DOCUMENT TECHNIQUE

EasySave Version 3.0

Équipe projet

- FABO GAËTAN
- ATONLE EDSON
- BOGNING GISSELIN
- KINDA ALEX
- FOGOU JORDAN

Table des matières

1. Introduction

EasySave 3.0 est une évolution majeure apportant l'exécution parallèle des travaux, la gestion des fichiers prioritaires, une interface graphique distante, et un pilotage intelligent des flux réseau. Elle est conçue pour un usage professionnel avec supervision temps réel et fiabilité accrue.

2. Architecture générale

Dans le contexte de notre projet .NET 8, le pattern MVVM (Model-View-ViewModel) est structuré comme suit :

2.1. Model

Les classes du dossier model (ex. : Model.cs, Backup.cs, DataLog.cs, FileInQueue.cs) représentent la logique métier et les données.

Elles gèrent :

•La manipulation des données (sauvegardes, logs, files d'attente…)

•Les accès aux ressources (fichiers, configurations JSON…)

2.2. ViewModel

Le dossier ViewModel contient la classe ViewModel.cs qui fait le lien entre la vue et le modèle.

Il expose :

•Les propriétés et collections à afficher dans la vue

•Les commandes (ICommand) pour réagir aux actions de l'utilisateur

•La logique de présentation (validation, filtrage, etc.)

2.3. View

Le dossier view contient les fichiers XAML (ex. : MainWindow.xaml) qui définissent l'interface utilisateur.

La vue :

•Se lie au ViewModel via le DataContext

•Utilise le data binding pour afficher et modifier les données

•Ne contient aucune logique métier

Ressources

Les fichiers du dossier Resources (ex. : LimitSize.json, PriorityExtensions.json, en-GB.xaml, fr-FR.xaml) servent à la configuration, la localisation et la personnalisation de l'application.

Résumé du MVVM dans ce projet :

•Model : gestion des données et logique métier

•ViewModel : interface entre la vue et le modèle, expose les données et commandes

•View : interface graphique liée au ViewModel, sans logique métier

Cette séparation permet une application modulaire, testable et maintenable.

3. Composants

3.1. Model (Dossier model)

Les classes du dossier model représentent la logique métier et les données de l'application.

Exemples dans ton projet :

- DataLog.cs : gestion des logs ou historiques.
- Model.cs : logique métier principale ou gestion des données globales.
- Backup.cs : gestion des sauvegardes.
- FileInQueue.cs : gestion des fichiers en file d'attente.

3.2. ViewModel (Dossier ViewModel)

Le ViewModel fait le lien entre la vue et le modèle. Il expose les données et les commandes que la vue peut utiliser, tout en encapsulant la logique de présentation.

•ViewModel.cs : contient les propriétés, commandes et la logique de liaison entre la vue et les modèles.

3.3. View (Dossier view)

La vue est responsable de l'affichage. Elle se lie au ViewModel via le data binding.

•MainWindow.xaml : interface utilisateur principale, liée à un ViewModel via le DataContext.

Ressources et configuration

•Resources\LimitSize.json, PriorityExtensions.json : fichiers de configuration ou de ressources.

•Resources\en-GB.xaml, fr-FR.xaml : ressources pour la localisation.

•App.xaml : configuration globale de l'application (styles, ressources, etc.).

4. Formats de données

- backupList.json : liste des travaux définis (nom, source, cible, type)
- state.json : état temps réel (nom, progression %, timestamp)
- DailyLogs/YYYY-MM-DD.json ou .xml : détails par fichier (source, cible, taille, transfert, cryptage)

5. Fonctionnalités avancées

- Sauvegardes en parallèle : gain de temps, exécution simultanée
- Play / Pause / Stop sur chaque tâche ou globalement
- Gestion des fichiers prioritaires : interdiction de traiter des fichiers non prioritaires s'il en reste à traiter
- Blocage de transferts simultanés de gros fichiers (taille > n Ko)
- Option de réduction automatique si la charge réseau est trop élevée

6. Console déportée

- Interface graphique distante synchronisée avec la machine hôte
- Affiche l'état des travaux + actions à distance (via sockets)
- Requiert configuration réseau minimale (ports ouverts)

7. Cryptage et CryptoSoft Mono-Instance

- Seuls les fichiers avec extensions autorisées sont cryptés
- CryptoSoft ne peut être lancé qu'en une seule instance à la fois
- Une file d'attente est gérée automatiquement
- Le log indique :

➢ 0 si pas de cryptage

➢ >0 durée en millisecondes

➢ <0 code erreur

8. Gestion des erreurs et du logiciel métier

- Si un logiciel métier (ex : calc.exe) est actif :

➢ Tous les travaux sont mis en pause automatique

➢ Reprise automatique à la fermeture du logiciel

- Tous les événements sont consignés dans le log
- Messages utilisateur clairs sur interface

9. Build, déploiement et configuration réseau

Compilation : dotnet build

Structure recommandée : sql

Solution 'EasySaveApp'

│

├─ EasySaveApp (projet principal WPF + MVVM)

│ ├─ model

│ │ ├─ Backup.cs

│ │ ├─ DataState.cs

│ │ ├─ DataLog.cs

│ │ ├─ FileCompare.cs

│ │ ├─ FileInQueue.cs

│ │ ├─ JailAppsFormat.cs

│ │ ├─ Model.cs

│ │ ├─ PriorityFormat.cs

│ ├─ Resources

│ │ ├─ CryptoSoft\CryptoSoft.exe

│ │ ├─ CryptExtension.json

│ │ ├─ JailApps.json

│ │ ├─ LimitSize.json

│ │ ├─ PriorityExtensions.json

│ │ ├─ en-GB.xaml

│ │ ├─ fr-FR.xaml

│ ├─ Socket

│ │ ├─ Connection.cs

│ │ ├─ ConnectionPool.cs

│ │ ├─ MessageContent.cs

│ │ └─ Server.cs

│ ├─ view

│ │ ├─ StateObject.cs

│ │ ├─ ViewModel.cs

│ │ └─ MainWindow.xaml / MainWindow.xaml.cs

│ ├─ App.xaml / App.config / AssemblyInfo.cs

│ └─ EasySaveLogo.ico

│

└─ EasySaveConsole (projet console qui se connecte en TCP au serveur intégré dans EasySaveApp)

├─ App.xaml / AssemblyInfo.cs

└─ MainWindow.xaml / MainWindow.xaml.cs (ou tout formulaire console utilisant une socket)

Bonnes pratiques :

- Ne pas versionner : /bin/, /obj/, /DailyLogs/
- Prévoir une **IP fixe ou nom de machine** pour la console distante
- **Ouvrir les ports TCP nécessaires** (ex : 8888) sur le pare-feu