

Description du projet et guide d'utilisation EasySaveProject

Matis Bouquin, Yanis Kadri, Mathias Duprat, Jason Bourgeois

14 février 2025

Table des matières

1	Description du projet	2
1.1	Choix Algorithmique	2
1.2	Gestion du GIT	3
1.3	Test du code	3
1.4	Gestion de projet	4
2	Version 1.0.0	5
2.1	Installation	5
2.1.1	Téléchargement et exécution	5
2.2	Choix de la langue et menu	5
2.3	Commandes : gestion des travaux de sauvegarde	6
2.3.1	Création d'un travail de sauvegarde	6
2.3.2	Exécution d'un travail de sauvegarde	6
2.3.3	Exécution séquentielle de tous les travaux de sauvegarde	6
2.3.4	Affichage de tous les travaux existants	6
2.3.5	Commande de sortie de l'application	7
2.4	Fichiers de journalisation et suivi en temps réel	7
2.4.1	Fichier Log journalier	7
2.4.2	Fichier d'état en temps réel	7
2.5	Configuration et compatibilité	8
2.6	Évolutions futures	8
3	Version 2.0.0	9
3.1	Installation	9
3.1.1	Téléchargement et exécution	9
3.2	Choix de la langue et menu	10
3.3	Commandes : gestion des travaux de sauvegarde	10
3.3.1	Création d'un travail de sauvegarde	10
3.3.2	Onglet principal	11
3.4	Fichiers de journalisation et suivi en temps réel	11
3.4.1	Fichier Log journalier	11
3.4.2	Fichier d'état en temps réel	12
3.5	Configuration et compatibilité	13
3.6	Évolutions futures	13

1 Description du projet

1.1 Choix Algorithmique

Le projet EasySaveProject vise à fournir une solution robuste et efficace pour la gestion et l'exécution de sauvegardes de fichiers. Son architecture repose sur des principes de conception bien établis permettant une extensibilité et une maintenance facilitées. L'un des aspects fondamentaux de ce projet est l'adoption de différents design patterns permettant d'assembler des modules et structures communes et connues de code dans un même projet. Les différentes bonnes pratiques de code ont été détaillées dans le fichier README du projet GitHub.

Le design pattern Command est un modèle de conception qui permet d'encapsuler une requête sous forme d'objet, facilitant ainsi la gestion des opérations et l'implémentation de fonctionnalités. Dans notre projet, cette approche est utilisée pour structurer les commandes qui interagissent avec le gestionnaire de tâches. L'interface ICommand définit un contrat standard que toutes les commandes doivent implémenter, assurant une séparation claire entre l'invocation et l'exécution des actions. Cette abstraction permet de manipuler des commandes de manière uniforme, ce qui simplifie la gestion des différentes actions disponibles.

Le diagramme de classes (voir annexe) illustre comment les différentes entités interagissent entre elles. La classe CommandController joue un rôle clé en orchestrant l'exécution des différentes commandes. Elle reçoit les requêtes et les transmet aux instances appropriées de ICommand pour traitement. Ces instances sont ensuite gérées par des classes spécifiques comme CreateJobCommand et ExecuteJobCommand, qui implémentent chacune une action distincte en rapport avec la gestion des tâches de sauvegarde. Ce design pattern Command permet de généraliser la création de commandes et rend l'ensemble de la fonctionnalité modulable. En effet, il est très facile d'ajouter une nouvelle commande et son comportement en l'intégrant dans le code existant.

Le projet repose également sur une approche modulaire, favorisant la séparation des responsabilités. Par exemple, la classe JobManager est responsable de la gestion des instances de JobModel via l'utilisation d'une Factory, qui elle, gère l'instanciation des objets de type Job, tandis que JobService fournit des fonctionnalités permettant d'exécuter et de superviser les processus de sauvegarde. Cette organisation permet d'assurer une meilleure évolutivité du système et facilite l'ajout de nouvelles fonctionnalités sans perturber les composants existants.

L'architecture logicielle utilisée ici s'apparente à un MVC (voir annexe). En effet, l'arborescence des fichiers laisse apparaître un dossier "Model" qui contient l'ensemble des classes définissant les structures des objets (sans les méthodes de logique métier). Ces méthodes sont créées dans les classes dites "Service", qui appliquent des comportements aux différents objets selon les besoins métier définis. L'aspect vue est lui séparé dans un dossier "View". Bien qu'actuellement le cahier des charges définisse une interface de type console, créer cette structure

permet de faciliter l'évolution et la transition vers une solution type IHM.

Enfin, l'utilisation de plusieurs interfaces dans la conception du projet permet une meilleure flexibilité et testabilité du code. Les services spécialisés, tels que `JobDifferencielService` et `JobCompleteService`, illustrent cette approche en encapsulant des comportements spécifiques liés aux types de sauvegardes disponibles. Cette architecture garantit une gestion efficace des opérations et une adaptation fluide aux éventuelles évolutions du projet.

1.2 Gestion du GIT

Le projet est versionné avec Git et suit une méthodologie rigoureuse pour assurer la qualité du code et la cohérence des évolutions. Nous utilisons une organisation structurée des branches avec une branche principale `main` représentant l'environnement de production. La branche `develop` regroupe l'ensemble des développements en cours et sert de point de départ pour toutes les nouvelles fonctionnalités.

Chaque nouvelle fonctionnalité est développée dans une branche dédiée issue de `develop`, en suivant une convention de nommage en kebab case avec le préfixe `feat[la-feature]` pour les fonctionnalités, `docs` pour la documentation, etc. Cette approche permet une identification rapide des items en cours de développement. Une fois la fonctionnalité terminée et validée via des tests, elle est fusionnée dans `develop`. Lorsqu'une version stable est atteinte, une release est effectuée depuis `develop` vers `main`, garantissant ainsi un cycle de développement maîtrisé et fiable.

En adoptant cette approche, nous garantissons une gestion efficace du projet, une meilleure collaboration entre les membres de l'équipe et une intégration continue fluide. L'utilisation de Git combinée à une CI/CD bien définie permet de réduire les erreurs et d'assurer un déploiement maîtrisé des nouvelles versions du projet, ce qui est une solution très intéressante dans le cadre de ce projet itératif en plusieurs versions.

1.3 Test du code

La qualité du code est assurée par un ensemble de tests automatisés ainsi qu'un pipeline CI/CD. Ce pipeline effectue une vérification systématique du respect des conventions de nommage des commits, garantissant une clarté et une homogénéité dans l'historique du projet. Chaque commit doit respecter une structure bien définie, favorisant ainsi la compréhension des évolutions du projet.

Afin de garantir la fiabilité de l'application, nous avons mis en place des tests automatisés end-to-end couvrant divers scénarios d'utilisation. Ces tests sont spécifiés sous forme de fichiers

feature, utilisant une syntaxe claire qui permet de définir précisément les entrées, les actions et les résultats attendus. L'utilisation de fichiers feature présente plusieurs avantages :

- D'une part, une personne néophyte peut comprendre l'ensemble des tests grâce à leur lisibilité.
- D'autre part, pour les développeurs, cette approche facilite l'écriture de tests automatisés et permet un gain de temps considérable lors de leur développement et de leur maintenance.

L'ensemble de ces tests permet d'anticiper et de prévenir les erreurs, garantissant ainsi un comportement fiable et robuste de l'application dans diverses situations d'utilisation.

1.4 Gestion de projet

Pour assurer une organisation efficace et une progression maîtrisée du développement, nous avons opté pour une méthodologie inspirée de SCRUM. Cette approche nous permet d'adopter un mode de travail itératif et adaptatif, particulièrement pertinent dans le cadre de notre projet qui se déroule en plusieurs phases. En effet, nous avons à enchaîner trois versions successives du projet, chacune accompagnée d'un cahier des charges enrichi. Cette structuration progressive nécessite un suivi rigoureux et une capacité d'adaptation rapide, ce que SCRUM nous offre grâce à son cadre méthodologique bien défini.

Enfin, la méthodologie SCRUM nous apporte une meilleure visibilité sur l'ensemble du projet, tout en offrant une flexibilité précieuse. Grâce à la rétrospective en fin de sprint, nous pouvons identifier ce qui fonctionne bien et ce qui doit être amélioré. Cette approche itérative et évolutive nous permet d'optimiser continuellement notre processus de travail et de garantir une meilleure qualité du livrable final.

2 Version 1.0.0

EasySave est une application console développée en .NET Core, permettant la gestion simple et efficace de travaux de sauvegarde. Cette version prend en charge jusqu'à cinq travaux de sauvegarde simultanés et assure un suivi détaillé des actions réalisées.

Les principales fonctionnalités incluent :

- Création et gestion de jusqu'à 5 travaux de sauvegarde.
- Sauvegarde complète ou différentielle des fichiers et répertoires.
- Compatibilité avec les disques locaux, externes et lecteurs réseau.
- Enregistrement en temps réel des actions dans un fichier log journalier.
- Suivi en temps réel de l'état des sauvegardes via un fichier dédié.
- Interface multilingue (français et anglais).
- Exécution des sauvegardes via des commandes spécifiques.

2.1 Installation

Avant d'utiliser EasySave, assurez-vous d'avoir .NET Core installé sur votre machine.

2.1.1 Téléchargement et exécution

- **Téléchargement** : Récupérez l'exécutable dans le dossier fourni avec le programme.
- **Exécution** : Ouvrez un terminal et exécutez la commande suivante :

```
dotnet EasySave.exe
```

Cela lancera l'application en mode console et vous donnera accès aux différentes fonctionnalités.

2.2 Choix de la langue et menu

Lors du lancement du programme, l'utilisateur est invité à sélectionner la langue de l'interface. Cette fonctionnalité permet d'offrir une expérience personnalisée et accessible à tous les utilisateurs, indépendamment de leur langue maternelle. Les options de langue disponibles peuvent inclure, par exemple, le français et l'anglais.

Une fois que l'utilisateur a choisi la langue, le menu principal s'affiche, proposant différentes options pour gérer les travaux de sauvegarde. Voici un exemple de menu traduit :

- 1. Afficher les jobs

- 2. Créer une tâche de sauvegarde
- 3. Exécuter une tâche spécifique
- 4. Exécuter toutes les tâches
- 5. Quitter

2.3 Commandes : gestion des travaux de sauvegarde

2.3.1 Création d'un travail de sauvegarde

Pour créer un nouveau travail de sauvegarde, suivez les étapes suivantes :

1. **Sélection de l'option** : Choisissez l'option permettant d'ajouter un nouveau travail de sauvegarde.
2. **Saisie des informations** :
 - **Nom de la sauvegarde** : Identifiant unique du travail.
 - **Répertoire source** : Dossier contenant les fichiers à sauvegarder.
 - **Répertoire cible** : Emplacement où seront stockées les sauvegardes.
 - **Type de sauvegarde** :
 - **Complète** : Tous les fichiers sont copiés à chaque exécution.
 - **Différentielle** : Seuls les fichiers modifiés depuis la dernière sauvegarde complète sont copiés.

2.3.2 Exécution d'un travail de sauvegarde

Une fois le travail de sauvegarde créé, il peut être exécuté de deux manières :

- **Exécution d'un seul travail** : Sélectionnez le numéro du travail à exécuter.
- **Exécution multiple** :
 - 1-3 : Exécute les sauvegardes 1 à 3 de manière séquentielle.
 - 1;3;5 : Exécute uniquement les sauvegardes 1, 3 et 5.

2.3.3 Exécution séquentielle de tous les travaux de sauvegarde

L'utilisateur a la possibilité de lancer tous les travaux de sauvegarde existants en une seule commande. Cela permet d'exécuter chaque travail de manière ordonnée, en suivant l'ordre de leur création.

2.3.4 Affichage de tous les travaux existants

L'utilisateur peut demander l'affichage de tous les travaux de sauvegarde actuellement enregistrés dans le système. Cela permet de visualiser rapidement les détails de chaque travail, tels que les chemins source et cible, la date de création et le type de sauvegarde.

2.3.5 Commande de sortie de l'application

L'utilisateur peut choisir de quitter l'application à tout moment en utilisant une commande dédiée. Cela permet de fermer l'application proprement et de sauvegarder tous les états ou configurations nécessaires.

2.4 Fichiers de journalisation et suivi en temps réel

L'ensemble de la journalisation est stocké dans un dossier **easySave** qui est créé dans le dossier **Documents** de la machine. Il se compose de deux sous-dossiers, **log** pour stocker la journalisation et **easySaveSetting** qui contient la liste des jobs enregistrés par l'utilisateur.

2.4.1 Fichier Log journalier

Chaque action réalisée durant une sauvegarde est enregistrée en temps réel dans un fichier log journalier au format JSON.

Les informations minimales attendues sont :

- **Timestamp** : Le moment exact où l'action a eu lieu.
- **JobName** : Le nom du job de sauvegarde en cours d'exécution.
- **SourcePath** : L'adresse UNC complète du fichier source.
- **TargetPath** : L'adresse UNC complète du fichier cible.
- **FileSize** : La taille du fichier en cours de sauvegarde.
- **TransferTime** : Le temps pris pour transférer le fichier en millisecondes (négatif en cas d'erreur).
- **Messages** : Une liste des messages indiquant les étapes ou les erreurs rencontrées lors du processus de sauvegarde.

2.4.2 Fichier d'état en temps réel

Le programme enregistre en temps réel l'état d'avancement des travaux de sauvegarde avec diverses informations telles que le nom du travail, l'horodatage, l'état actuel et la progression.

Fichier d'état temps réel : Le logiciel doit enregistrer en temps réel, dans un fichier unique, l'état d'avancement des travaux de sauvegarde et l'action en cours. Les informations à enregistrer pour chaque travail de sauvegarde sont à minima :

- **JobName** : L'appellation du travail de sauvegarde.
- **LastActionTimestamp** : L'horodatage de la dernière action effectuée.
- **JobStatus** : L'état du travail de sauvegarde (ex : Actif, Non Actif...).
- **TotalEligibleFiles** : Le nombre total de fichiers éligibles pour la sauvegarde.
- **TotalFileSize** : La taille totale des fichiers à transférer.
- **Progress** : La progression actuelle du transfert.
- **RemainingFiles** : Le nombre de fichiers restants à transférer.

- **RemainingFileSize** : La taille des fichiers restants à transférer.
- **CurrentSourceFilePath** : L'adresse complète du fichier source en cours de sauvegarde.
- **CurrentDestinationFilePath** : L'adresse complète du fichier de destination.

2.5 Configuration et compatibilité

- EasySave fonctionne sur les disques locaux, disques externes et lecteurs réseau.
- Les fichiers de journalisation et d'état sont au format JSON.
- Les chemins tels que `C:\temp\` sont proscrits pour assurer une compatibilité serveur et éviter de perdre les données de journalisation.
- Une pagination dans le fichier d'état JSON permet une meilleure lisibilité.

2.6 Évolutions futures

Si la version 1.0 d'EasySave répond aux attentes, une version 2.0 avec interface graphique (architecture MVVM) pourra être développée.

3 Version 2.0.0

L'évolution de cette version implique la mise en place d'une interface graphique réalisé sous Avalonia UI permet une inter-opérabilité entre les OS (mac / windows). La partie métier elle est toujours réalisé via .NET.

Les principales fonctionnalités incluent :

- Nouvelle interface basée sur Avalonia pour une meilleure expérience utilisateur
- Prise en charge du cryptage des fichiers via CryptoSoft
- Cryptage des fichiers selon les extensions définies par l'utilisateur dans les paramètres
- Blocage des sauvegardes si un logiciel métier est en cours d'exécution
- Terminaison propre du fichier en cours de sauvegarde dans le cas de travaux séquentiels
- Possibilité de définir un logiciel métier dans les paramètres
- L'arrêt des travaux est consigné dans le fichier log
- Les journaux peuvent être enregistrés en JSON ou XML

Les fonctionnalités modifiées :

- Abandon du mode Console
- Plus aucune restriction sur le nombre de travaux créés
- Ajout d'une nouvelle information dans les fichiers de logs : le temps de cryptage des fichiers (en ms) :
 - 0** Pas de cryptage
 - > 0** Temps de cryptage (en ms)
 - < 0** Code erreur

3.1 Installation

Avant d'utiliser EasySave, assurez-vous d'avoir .NET Core installé sur votre machine.

3.1.1 Téléchargement et exécution

- **Téléchargement** : Récupérez l'exécutable dans le dossier fourni avec le programme.
- **Exécution** : Ouvrez un terminal et exécutez la commande suivante :

```
dotnet EasySave.exe
```

Cela lancera l'application en mode interface.

3.2 Choix de la langue et menu

L'onglet paramètre permet de choisir la langue et le format des logs (JSON ou XML). Lors du lancement du programme, l'utilisateur accède à une page d'accueil avec différents onglets comme suivant :

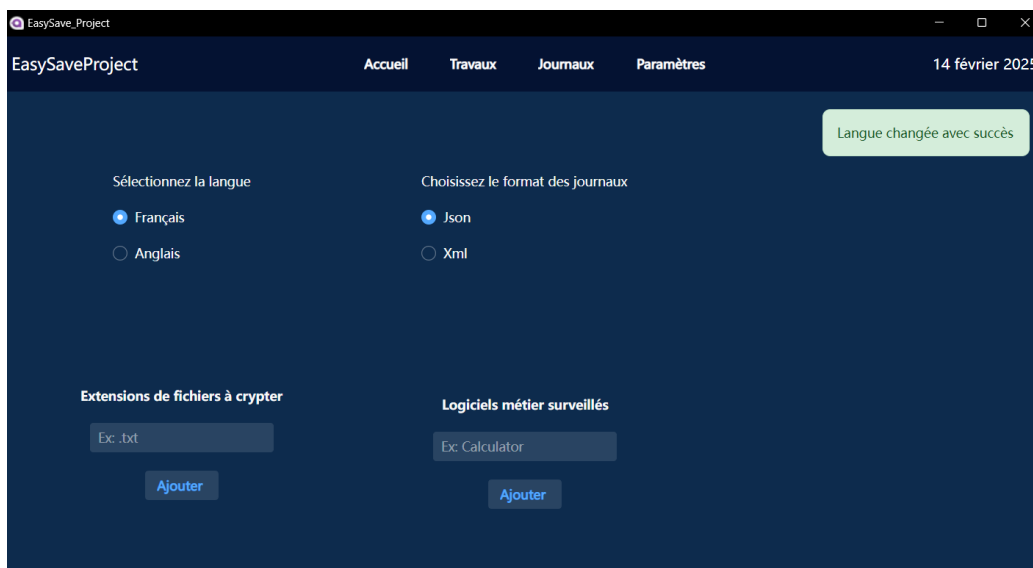


FIGURE 1 – Interface de l'application

3.3 Commandes : gestion des travaux de sauvegarde

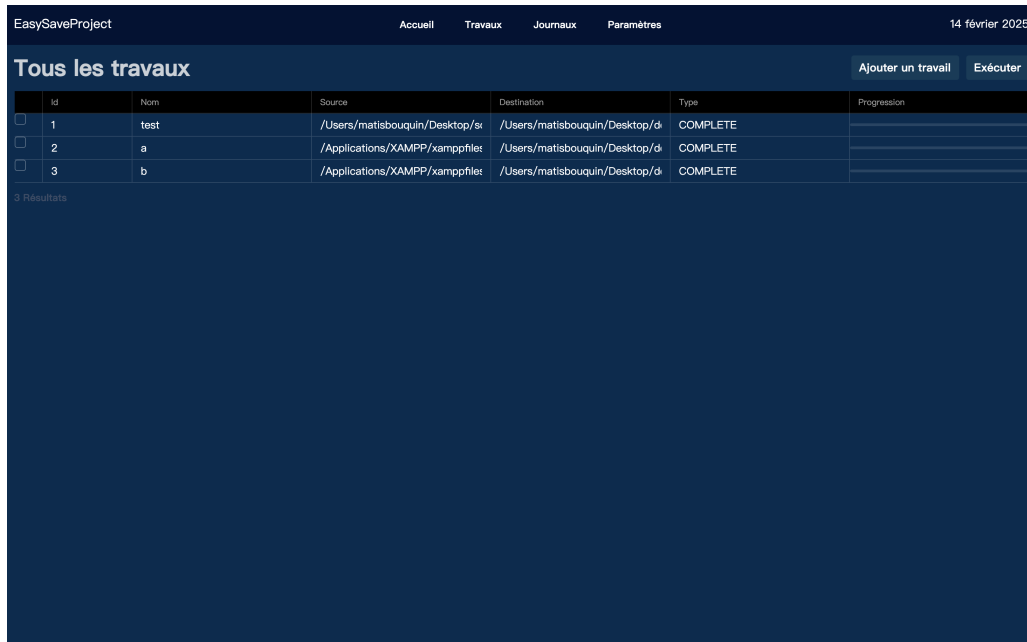
3.3.1 Création d'un travail de sauvegarde

Pour créer un nouveau travail de sauvegarde, suivez les étapes suivantes :

1. **Sélection de l'option** : Choisissez l'option permettant d'ajouter un nouveau travail de sauvegarde.
2. **Saisie des informations** :
 - **Nom de la sauvegarde** : Identifiant unique du travail.
 - **Répertoire source** : Dossier contenant les fichiers à sauvegarder.
 - **Répertoire cible** : Emplacement où seront stockées les sauvegardes.
 - **Type de sauvegarde** :
 - **Complète** : Tous les fichiers sont copiés à chaque exécution.
 - **Différentielle** : Seuls les fichiers modifiés depuis la dernière sauvegarde complète sont copiés.

3.3.2 Onglet principal

Cet onglet permet d'exécuter de 1 à N travaux de sauvegarde ainsi que de visualiser l'ensemble des travaux enregistrés. Il permet également de visualiser en temps réel l'état d'avancement de la sauvegarde



	Id	Nom	Source	Destination	Type	Progression
<input type="checkbox"/>	1	test	/Users/matisbouquin/Desktop/sr	/Users/matisbouquin/Desktop/d	COMPLETE	
<input type="checkbox"/>	2	a	/Applications/XAMPP/xamppfiles	/Users/matisbouquin/Desktop/d	COMPLETE	
<input type="checkbox"/>	3	b	/Applications/XAMPP/xamppfiles	/Users/matisbouquin/Desktop/d	COMPLETE	

3 Résultats

FIGURE 2 – Interface des travaux

3.4 Fichiers de journalisation et suivi en temps réel

L'ensemble de la journalisation est stocké dans un dossier **easySave** qui est créé dans le dossier **Documents** de la machine. Il se compose de deux sous-dossiers, **log** pour stocker la journalisation et **easySaveSetting** qui contient la liste des jobs enregistrés par l'utilisateur.

3.4.1 Fichier Log journalier

Chaque action réalisée durant une sauvegarde est enregistrée en temps réel dans un fichier log journalier au format JSON. et visualisable dans l'onglet associé paginé par jour.

Les informations minimales attendues sont :

- **Timestamp** : Le moment exact où l'action a eu lieu.
- **JobName** : Le nom du job de sauvegarde en cours d'exécution.
- **SourcePath** : L'adresse UNC complète du fichier source.
- **TargetPath** : L'adresse UNC complète du fichier cible.
- **FileSize** : La taille du fichier en cours de sauvegarde.

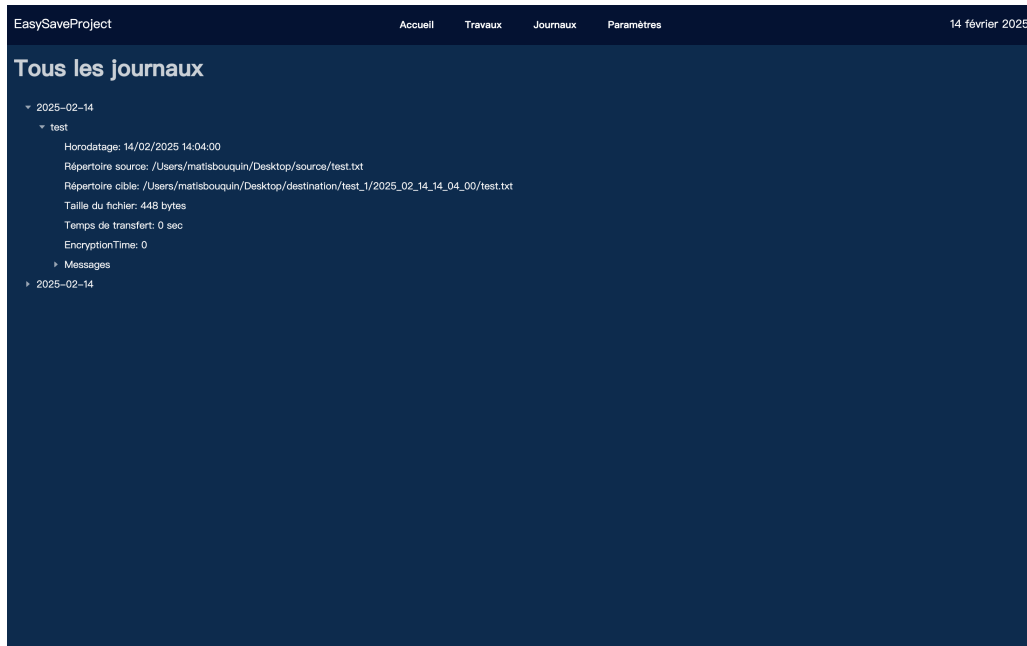


FIGURE 3 – Interface de la journalisation

- **TransferTime** : Le temps pris pour transférer le fichier en millisecondes (négatif en cas d'erreur).
- **Messages** : Une liste des messages indiquant les étapes ou les erreurs rencontrées lors du processus de sauvegarde.

3.4.2 Fichier d'état en temps réel

Le programme enregistre en temps réel l'état d'avancement des travaux de sauvegarde avec diverses informations telles que le nom du travail, l'horodatage, l'état actuel et la progression.

Fichier d'état temps réel : Le logiciel doit enregistrer en temps réel, dans un fichier unique, l'état d'avancement des travaux de sauvegarde et l'action en cours. Les informations à enregistrer pour chaque travail de sauvegarde sont à minima :

- **JobName** : L'appellation du travail de sauvegarde.
- **LastActionTimestamp** : L'horodatage de la dernière action effectuée.
- **JobStatus** : L'état du travail de sauvegarde (ex : Actif, Non Actif...).
- **TotalEligibleFiles** : Le nombre total de fichiers éligibles pour la sauvegarde.
- **TotalFileSize** : La taille totale des fichiers à transférer.
- **Progress** : La progression actuelle du transfert.
- **RemainingFiles** : Le nombre de fichiers restants à transférer.
- **RemainingFileSize** : La taille des fichiers restants à transférer.
- **CurrentSourceFilePath** : L'adresse complète du fichier source en cours de sauvegarde.
- **CurrentDestinationFilePath** : L'adresse complète du fichier de destination.

3.5 Configuration et compatibilité

- EasySave fonctionne sur les disques locaux, disques externes et lecteurs réseau.
- Les fichiers de journalisation et d'état sont au format JSON ou XML.
- Les chemins tels que `C:\temp\` sont proscrits pour assurer une compatibilité serveur et éviter de perdre les données de journalisation.
- Une pagination dans le fichier d'état JSON et XML permet une meilleure lisibilité.

3.6 Évolutions futures

La version 3.0.0 de easySave permettra d'utiliser des socket pour interagir à distance avec les travaux ainsi qu'à exécuter les travaux en parallèles grâce aux threads.