PROJET EASYSAVE

LIVRABLE 1

Groupe 1

ALBAN CALVO

EVAN JOASSON

MATHEO PINGET

JONAS MIONNET

05/02/2025

Sommaire

Contexte 03

Introduction 04

Description du Projet 05

Fonctionnalités Développées 06

Architecture et Développement 07

Gestion de projet 08

Gestion de Version 09

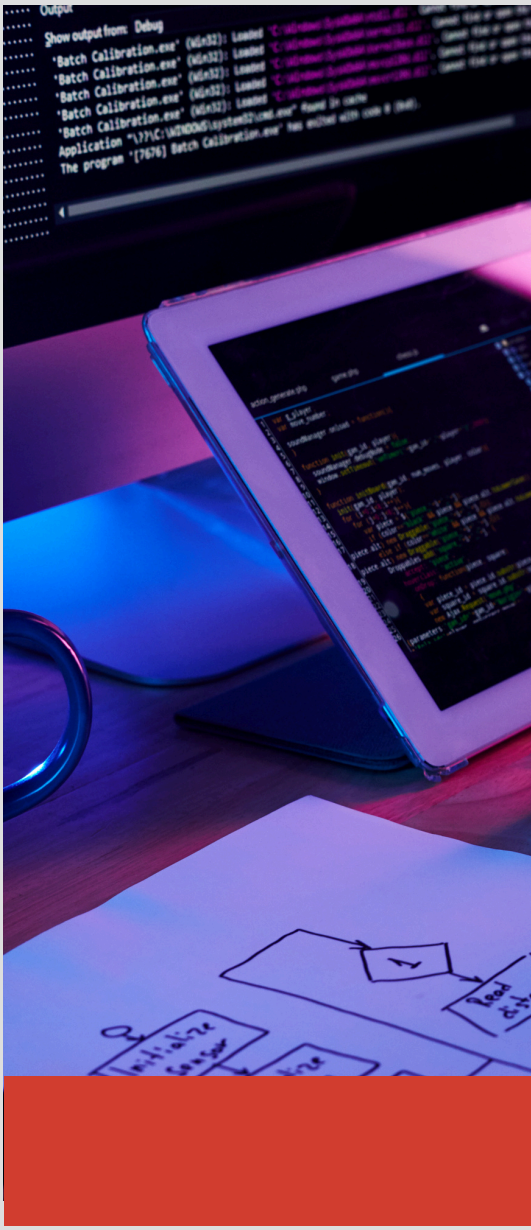
Diagrammes UML 10

Développement 17

Documentation 18

Architecture 19

Logs 20



Contexte

Le projet EasySave s'inscrit dans le cadre d'un développement logiciel au sein de l'éditeur de logiciels ProSoft. L'objectif est de concevoir un outil de sauvegarde performant et évolutif, répondant aux besoins des utilisateurs professionnels. Ce projet fait partie d'une démarche pédagogique visant à appliquer les principes de génie logiciel, notamment en matière de gestion des versions, de qualité du code et de documentation.

EasySave est destiné à être distribué chez les clients de ProSoft, ce qui impose des exigences élevées en matière de fiabilité et de facilité d'utilisation. La première version, objet de ce livrable, pose les bases fonctionnelles du logiciel, avec une architecture extensible permettant des évolutions futures.

Introduction

Ce rapport présente le développement et la livraison du livrable 1 du projet EasySave, conformément aux exigences du cahier des charges. EasySave est une application console permettant la gestion de travaux de sauvegarde, conçue pour être utilisée par des utilisateurs francophones et anglophones.

Description du Projet

→ Objectif

Le projet EasySave vise à fournir une solution de sauvegarde efficace et évolutive. La version 1.0 est une application console permettant de :

- Créer jusqu'à cinq travaux de sauvegarde.
- Exécuter des sauvegardes complètes ou différentielles.
- Enregistrer en temps réel l'état d'avancement des travaux et les logs d’exécution.

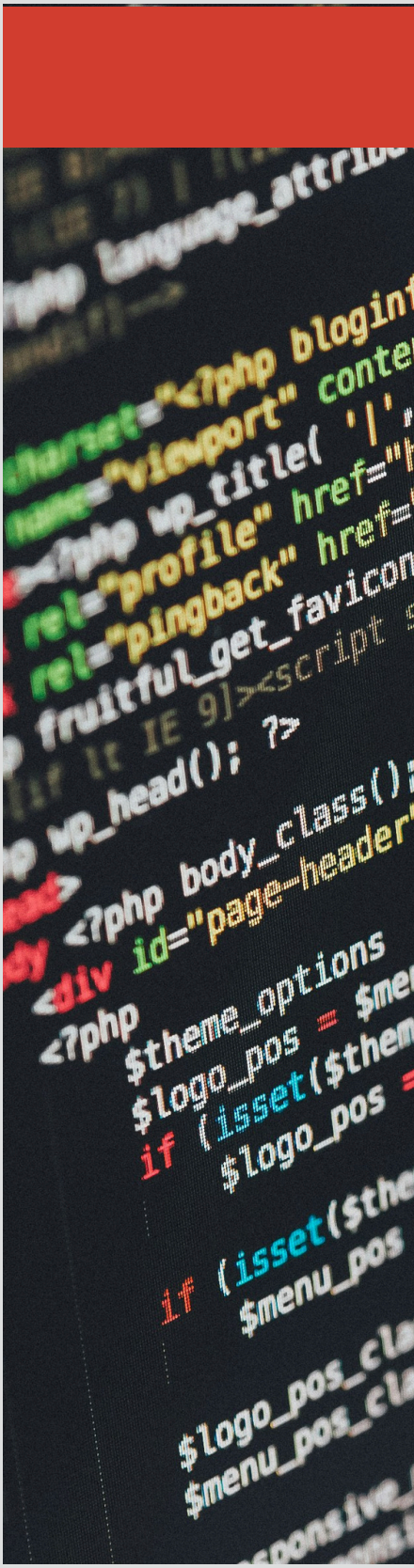
→ Contraintes Techniques

- Développé en C# avec .NET 8.0.
- Compatible avec les disques locaux, externes et lecteurs réseau.
- Interface en ligne de commande.
- Enregistrement des logs et de l’état en format JSON.

Fonctionnalités Développées

- Création et Gestion des Travaux de Sauvegarde
- Gestion des Logs
- Suivi de l'État des Sauvegardes

Architecture et Développement



→ Structure du Code

- Suivi des bonnes pratiques de développement (modularité, maintenabilité, lisibilité).
- Gestion centralisée des fichiers JSON pour éviter les duplications de code.
- Intégration de logs et d'état en temps réel dans des classes dédiées.

→ Gestion de Version

- Utilisation de GitHub pour le suivi des versions et la collaboration.
- Respect des conventions de nommage et bonnes pratiques de développement.

→ Gestion de Projet

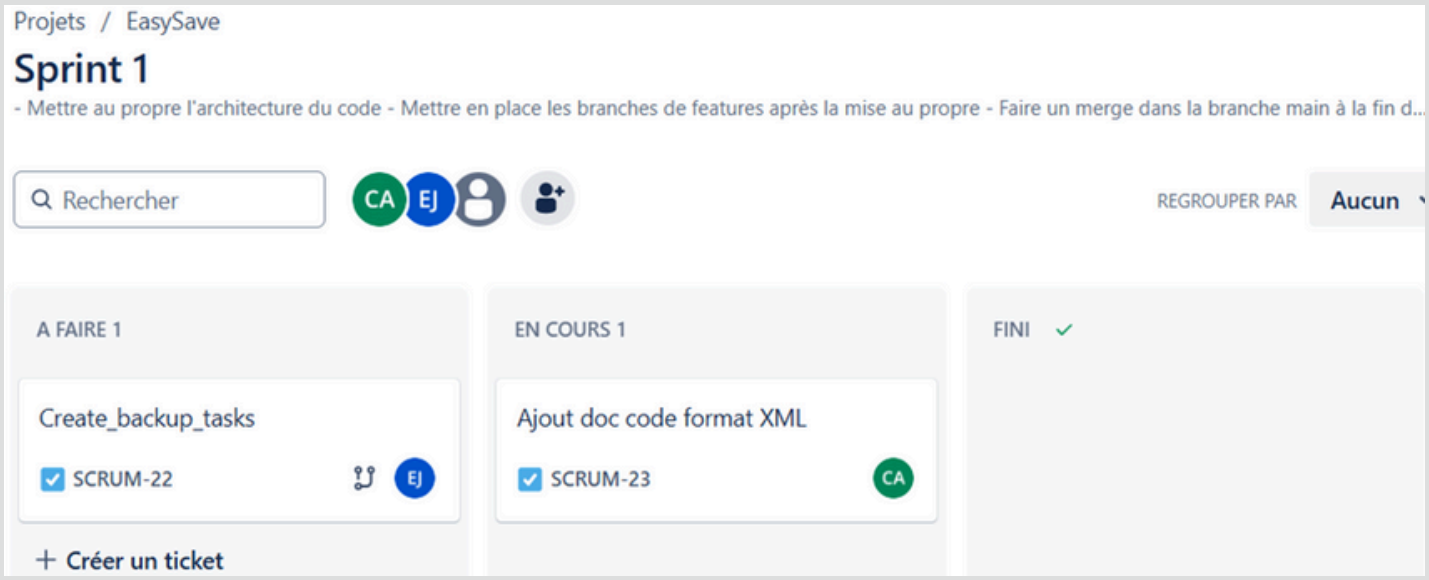
Nous avons adopté une approche agile en mode Scrum afin d'optimiser la gestion du projet et de faciliter le suivi des tâches.

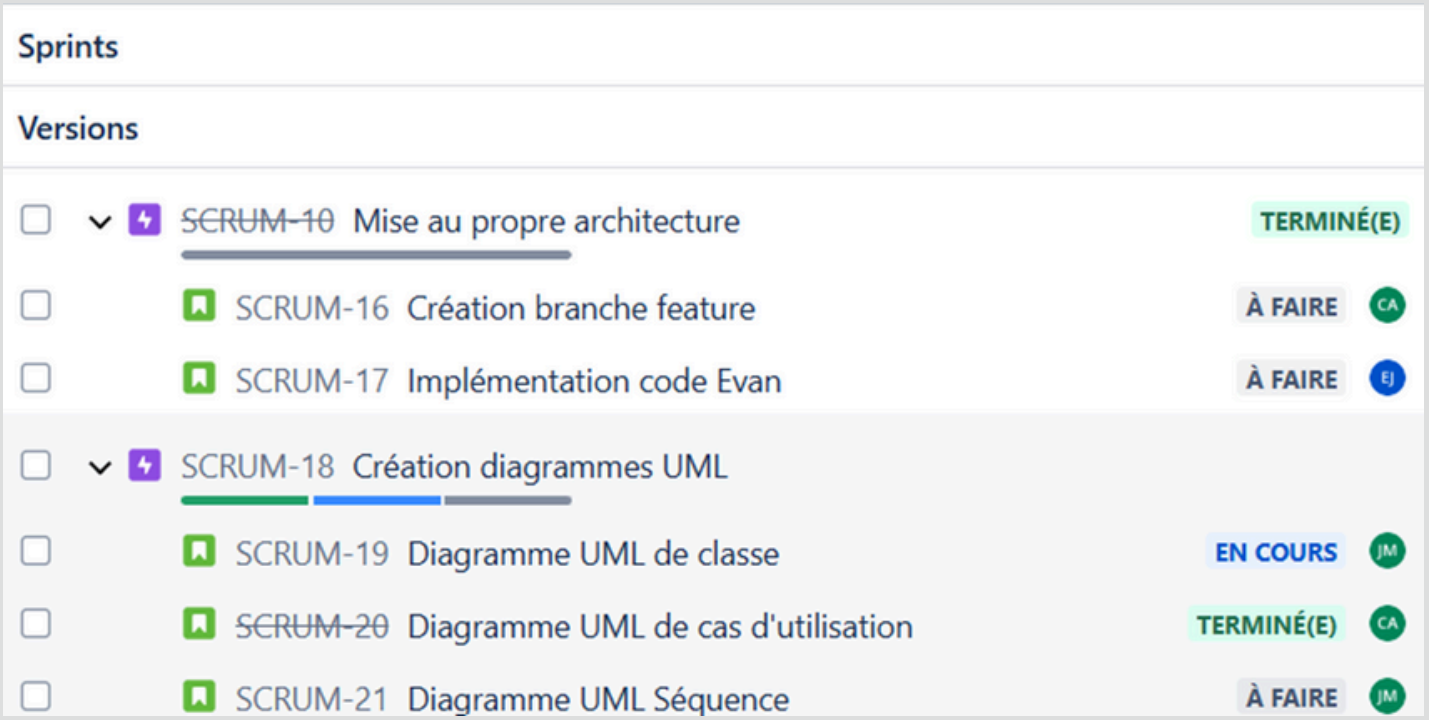
Gestion de projet

L'outil JIRA a été sélectionné pour la gestion de projet en méthodologie agile (Scrum).

Chaque membre est assigné à des tâches sous forme de tickets, permettant un suivi précis du statut des activités :

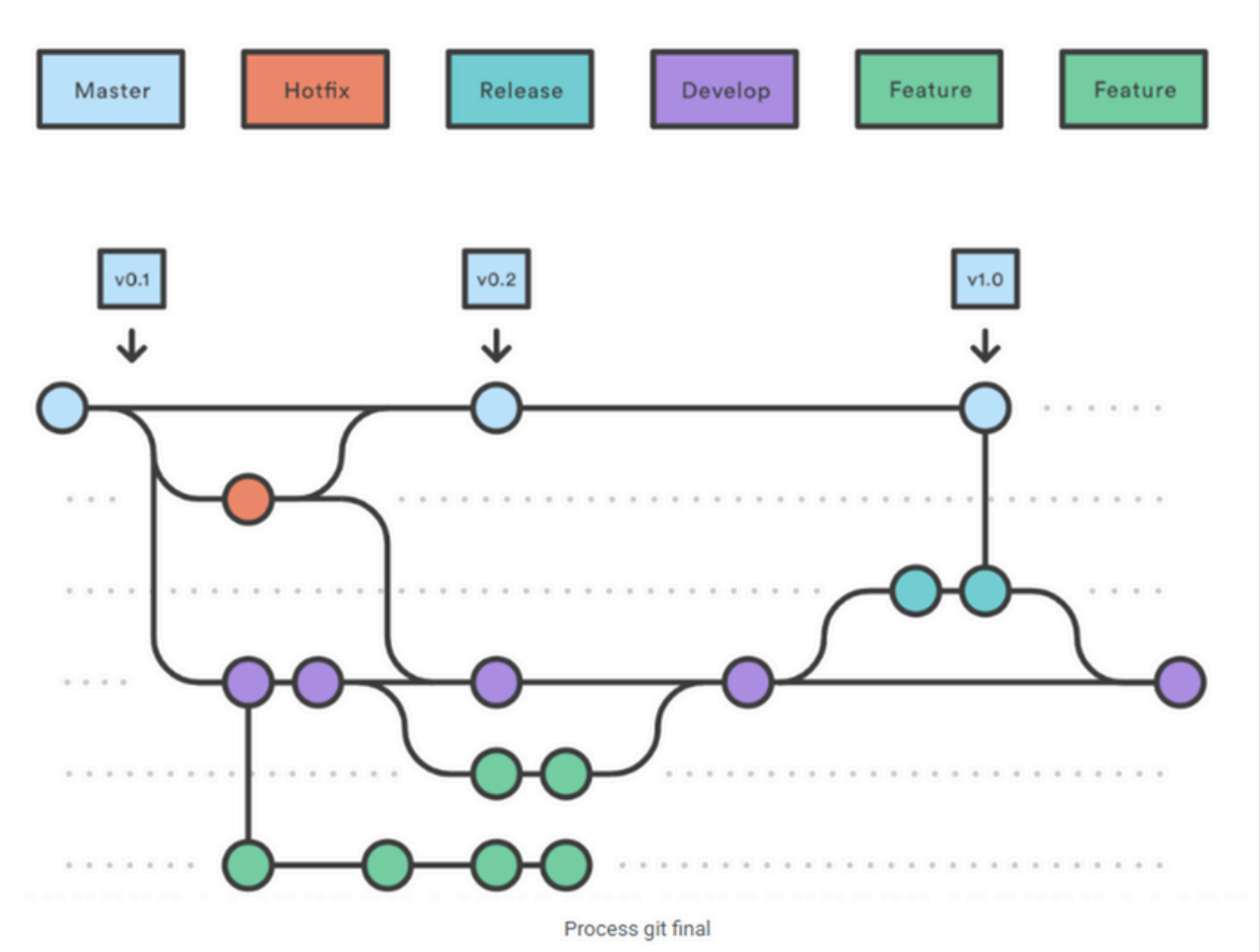
- À faire
- En cours
- Terminée





Gestion de Version

Git nous permet d'augmenter la productivité de l'équipe en utilisant les fonctionnalités de versionning du projet.



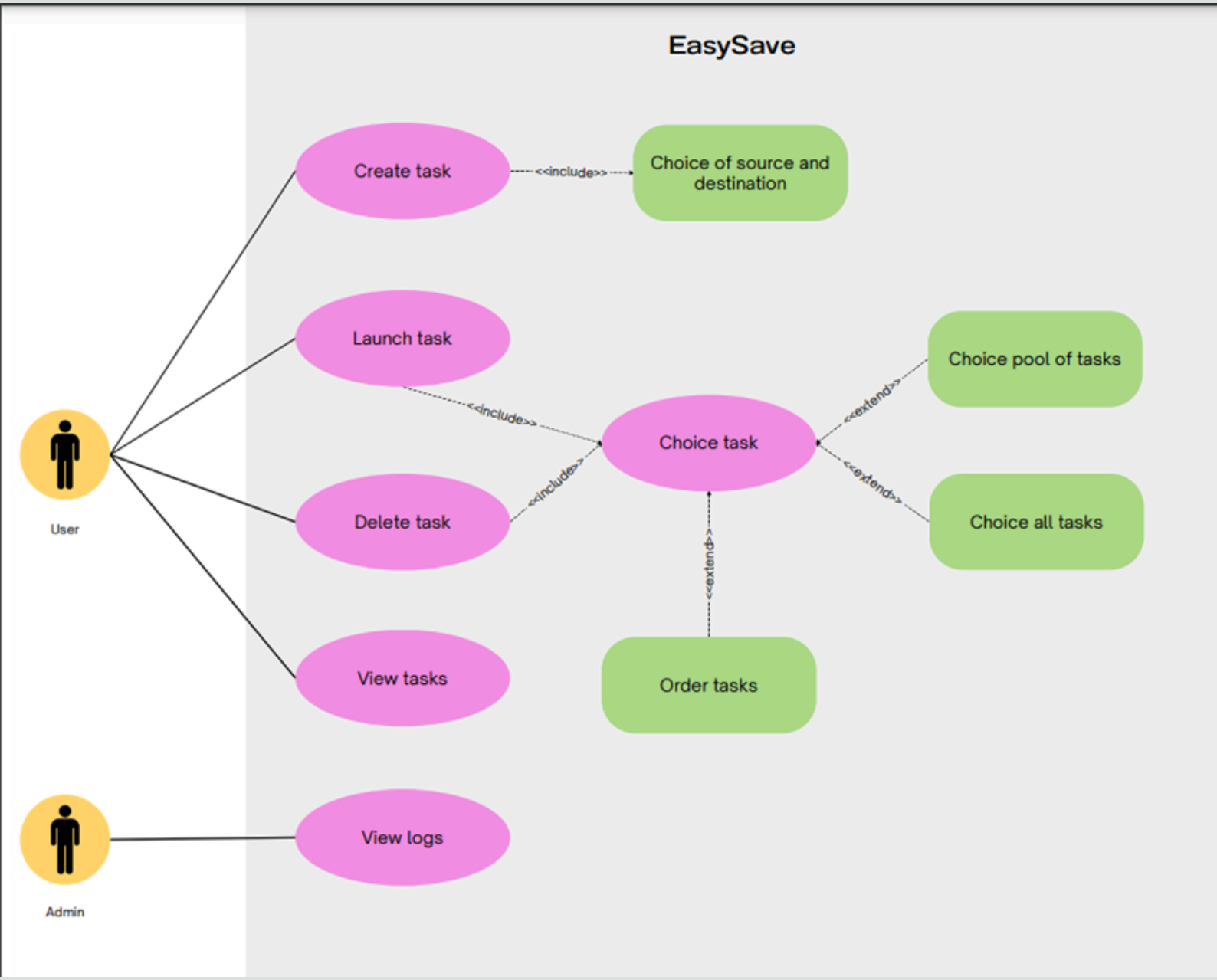
Process git final

Lors du développement, on part de la base du code et on créer de nouvelles branches. La branche main est la branche du code en production. A partir de cette branche, on créer une branche dev, qui contient le code à tester avant de le déployer en production. Depuis la branche dev on créer des branches de features qui contiennent le type, le numéro de ticket et la description, par exemple : **Feature/SCRUM-22-Create_backup_tasks**

Chaque développeur pousse ses modifications dans la branche de feature, on réalise un merge pour fusionner les modifications. Ensuite, on fait un merge de la branche feature dans la branche dev, on fait les tests et s'ils sont validés, on fait le merge dans une branche de release. La branche de release sert à nettoyer le code, ajouter la documentation, mais on ne modifie pas le fonctionnement. Ensuite on pousse dans la branche main avec un tag de numéro de version du type « v1.0 ».

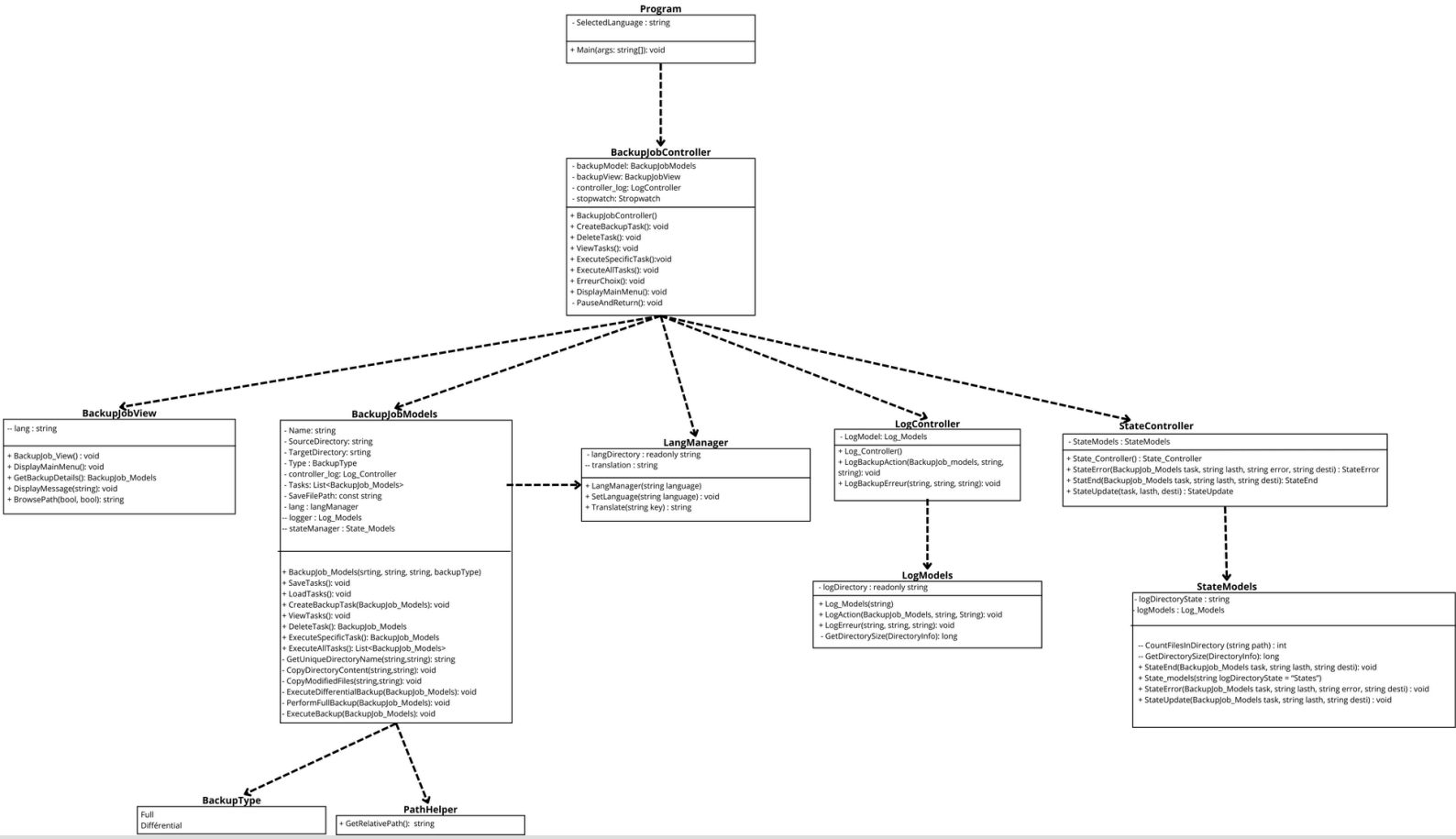
Diagrammes UML

Diagramme de cas d'utilisation



Diagrammes UML

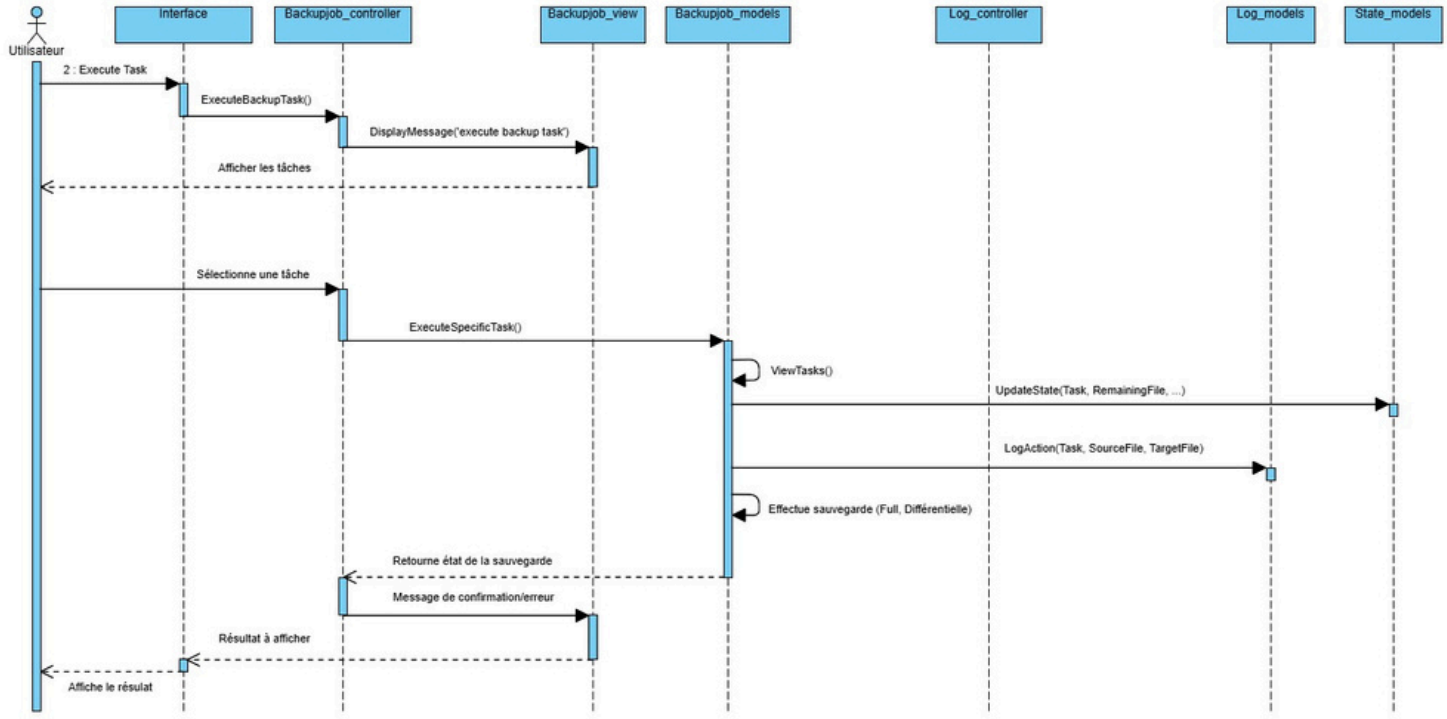
Diagramme de classe



Diagrammes UML

Diagrammes de séquence

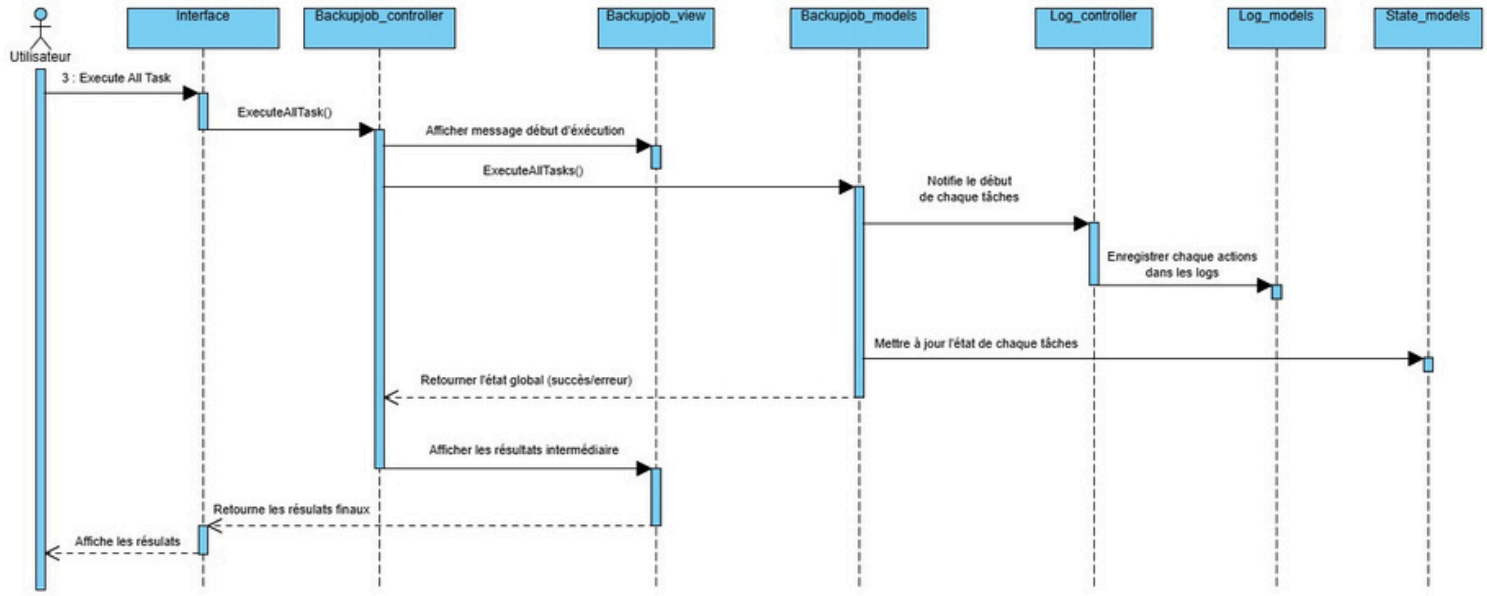
Diagramme de séquence ExecuteTask



Diagrammes UML

Diagrammes de séquence

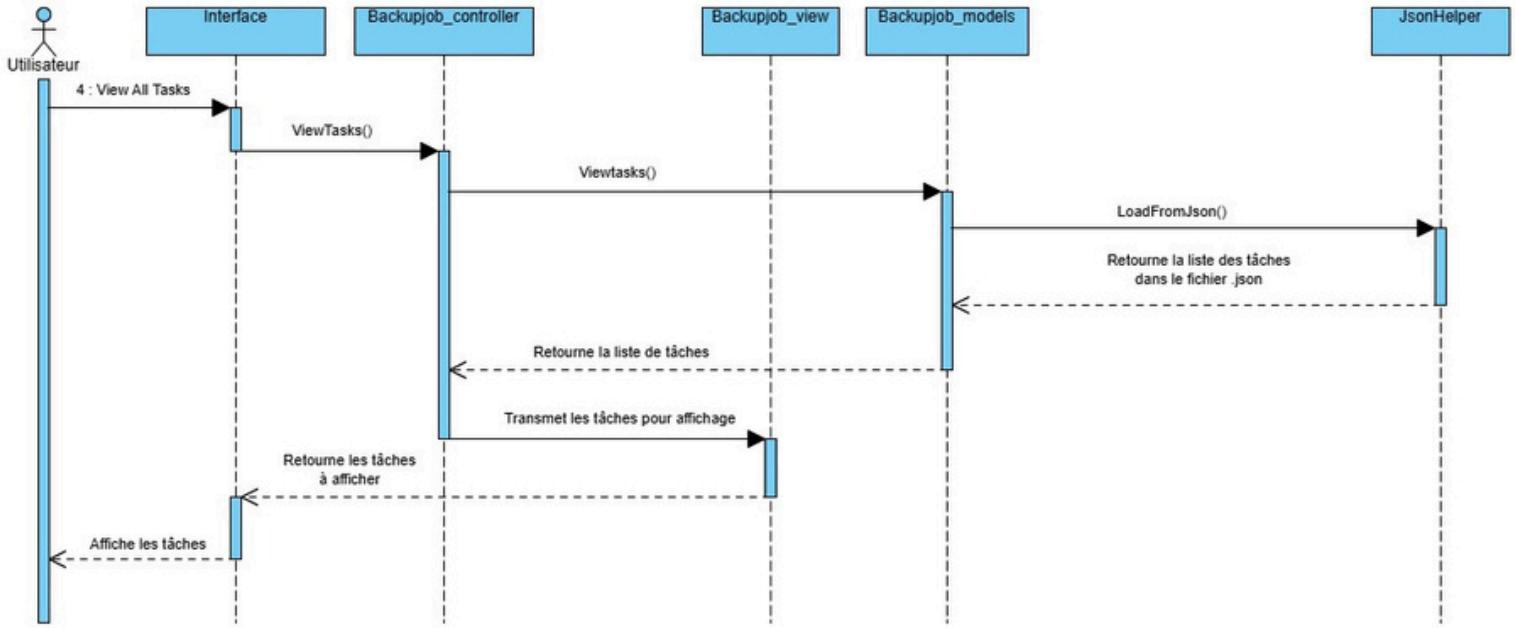
Diagramme de séquence ExecuteAllTasks



Diagrammes UML

Diagrammes de séquence

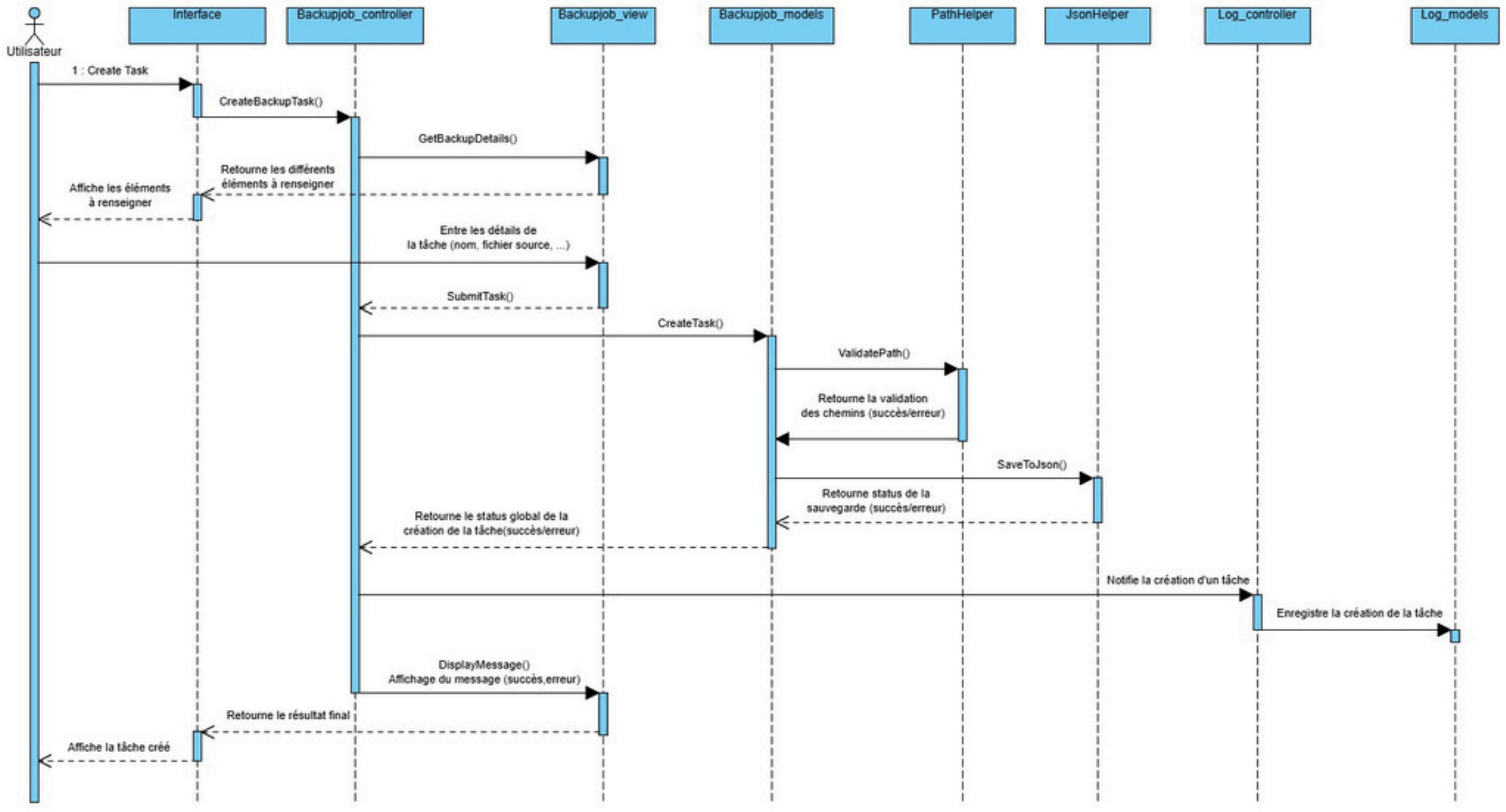
Diagramme de séquence ViewTasks



Diagrammes UML

Diagrammes de séquence

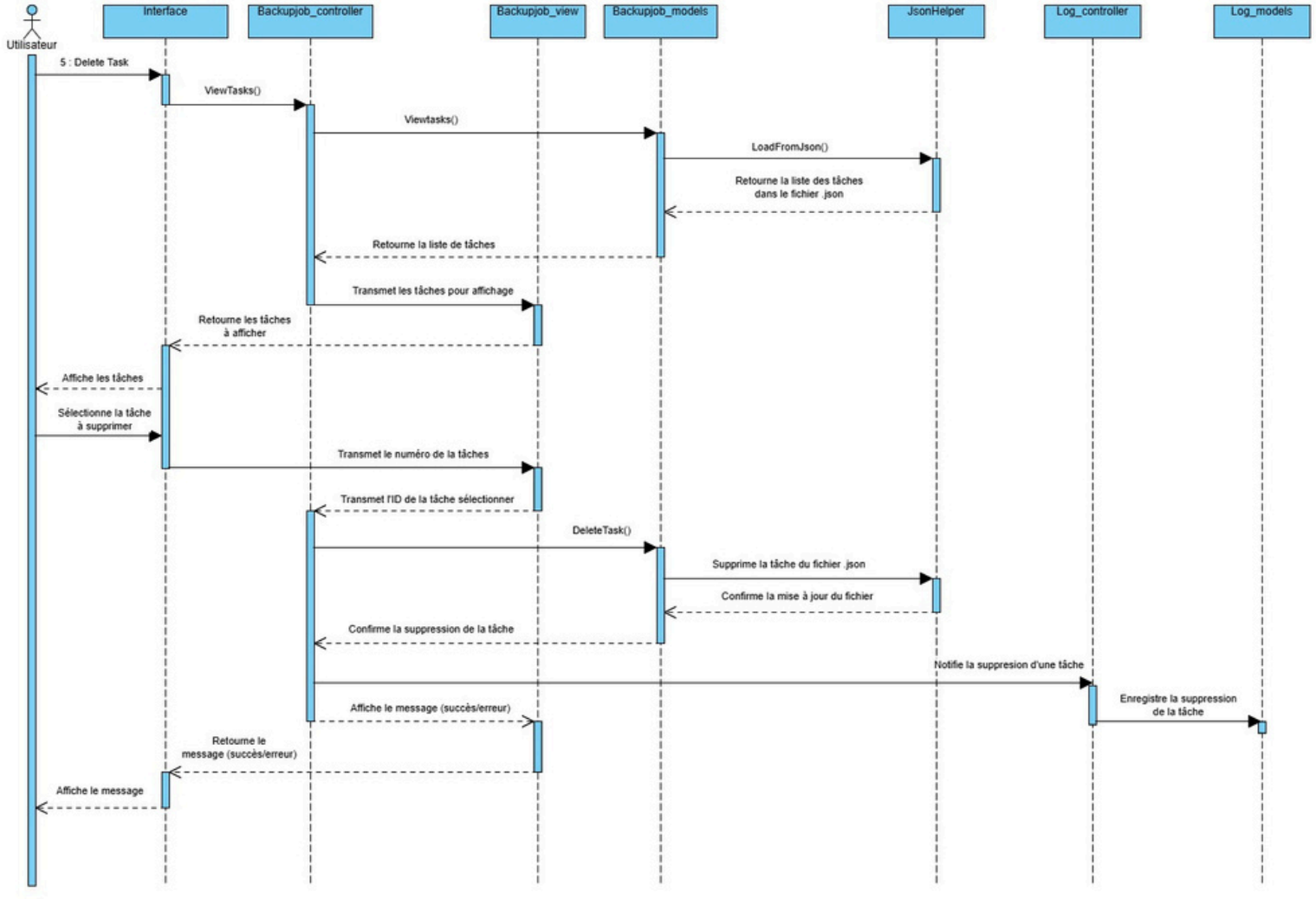
Diagramme de séquence CreateTask



Diagrammes UML

Diagrammes de séquence

Diagramme de séquence DeleteTask



Développement

→ Technologies utilisées

- **Langage : C#**
- **Framework : .NET (Microsoft)**

Le choix de C# et .NET permet de :

- Développer une application portable et performante
- Utiliser la programmation orientée objet (POO) pour structurer le code
- Gérer efficacement la mémoire grâce au garbage collector
- Assurer une persistance des données en utilisant des fichiers JSON
- Exploiter le multithreading pour améliorer la performance des sauvegardes
- Manipuler les fichiers de manière sécurisée avec System.IO

Documentation

→ Documentation technique

- Générée avec Doxygen, elle analyse le code et produit une documentation au format HTML.
- Elle permet aux développeurs de mieux comprendre la structure et le fonctionnement du projet.

→ Documentation utilisateur

- Fournie en format PDF, elle sert de manuel d'utilisation et décrit les fonctionnalités du logiciel, son installation et son mode d'emploi.

Architecture

→ MVC (Modèle-Vue-Contrôleur)

- L'architecture MVC permet de séparer :
  - Le Modèle : gestion des données et logique métier
  - La Vue : interface utilisateur
  - Le Contrôleur : communication entre la vue et le modèle

- Avantages :
  - Séparation claire des responsabilités
  - Facilité de maintenance et évolutivité
  - Modularité accrue

Gestion des logs

→ Objectif des logs

Les logs permettent de tracer les actions des utilisateurs et de faciliter le debugging en cas d'erreur ou d'anomalie.

→ Format des logs (JSON)

Chaque action est enregistrée sous la forme suivante :

```json
{
  "Action": "Create Tasks",
  "Timestamp": "2025-02-04 19:25:40",
  "TaskName": "Backup_test",
  "SourceFile": "D:\\SOURCE",
  "TargetFile": "D:\\DESTINATION",
  "FileSize": 0.0078125,
  "TransferTimeMs": "00:01:03.437"
}
```

Chaque entrée inclut :

- L'action effectuée
- L'horodatage
- Le nom de la tâche
- Les chemins source et destination
- La taille du fichier et le temps de transfert