EasySave - Solution de Sauvegarde Professionnelle (V3.0)

Documentation Utilisateurs

Groupe 5

5 février 2026

Table des matières

Introduction

Bienvenue dans la documentation technique de la version 3.0 d'EasySave, la solution de sauvegarde phare de la suite ProSoft.

Cette mise à jour majeure marque une transition technologique importante : le passage d'une exécution séquentielle à un modèle de traitement en parallèle. Conçue pour répondre aux exigences des infrastructures serveurs modernes, la V3.0 optimise l'utilisation des ressources système tout en offrant un contrôle granulaire et une visibilité en temps réel sur chaque flux de données.

Les piliers de la version 3.0

Pour cette itération, l'architecture logicielle a été renforcée autour de quatre axes stratégiques :

- **Multi-threading & Performance :** Abandon du mode séquentiel au profit de sauvegardes simultanées, permettant un gain de temps considérable sur les gros volumes de données.
- **Gestion Intelligente des Flux :** Introduction de priorités par extensions de fichiers et limitation de la bande passante pour les fichiers volumineux ($> n$ Ko), évitant ainsi la saturation des ressources réseau.
- **Contrôle Interactif :** Une interface utilisateur enrichie permettant de piloter chaque travail individuellement (Play, Pause, Stop) avec un retour d'état en temps réel.
- **Centralisation & Docker :** Une nouvelle capacité de déport des logs via un service conteneurisé Docker, facilitant l'administration de parcs informatiques multi-sites.

Objectif de ce document

Ce manuel de support a pour but de fournir aux techniciens les clés nécessaires pour assister les clients sur les nouvelles fonctionnalités réseau (Docker), les paramètres de priorité et la gestion des conflits liés aux logiciels métiers.

1. Documentation Utilisateur

2.1. Configuration Minimale Requise

Pour garantir le bon fonctionnement des sauvegardes en parallèle et de l'interface graphique Avalonia, les postes clients et serveurs doivent respecter les critères suivants :

- Système d'exploitation : Windows 10 (1809+) ou Windows Server 2019+
- Runtime : **Microsoft .NET 10 Desktop Runtime**,
- Processeur : Multi-cœur recommandé (pour l'optimisation du multi-threading V3.0)
- Réseau : Accès TCP/IP vers le serveur Docker (si la centralisation des logs est activée)

2.2. Emplacements par défaut et Arborescence

Conformément aux directives de la DSI, aucun fichier n'est stocké dans des dossiers temporaires volatils (type C:\temp). L'installation type se décline comme suit :

Composant	Emplacement / Nom du fichier	Description
Exécutable	C:\Program Files\ProSoft\EasySave\EasySave.exe	Application principale (UI & Logique)
Librairie Log	C:\Program Files\ProSoft\EasySave\EasyLog.dll	DLL partagée pour la gestion des logs
Configuration	AppData\EasySave\settings.json	Paramètres généraux (Logiciel métier, extensions prioritaires, seuil n Ko)
État Temps Réel	AppData\EasySave\state.json	État d'avancement des travaux (lecture Notepad optimisée)
Logs Locaux	AppData\EasySave\Logs\2026-02-25.json	Historique journalier des transferts

Figure 1 : Tableau de l'architecture

2.3. Fonctionnement de la Centralisation

Pour les clients Grands Comptes, les logs peuvent être déportés.

- **Conteneur :** Image Docker prosoft/easylog-centralizer
- **Port par défaut :** 5000 (TCP)
- **Mode de fonctionnement :** Le logiciel EasySave tente une connexion au service de centralisation. En cas d'échec, les logs sont conservés localement pour éviter toute perte de données.

2.4. Paramètres Critiques

Le support peut modifier directement les comportements suivants dans le fichier settings.json :

- **CriticalProcesses :** Nom du processus à surveiller (ex: CalculatorApp).
- **PriorityExtensions :** Liste des extensions à traiter en priorité (ex: docx, pdf).
- **MaxKo :** Valeur n en Ko pour la limitation du transfert simultané.

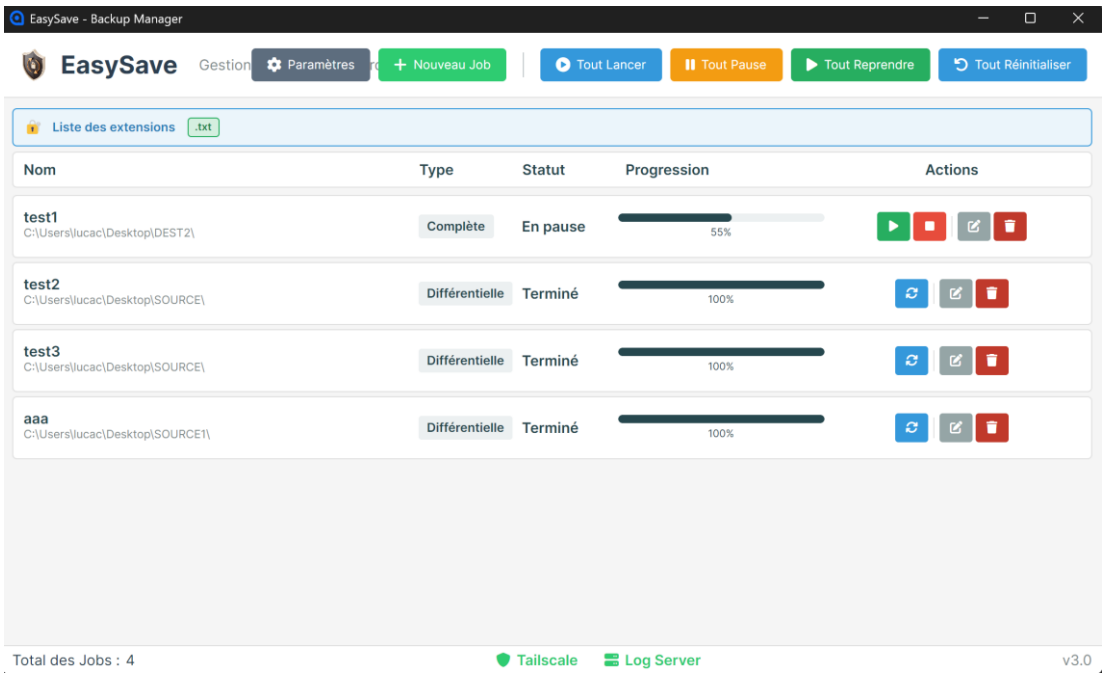

Figure 2 : Capture de l'application

1. Prise en main rapide

Le logiciel EasySave vous permet de sécuriser vos données via une interface graphique intuitive ou par ligne de commande.

- **Changement de langue :** Sélectionnez votre langue (Français/Anglais) dans le menu Paramètres.

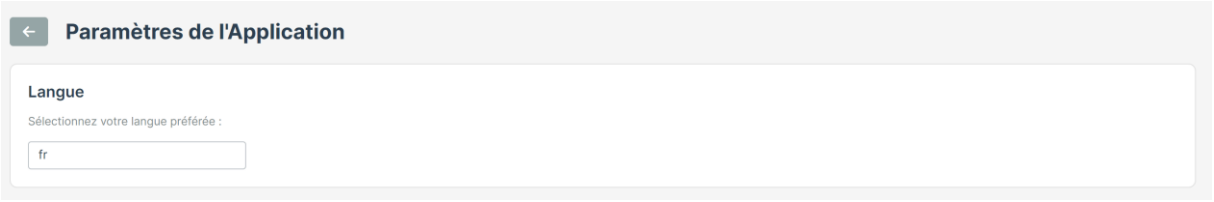

Figure 3 : Capture des paramètres

2. Création et Exécution d'un travail

1. Cliquez sur **"Ajouter un travail"**.
2. Saisissez un **Nom**, le **Répertoire Source** et le **Répertoire Cible**.
3. Choisissez le type : **Complète** (tout copier) ou **Différentielle** (uniquement les fichiers modifiés).
4. Cliquez sur le bouton **Play** pour démarrer.

Figure 4 : Capture de la création d'un travail

3. Contrôle en Temps Réel (Nouveautés V3.0)

Grâce au mode parallèle, vous pouvez piloter chaque sauvegarde indépendamment :

- **Pause :** Suspend le travail après la fin du transfert du fichier actuel.
- **Play :** Reprend une sauvegarde là où elle s'était arrêtée.
- **Stop :** Arrête immédiatement l'ensemble du processus de sauvegarde.
- **Barre de progression :** Suivez le pourcentage d'avancement et le volume de données restant en temps réel.

Figure 5 : Capture des sauveguardes

4. Automatisation et Ligne de Commande

Vous pouvez lancer vos sauvegardes directement depuis un terminal :

- EasySave.exe 1-5 : Exécute les travaux de 1 à 5.
- EasySave.exe 2;4 : Exécute uniquement les travaux 2 et 4.

```
C:\Users\lucac\Documents\cesi\genie_logiciel\EasySave\EasySave.GUI\bin\Debug\net10.0>EasySave.GUI.exe 1-3
```

5. Règles de Sécurité et Priorités

Logiciel Métier : *Si votre logiciel de travail (ex: Calculatrice, ERP) est ouvert, EasySave se mettra automatiquement en* ***Pause*** *pour ne pas ralentir votre activité. Il reprendra dès la fermeture de celui-ci.*

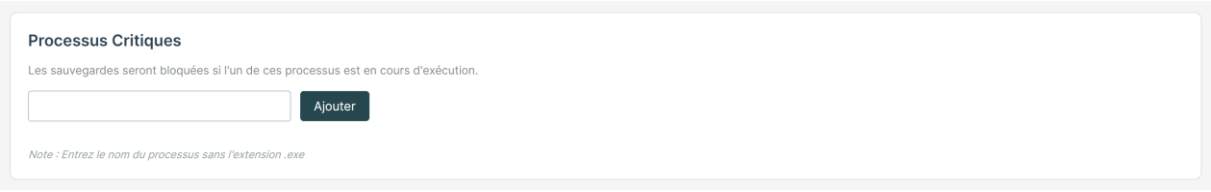

Figure 6 : Capture des processus critique

- **Fichiers Prioritaires :** Vos fichiers importants (définis par extension) sont toujours transférés avant les autres.

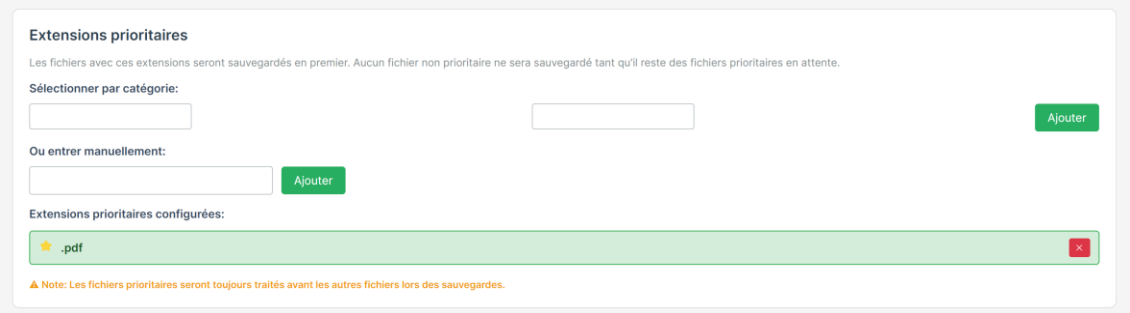

Figure 7 : Capture des extensions prioritaire

- **Chiffrement :** Les fichiers sensibles sont automatiquement cryptés via

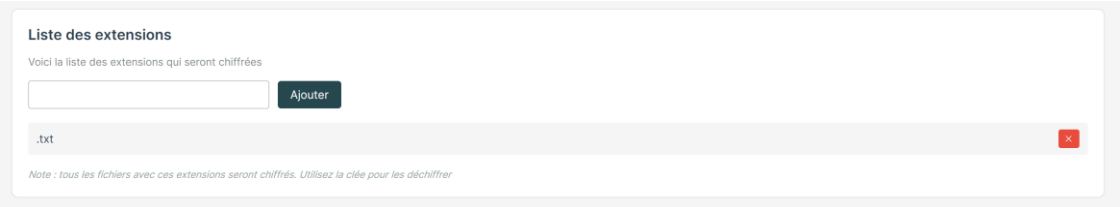

Figure 8 : Capture des extensions cryptées

- **CryptoSoft** selon les règles définies par votre administrateur.

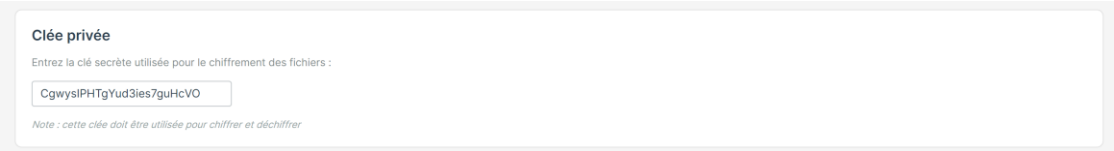

Figure 8 : Capture de la clée privée

6. Les logs

Le format des logs, xml ou json :

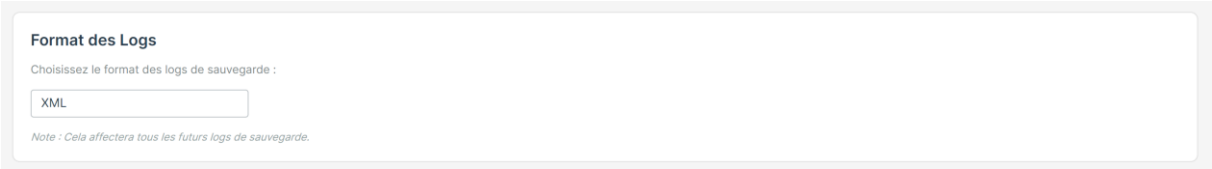

Figure 9 : Capture du format des logs

La configuration de l'enregistrement des logs, sur le serveur, en local (sur le pc), ou sur le Windows event viewer

Figure 10 : Capture de la configuration des logs

Et la taille maximum autorisé à être télécharger en parallèle

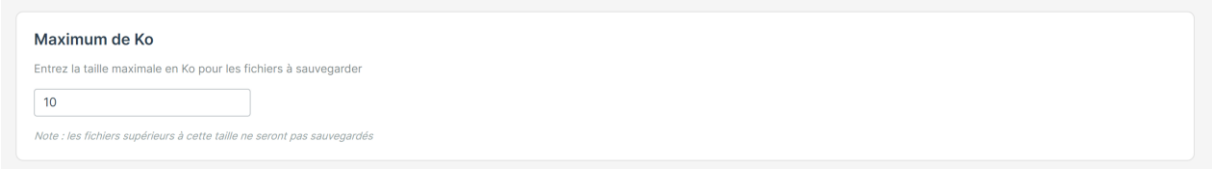

Figure 11 : Taille des fichiers maximum

2. Choix Techniques

3.1. Sauvegarde en Parallèle

La version précédente exécutait les travaux de sauvegarde de manière séquentielle, ce qui entraînait des temps d'exécution longs lorsque plusieurs jobs étaient lancés. L'objectif est de permettre l'exécution simultanée de plusieurs sauvegardes pour optimiser le temps total de traitement.

Nous avons opté pour l'utilisation du **Task Parallel Library (TPL)** de .NET, qui permet de créer des tâches asynchrones exécutées en parallèle. Chaque job de sauvegarde est encapsulé dans une tâche indépendante, et l'ensemble des tâches est exécuté simultanément via un mécanisme d'attente groupée.

Lorsque l'utilisateur lance plusieurs jobs, le système crée une tâche pour chacun d'eux. Ces tâches sont distribuées sur le pool de threads du système d'exploitation, permettant une exécution véritablement parallèle sur les processeurs multi-cœurs. Le système attend ensuite la fin de toutes les tâches avant de signaler la complétion.

Justification du Choix

Alternative Envisagée	Avantages	Inconvénients	Décision
TPL (Task.Run)	Natif .NET, simple, scalable, support de l'annulation	-	Retenu
Parallel.ForEach	Parallélisme automatique	Moins de contrôle individuel sur chaque tâche	Écarté
Threads manuels	Contrôle total	Complexité de gestion, risques d'erreurs	Écarté

Figure 12 : Tableau du choix des sauvegarde

Le TPL a été choisi car il offre le meilleur équilibre entre simplicité d'implémentation et contrôle sur l'exécution. Il permet notamment d'associer un jeton d'annulation à chaque tâche, ce qui est essentiel pour les fonctionnalités de pause et d'arrêt.

3.2. Gestion des Fichiers Prioritaires

Certains fichiers sont considérés comme critiques et doivent être sauvegardés en priorité. L'exigence stipule qu'aucun fichier non prioritaire ne peut être traité tant qu'il reste des fichiers prioritaires en attente.

Nous avons implémenté un **système de tri préalable** des fichiers avant le traitement. Avant de commencer la copie, la liste des fichiers est triée de manière à placer les fichiers ayant une extension prioritaire en tête de liste.

1. L'utilisateur configure les extensions prioritaires dans les paramètres généraux (exemple : .docx, .xlsx, .pdf)
2. Au démarrage d'un job, le système récupère la liste complète des fichiers à sauvegarder
3. Cette liste est triée : les fichiers dont l'extension figure dans la liste prioritaire sont placés en premier
4. Le traitement s'effectue ensuite dans l'ordre de la liste triée

Le tri préalable est la solution la plus élégante car elle ne nécessite qu'une seule passe de traitement et garantit que tous les fichiers prioritaires sont traités avant les autres.

3.3. Limitation des Transferts de Gros Fichiers (MaxKo)

Pour éviter la saturation de la bande passante réseau ou disque, il est interdit de transférer simultanément deux fichiers dont la taille dépasse un seuil configurable (MaxKo). Cependant, les fichiers plus petits doivent pouvoir continuer à être transférés pendant qu'un gros fichier est en cours de copie.

Nous avons créé un composant dédié appelé **LargeFileTransferManager** qui utilise un **sémaphore** avec une capacité de 1. Ce sémaphore est partagé entre tous les jobs et n'est acquis que pour les fichiers dépassant le seuil MaxKo.

1. Avant de copier un fichier, le système vérifie si sa taille dépasse le seuil MaxKo
2. Si oui, il tente d'acquérir le sémaphore :
    - Si le sémaphore est disponible, le transfert commence
    - Si le sémaphore est déjà pris (un autre gros fichier est en cours), le job attend
3. Si non (fichier petit), le transfert commence immédiatement sans attendre
4. À la fin du transfert d'un gros fichier, le sémaphore est libéré

Justification du Choix

Alternative Envisagée	**Avantages**	**Inconvénients**	**Décision**
Sémaphore partagé	Thread-safe, natif .NET, non-bloquant pour petits fichiers	-	Retenu
Mutex global	Fonctionne inter-processus	Bloque tous les fichiers, pas seulement les gros	Écarté
Queue centralisée	Contrôle très fin	Complexité excessive	Écarté

Figure 13 : Tableau du choix de la limitation des fichiers

Le sémaphore est la solution idéale car il permet de bloquer uniquement les gros fichiers tout en laissant passer les petits fichiers sans attente.

Le seuil MaxKo est configurable dans les paramètres généraux de l'application. La valeur par défaut est de 10 Ko.

3.4. Interaction Temps Réel (Pause/Play/Stop)

L'utilisateur doit pouvoir contrôler chaque travail de sauvegarde en temps réel : mettre en pause, reprendre ou arrêter complètement un job. De plus, il doit pouvoir suivre la progression de chaque travail.

Nous avons implémenté un système de **flags thread-safe** combiné à des **jetons d'annulation** :

- **Pause/Resume** : Un dictionnaire concurrent stocke l'état de pause de chaque job
- **Stop** : Un jeton d'annulation est associé à chaque job pour permettre l'arrêt immédiat
- **Progression** : Un gestionnaire d'état notifie l'interface utilisateur via le pattern Observer

Après chaque fichier copié, le système vérifie si une demande de pause a été émise pour ce job. Si oui, le job entre dans une boucle d'attente jusqu'à ce que l'utilisateur demande la reprise.

L'arrêt utilise le mécanisme standard de .NET pour l'annulation coopérative. Lorsque l'utilisateur clique sur Stop, le jeton d'annulation est déclenché, et le job vérifie ce jeton avant chaque opération pour sortir proprement.

Le gestionnaire d'état (StateManager) implémente le pattern Observer. À chaque mise à jour de l'état d'un job (fichier copié, progression, etc.), un événement est émis et l'interface utilisateur est notifiée pour rafraîchir l'affichage.

3.5. Pause Automatique sur Logiciel Métier

Si un logiciel métier critique est en cours d'exécution (exemple : Outlook, Excel, Calculatrice), les sauvegardes doivent automatiquement se mettre en pause pour éviter les conflits. La reprise doit être automatique dès que le logiciel métier est fermé.

Nous avons implémenté un système de **détection de processus par polling**. Avant chaque fichier, le système vérifie si l'un des processus critiques configurés est en cours d'exécution.

1. L'utilisateur configure la liste des processus critiques pour chaque job (exemple : outlook, excel, calculatrice)
2. Avant de traiter chaque fichier, le système interroge la liste des processus Windows en cours
3. Si un processus critique est détecté :
    - Le job passe en état "Paused"
    - Une boucle de vérification s'active (toutes les secondes)
    - Dès que le processus n'est plus détecté, le job reprend automatiquement

Le polling est suffisant pour ce cas d'usage car une latence d'une seconde est acceptable. La simplicité et la fiabilité de cette approche l'emportent sur les alternatives plus complexes.

3.6. CryptoSoft Mono-Instance

Le logiciel de chiffrement CryptoSoft ne peut être exécuté qu'en une seule instance à la fois sur un même ordinateur. Avec l'exécution parallèle des jobs, plusieurs tentatives de lancement simultané peuvent survenir.

Nous avons mis en place une **double protection** :

1. **Côté CryptoSoft** : Un Mutex global garantit qu'une seule instance peut s'exécuter
2. **Côté EasySave** : Un gestionnaire (CryptoSoftManager) avec sémaphore sérialise les appels

Au démarrage, CryptoSoft tente d'acquérir un Mutex global identifié par un GUID unique. Si le Mutex est déjà pris, le programme retourne immédiatement avec un code d'erreur (-1).

Avant de lancer CryptoSoft, EasySave acquiert un sémaphore interne. Cela évite de lancer CryptoSoft si on sait déjà qu'une autre instance est en cours. De plus, un mécanisme de retry avec délai croissant est implémenté pour gérer les cas de conflit.

3.7. Centralisation des Logs (Docker)

Le logiciel pouvant être déployé sur plusieurs postes, il est nécessaire de centraliser les logs pour simplifier leur gestion. L'utilisateur doit pouvoir choisir entre :

- Logs uniquement en local
- Logs uniquement sur le serveur centralisé
- Logs en local ET sur le serveur

Nous avons développé un **serveur de logs sous Docker** basé sur ASP.NET Core, exposant une API REST pour la réception des logs.

Fonctionnement

1. **Envoi des logs** : Chaque client EasySave envoie ses logs via une requête HTTP POST au serveur
2. **Identification** : Chaque log contient un identifiant utilisateur pour différencier les sources
3. **Stockage** : Le serveur stocke les logs dans des fichiers JSON/XML organisés par date et utilisateur
4. **Persistance** : Un volume Docker assure la persistance des données

Mode	Local	Serveur	Description
Local	x	-	Logs uniquement sur le poste client
Server	-	x	Logs uniquement sur le serveur Docker
Both	x	x	Logs sur le poste client ET le serveur

Figure 14 : Tableau de l'enregistrement des logs

Docker a été choisi pour sa portabilité et sa facilité de déploiement. L'API REST permet une intégration simple avec n'importe quel client HTTP.

Le serveur est configuré via un fichier docker-compose.yml qui définit :

- L'image à construire
- Le port d'exposition (5005)
- Le volume de persistance pour les logs

3.8. API de Contrôle à Distance

En complément de la centralisation des logs, nous avons souhaité permettre le contrôle des jobs à distance via une interface web hébergée sur le serveur Docker.

Nous avons étendu l'API REST du serveur Docker pour inclure des **endpoints de commande** et développé une **interface web React** pour la supervision.

Le système utilise un mécanisme de **polling** : le client EasySave interroge régulièrement le serveur pour récupérer les commandes en attente.

Endpoints de l'API

Endpoint	Méthode	Description
/api/commands	POST	Envoyer une commande (pause, resume, stop)
/api/commands/{userId}/pending	GET	Récupérer les commandes en attente
/api/commands/{id}/ack	POST	Confirmer l'exécution d'une commande
/api/state/{userId}	GET	Récupérer l'état des jobs d'un utilisateur
/api/state	POST	Mettre à jour l'état d'un job
/api/logs	POST	Envoyer un log
/api/logs/{userId}	GET	Récupérer les logs d'un utilisateur

Figure 15 : Tableaux des endpoints de l'API

Justification du Choix

Alternative Envisagée	Avantages	Inconvénients	Décision
Polling HTTP	Simple, fonctionne partout, fiable	Latence de 2 secondes	Retenu
WebSocket	Temps réel	Complexité de gestion des connexions	Écarté
SignalR	Abstraction WebSocket	Dépendance supplémentaire	Écarté

Figure 16 : Tableau du choix des technologie

Le polling a été choisi car une latence de 2 secondes est acceptable pour des commandes de type pause/stop. Cette approche est également plus robuste face aux déconnexions réseau.

Conclusion

La version 3.0 d'EasySave représente une évolution majeure de l'application, passant d'un modèle séquentiel simple à une architecture parallèle sophistiquée. Les choix techniques effectués permettent de répondre à toutes les exigences tout en maintenant une base de code maintenable et extensible.

Les mécanismes de synchronisation (sémaphores, jetons d'annulation, dictionnaires concurrents) garantissent un fonctionnement correct en environnement multi-thread, tandis que l'architecture Docker permet une centralisation efficace des logs et un contrôle à distance des sauvegardes.