MVVM n'est pas forcément appliqué sur toutes les parties du projet, mais il y a un bon exemple qui est situé au niveau du bouton de Modification de Travail de Sauvegarde.

En effet, avec cette fonctionnalité on utilise plusieurs éléments de cette architecture, notamment des *Commands* pour les boutons au lieu des clics. Le *Data Binding* avec une liaison entre les détails de l'élément à modifier sur la grille et l'objet qui le représente tout au long de l'exécution du programme. La séparation entre la Vue (les fenêtres) et le ViewModel qui lui même contient la logique de la modification du travail.

Logiques implémentées :

Catégorisation :

Au vu de la demande de gestion de fichiers prioritaires selon les extensions et la taille, on définit quatre listes de fichier : Priorité Grands – Priorité tout court – Grand – Autres

Ce choix facilitera par la suite la répartition de thread, par exemple Priorité Grands 1 Thr et Priorité 4 Thr (si NbFichiersPrioGrands <<< NbFichiersPrio ...)

Multi Threading :

On définit dans la classe CopyController le nombre de Threads alloué pour chaque travail de sauvegarde. Ensuite après une Catégorisation, on commence par répartir les threads. A chaque fois, il y a du multithreading sur les deux premières listes puis sur les deux dernières.

Il y a une gestion d'accès concurrentiels aux opérations de logs par des Locks et des Mutex.

Copie complète :

La destination est vidée grâce au premier thread puis est remplacée par les fichiers de la source avec gestion de priorité et de chiffrement. On calcule pour chaque fichier, un hachage qu'on stocke dans un fichier avec les Logs.

Copie différentielle :

Si la destination est vide, on effectue une sauvegarde complète. Sinon on utilise le fichier contenant les hachages et pour chaque fichier de la source, on calcule le hachage et on le compare avec celui déjà noté auparavant, s'il y a une différence on copie ce fichier, sinon on passe au suivant.

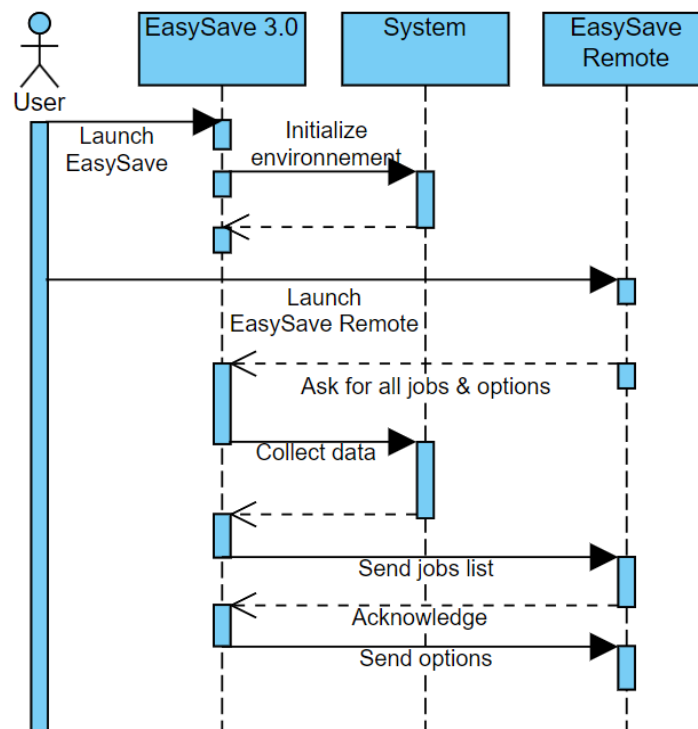
Gestion des langues :

On utilise des dictionnaires en JSON avec du Data Binding qui permet d'actualiser pendant l'exécution, les champs de texte.

Interface déportée :

Il s'agit d'une refonte de la V3.0 avec l'utilisation de balise de type "<|RJ|>" pour RemoveJob et qui de la part du client spécifie au serveur quelle opération est attendue. Elle doit être repensée pour être plus robuste et évolutive avec la version principale. Pour le moment, les données sont transférées en la V0.1, une amélioration du protocole utilisé, ainsi que de la trame est prévue en même temps que la gestion des flux de données.

Ci dessous se trouve un diagramme de séquence représentant le flux de données au moment de la connexion entre le serveur EasySave (EasySave 3.0), le serveur de fichier (System) et l'interface déportée (EasySave Remote).



Informations pertinentes :

Logs : Le dossier contenant les différents Logs (Logs quotidiens, fichier d'état, fichier de configuration) est créé lors du premier lancement de l'application à l'emplacement suivant : C:\Prosoft\EasySave. Un sous-dossier Logs est créé à l'intérieur, contenant les fichiers d'état ainsi que les journaux quotidiens.

Configuration minimale :

- Windows 7 ou ultérieur.
- Processeur x64 ou x86.
- Minimum 4 Go de RAM.
- Minimum 2 Go d'espace de stockage pour l'application et ses dépendances.

Assistance :

En cas de bug, de questionnement ou de tout autre problème, veuillez vous référer à la notice utilisateur. Si le problème persiste, n'hésitez pas à contacter notre support au 03 12 34 56 78.

Mock-up et amélioration pour la future version :

Language

New Job

Play

Delete

Edit






Play/Pause

Stop

Option

ON

OFF

Job Name	Backup Type	Source Path	Destination Path	Actions	Progression
Test	Full	C:\Source	C:\Destination	 	<div></div>
Test 2	Diff	C:\Source	C:\Destination	 	

EVENTS

On pourra noter notamment des améliorations au niveau l'expérience utilisateur avec plus de couleurs et l'addition d'un chargement pour l'opération de catégorisation qui peut prendre beaucoup plus de temps que prévu.

Améliorations prévue :

- **Catégorisation** : Utiliser des **tables de hachages** au lieu de listes pour optimiser le temps de calcul. Cela permettra l'utilisation d'une fonctionnalité de **comparaison en parallèle**, qui permet de passer par plusieurs conditions if() en même temps pour une itération et donc éviter la passage par quatres conditionnelles.
- **Compression** : La prochaine étape principale sera l'utilisation d'**algorithmes de compression** de fichier, pour pouvoir **réduire la taille** des fichiers sauvegardés sans en altérer le contenu.
- **Événement** : Concernant l'expérience utilisateur, l'ajout de l'affichage des **événements principaux** (comme le lancement d'un job, l'ajout ou autre) sur l'interface pourrait faciliter la compréhension et l'utilisation de l'application.
- **Langues** : L'ajout de langue peut être un plus, afin de **diversifier les clients**, pouvant possiblement permettre une utilisation dans des pays non francophones, et anglophones.

Conclusion :

La version 3.0 d'EasySave offre des fonctionnalités avancées. Les évolutions apportées répondent aux besoins demandés dans le cahier des charges par les utilisateurs en matière de sauvegarde de données, de gestion des priorités, et d'optimisation des performances.

L'interface utilisateur réalisée en WPF offre une expérience visuelle améliorée et la console déportée permet un suivi en temps réel des sauvegardes sur un poste distant. De plus, l'application mono-instance garantit une utilisation unique sur chaque ordinateur, évitant ainsi les conflits d'exécution.

Cette version 3.0 d'EasySave représente une amélioration de la console en interface pour une meilleure expérience utilisateur.