Projet : EASYSAVE

Livrable 2 :
EasySave version 2.0

Romain HEMART - Noah VOISIN - Cyr-Manuel DJOKI - Alban GODIER

Les fichiers sources sont disponibles dans le dossier ‘Doc’ du dépôt GitHub du projet.

Table des matières

Table des figures 3
Diagramme de classes 4
Diagramme d'activité 12
Diagramme de séquence 14
Initialisation 14
Exécution des travaux de sauvegarde 15
Modification de la langue 16
Diagramme cas d'utilisation 17
Diagramme de composant 19

Table des figures

Figure 1 : Diagramme de classes - Vue d'ensemble 5
Figure 2 : Diagramme de classes - Journalisation 6
Figure 3 : Diagramme de classes - Gestion de la langue 7
Figure 4 : Diagramme de classes - Configuration 8
Figure 5 : Diagramme de classes - Exécution des travaux de sauvegarde 9
Figure 6 : Diagramme de classes - Suivi de l'état des travaux de sauvegarde 11
Figure 7 : Diagramme d'activité - Lancement d'un travail de sauvegarde - Partie 1 12
Figure 8 : Diagramme d'activité - Lancement d'un travail de sauvegarde - Partie 2 13
Figure 9 : Diagramme de séquence – Initialisation 15
Figure 10 : Diagramme de séquence - Exécution des travaux de sauvegarde 16
Figure 11 : Diagramme de séquence - Modification de la langue 16
Figure 12 : Diagramme de cas d'utilisation - Partie 1 17
Figure : Diagramme de cas d'utilisation – Partie 2 18
Figure : Diagramme de composant 19

Diagramme de classes

Ce diagramme de classes représente les différentes classes du projet ainsi que leurs relations. Il est divisé en plusieurs sections :

- **View** : Cette section contient les classes qui gèrent l'affichage des données à l'utilisateur.
- **ViewModel** : Cette section contient les classes qui gèrent la logique de présentation des données.
- **Model (Tout le reste)** : Cette section contient les classes qui gèrent les données et la logique métier de l'application.
    - **Logger** : Cette sous-partie contient les classes qui gèrent la journalisation des événements et des erreurs dans un fichier de log.
    - **Language** : Cette sous-partie contient les classes qui gèrent la langue de l'application : changement de langue, traduction des chaînes de caractères, etc.
    - **Configuration** : Cette sous-partie contient les classes qui gèrent la configuration : Langue et liste des travaux de sauvegarde.
    - **Job** : Cette sous-partie contient les classes qui gèrent l'exécution des travaux de sauvegarde.
    - **State** : Cette sous-partie contient les classes qui assurent le suivi de l'état des travaux de sauvegarde.

L'ensemble des classes présentées ci-dessous sont des interfaces C#. Elles seront implémentées par des classes concrètes dans le projet.

Tout d'abord, pour avoir une vue d'ensemble du diagramme de classes, voici une représentation simplifiée en réduisant les sous-parties à des composants :

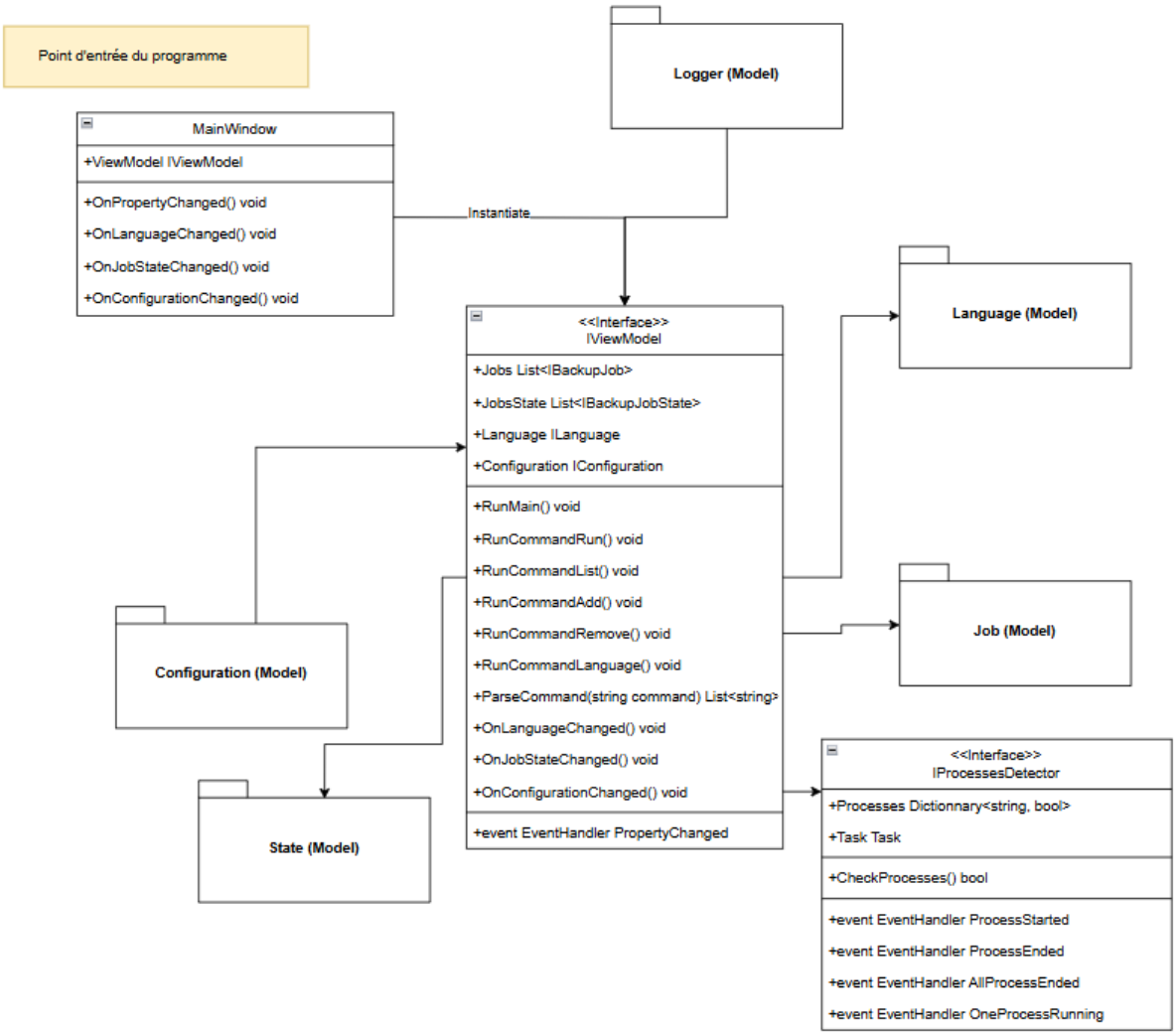

Figure 1 : Diagramme de classes - Vue d'ensemble

- **MainWindow** : Cette classe représente la vue principale de l'application dans le contexte WPF. Elle est responsable de l'interaction des utilisateurs avec l'interface graphique.
- **IViewModel** : Cette interface représente le modèle de présentation de l'application. Elle est responsable de la logique de présentation des données et de la gestion des interactions entre la vue et le modèle. Elle suit les événements remontés par le modèle et les transmet à la vue.
- **IProcessesDetector** : Cette interface permet de suivre en temps réel l'exécution de processus métier et de stopper les travaux de sauvegarde en cas d'exécution de ces processus.

Par la suite, on réutilisera la représentation des sous-parties sous forme de composants pour détailler chaque partie.

Logger :

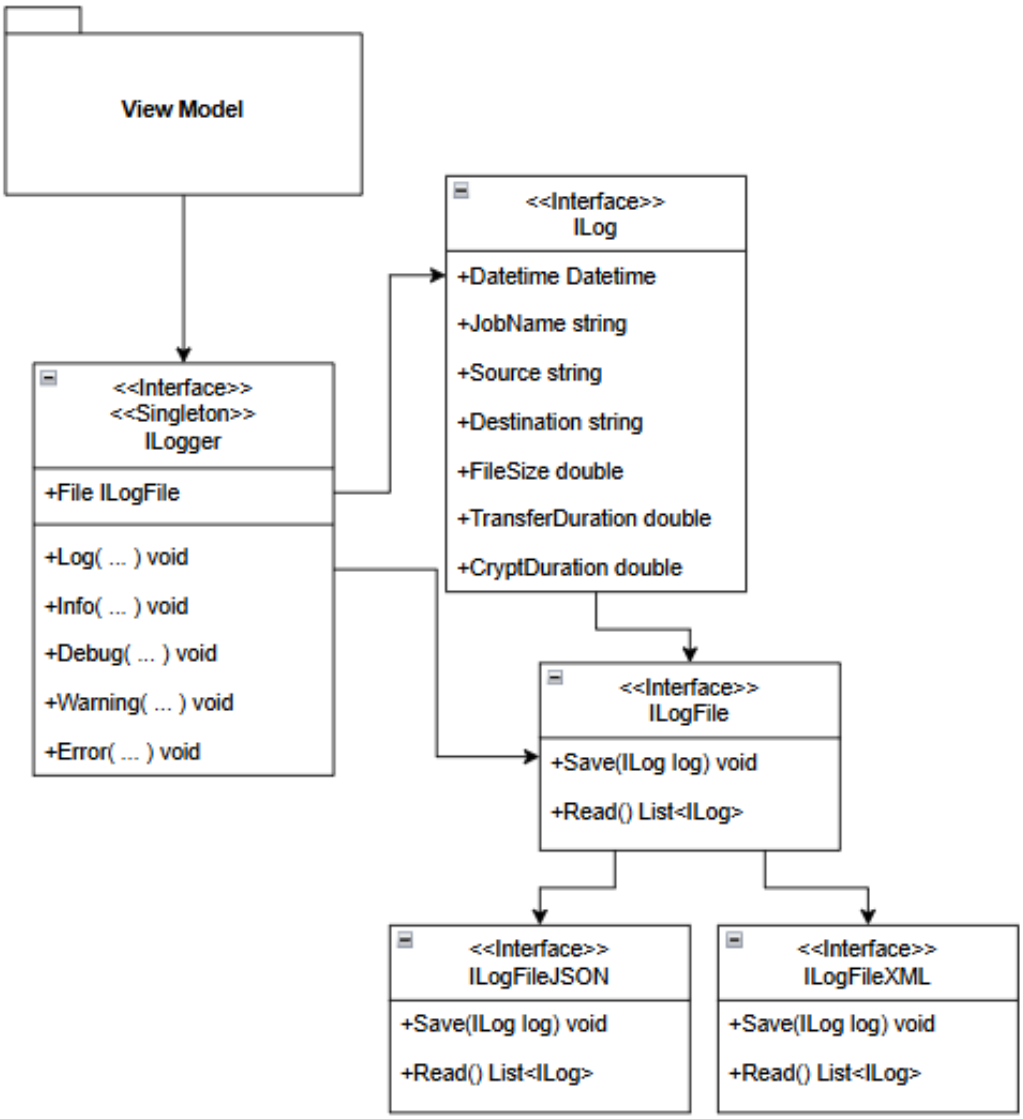

Figure 2 : Diagramme de classes - Journalisation

- **ILogger** : Cette interface représente le logger de l'application. Elle est responsable de la journalisation des événements et des erreurs dans un fichier de log. Cette interface représente un singleton et ne doit pas être instanciée plusieurs fois. Elle est destinée à être utilisée par toutes les classes de l'application pour enregistrer des événements et des erreurs.
- **ILog** : Cette interface représente un message de log et contient les propriétés qui seront utilisées pour enregistrer un message de log. Elle contient les propriétés suivantes :
    - **Datetime** : La date et l'heure de l'événement.
    - **JobName** : Le nom du travail de sauvegarde associé à l'événement.
    - **Source** : Le fichier ou le dossier source associée à l'événement.
    - **Destination** : Le fichier ou le dossier de destination associée à l'événement.
    - **FileSize** : La taille du fichier associée à l'événement.
    - **TransferDuration** : La durée du transfert associée à l'événement.

- **ILogFile** : Cette interface fait la liaison entre le logger et le fichier de log. Elle est responsable de l'écriture des messages de log dans le fichier de log.
- **ILogFileXML** : Cette interface fait la liaison entre le logger et le fichier de log. Elle est responsable de l'écriture des messages de log dans le fichier de log au format XML.
- **ILogFileJSON** : Cette interface fait la liaison entre le logger et le fichier de log. Elle est responsable de l'écriture des messages de log dans le fichier de log au format JSON.

Gestion de la langue :

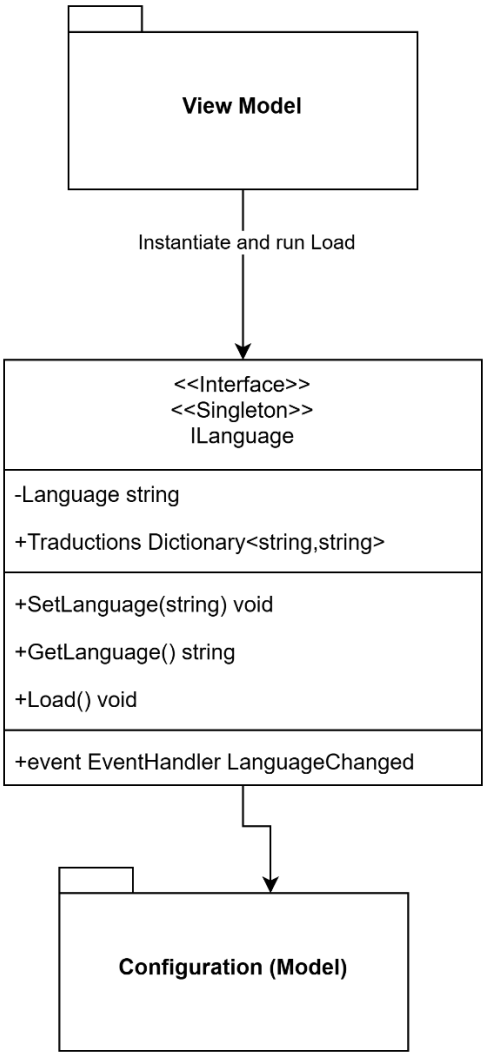

Figure 3 : Diagramme de classes - Gestion de la langue

- **ILanguage** : Cette interface représente la langue de l'application. Elle est responsable de la gestion de la langue de l'application et de la traduction des chaînes de caractères. Lors du changement de langue est déclenche un événement qui est remonté au ViewModel.

Configuration :

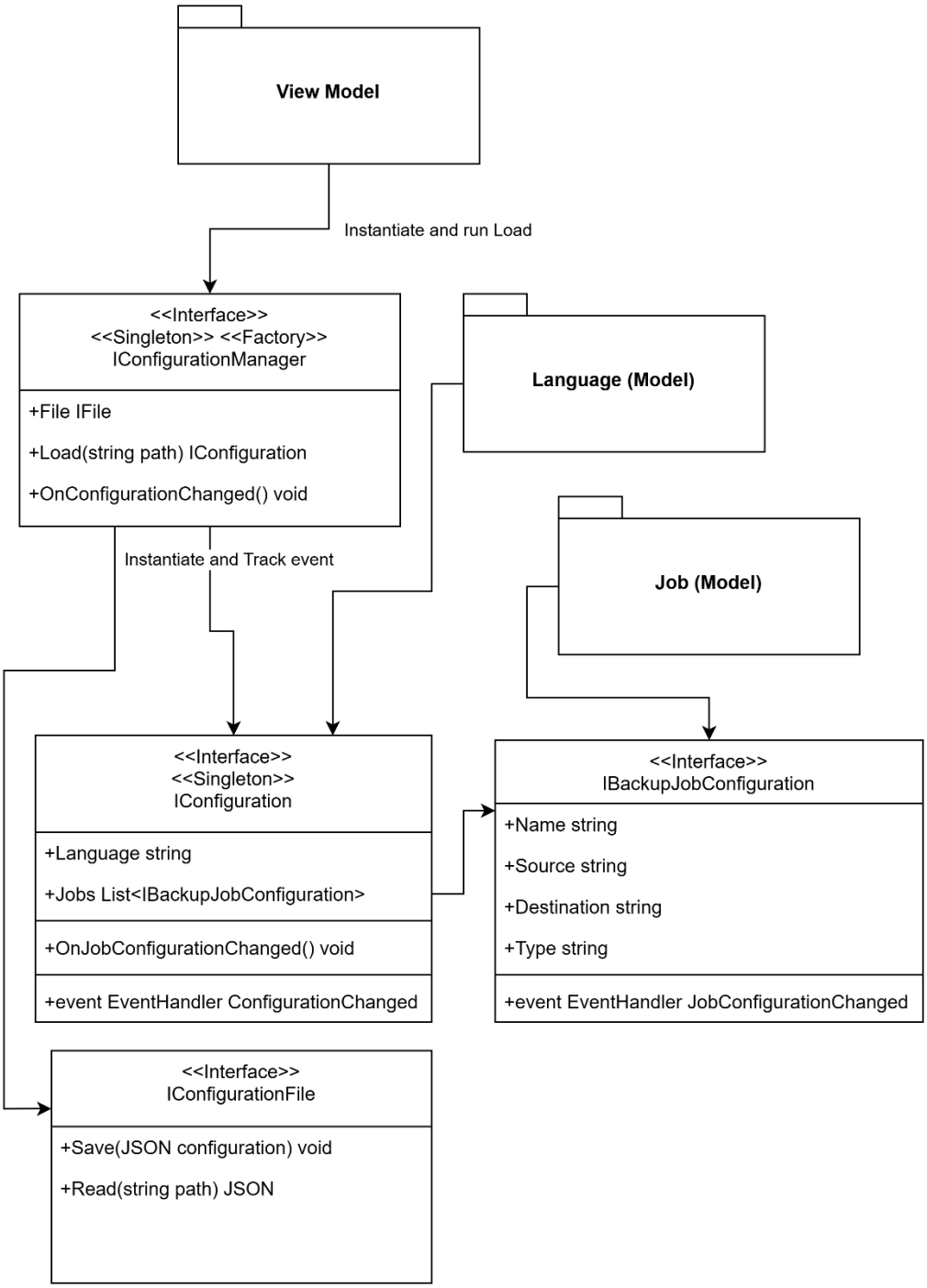

Figure 4 : Diagramme de classes - Configuration

- **IConfiguration** : Cette interface représente la configuration de l'application. Elle ne contient que les propriétés enregistrées dans le fichier de configuration. Lors de la modification d'une de ces propriétés, un événement est déclenché et remonté à la classe ConfigurationManager.
- **IBackupJobConfiguration** : Cette interface représente la configuration d'un travail de sauvegarde. Elle contient les propriétés suivantes :
    - **Name** : Le nom du travail de sauvegarde.

- **Source** : Le fichier ou le dossier source à sauvegarder.
- **Destination** : Le fichier ou le dossier de destination de la sauvegarde.
- **Type** : Le type de sauvegarde (complète ou séquentielle).

- **IConfigurationManager** : Cette interface représente le gestionnaire de configuration de l'application. Elle est responsable de la gestion de la configuration de l'application et de la sauvegarde des données dans le fichier de configuration. Elle est responsable de la gestion des événements remontés par les classes Configuration et Language.
- **IConfigurationFile** : Cette interface fait la liaison entre le gestionnaire de configuration et le fichier de configuration. Elle est responsable de la lecture et de l'écriture des données dans le fichier de configuration.

Exécution des travaux de sauvegarde :

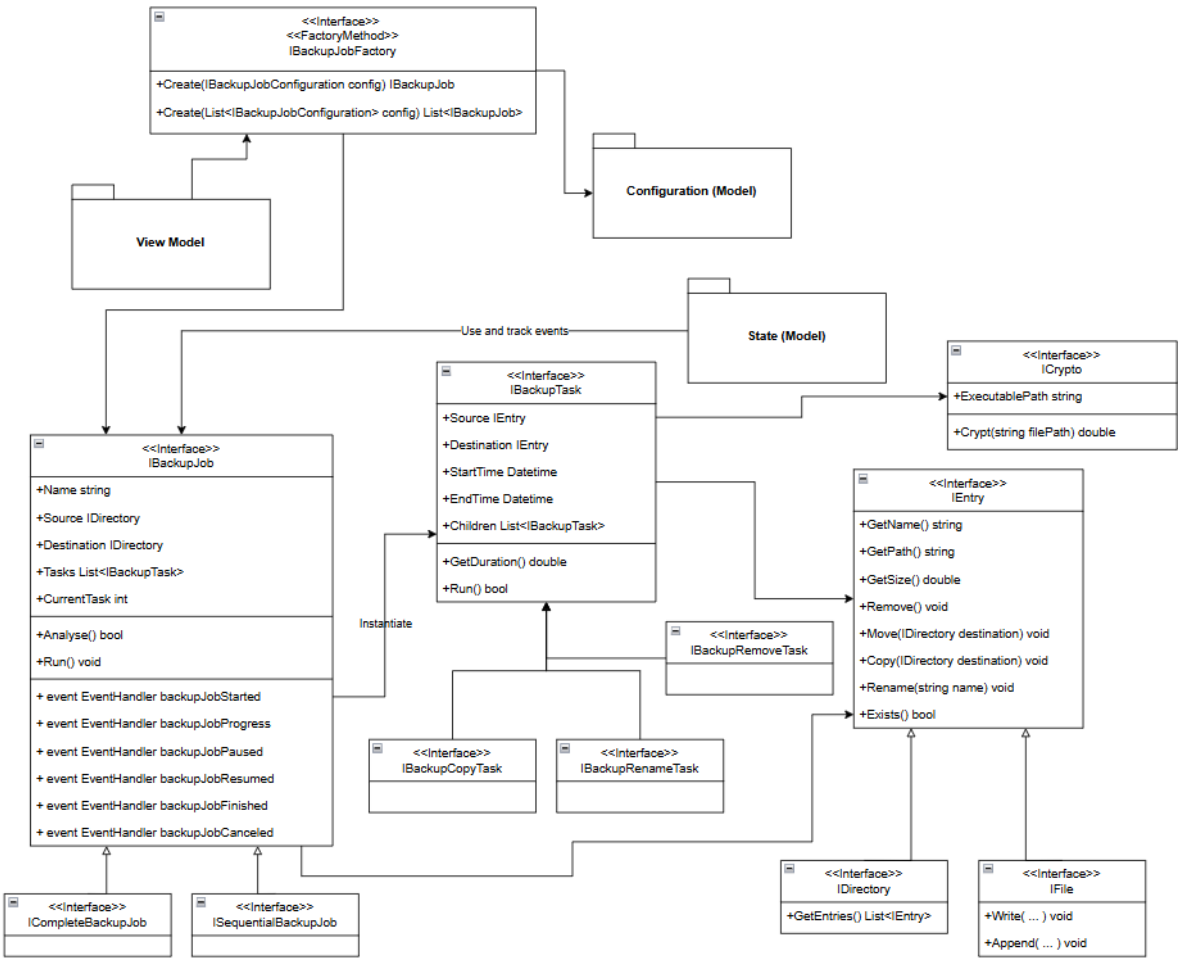

Figure 5 : Diagramme de classes - Exécution des travaux de sauvegarde

- **IBackupJobFactory** : Cette interface représente la fabrique de travaux de sauvegarde. Elle possède une méthode qui crée un travail de sauvegarde à partir d'une configuration.
- **IBackupJob** : Cette interface représente un travail de sauvegarde. Elle est responsable de l'exécution du travail de sauvegarde et du déclenchement des événements associés à la progression du travail :

    - **Started** : Déclenché lorsque le travail de sauvegarde démarre.
    - **Progress** : Déclenché lorsque le travail de sauvegarde progresse.
    - **Finished** : Déclenché lorsque le travail de sauvegarde est terminé.
    - **Cancelled** : Déclenché lorsque le travail de sauvegarde est annulé.
    - **Paused** : Déclenché lorsque le travail de sauvegarde est mis en pause.
    - **Resumed** : Déclenché lorsque le travail de sauvegarde est repris.
- **ICompleteBackupJob** : Cette interface représente un travail de sauvegarde de type complet (tous les fichiers et dossiers sont sauvegardés)
- **ISequentialBackupJob** : Cette interface représente un travail de sauvegarde de type séquentiel (seuls les fichiers et dossiers modifiés sont sauvegardés)
- **IBackupTask** : Cette interface représente une tâche de sauvegarde. Une tâche est une action réalisée sur un fichier ou un dossier comme la suppression, la copie ou le renommage.
- **IBackupCopyTask** : Cette interface représente une tâche de sauvegarde de type copie. Elle est responsable de la copie d'un fichier ou d'un dossier.
- **IBackupRemoveTask** : Cette interface représente une tâche de sauvegarde de type suppression. Elle est responsable de la suppression d'un fichier ou d'un dossier.
- **IBackupRenameTask** : Cette interface représente une tâche de sauvegarde de type renommage. Elle est responsable du renommage d'un fichier ou d'un dossier.
- **IEntry** : Cette interface représente un élément dans un dossier (dossier ou fichier).
- **IFile** : Cette interface représente un fichier.
- **IDirectory** : Cette interface représente un dossier.

Fonctionnalités apportées avec la version 2.0 :

- **ICrypto** : Cette interface va appeler le logiciel CryptoSoft qui est un logiciel qui permet de chiffrer un fichier via un XOR

Suivi de l’état des travaux de sauvegarde :

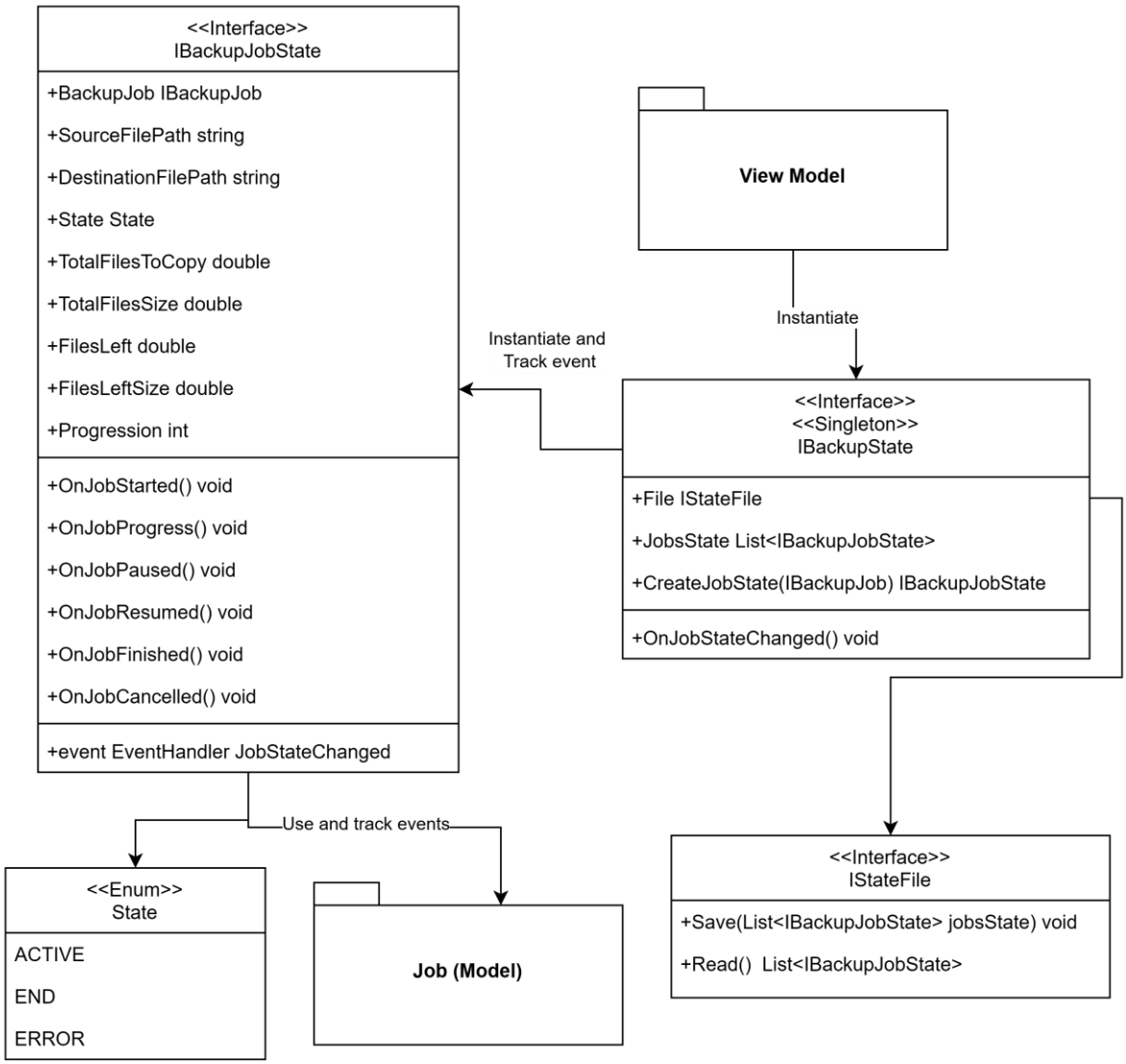

Figure 6 : Diagramme de classes - Suivi de l'état des travaux de sauvegarde

- **IBackupJobState** : Cette interface représente l'état d'un travail de sauvegarde. Elle assure le suivi des étapes du travail de sauvegarde (grâce à des événements) et de l'état du travail de sauvegarde (en cours, terminé, annulé, en pause, repris).
- **IBackupState** : Cette interface représente l'état global des travaux de sauvegarde. Elle suit l'état de tous les travaux de sauvegarde et déclenche des événements lorsque l'état d'un travail de sauvegarde change.
- **IStateFile** : Cette interface fait la liaison entre l'état des travaux de sauvegarde et le fichier d'état. Elle est responsable de la lecture et de l'écriture des données dans le fichier d'état.

Diagramme d’activité

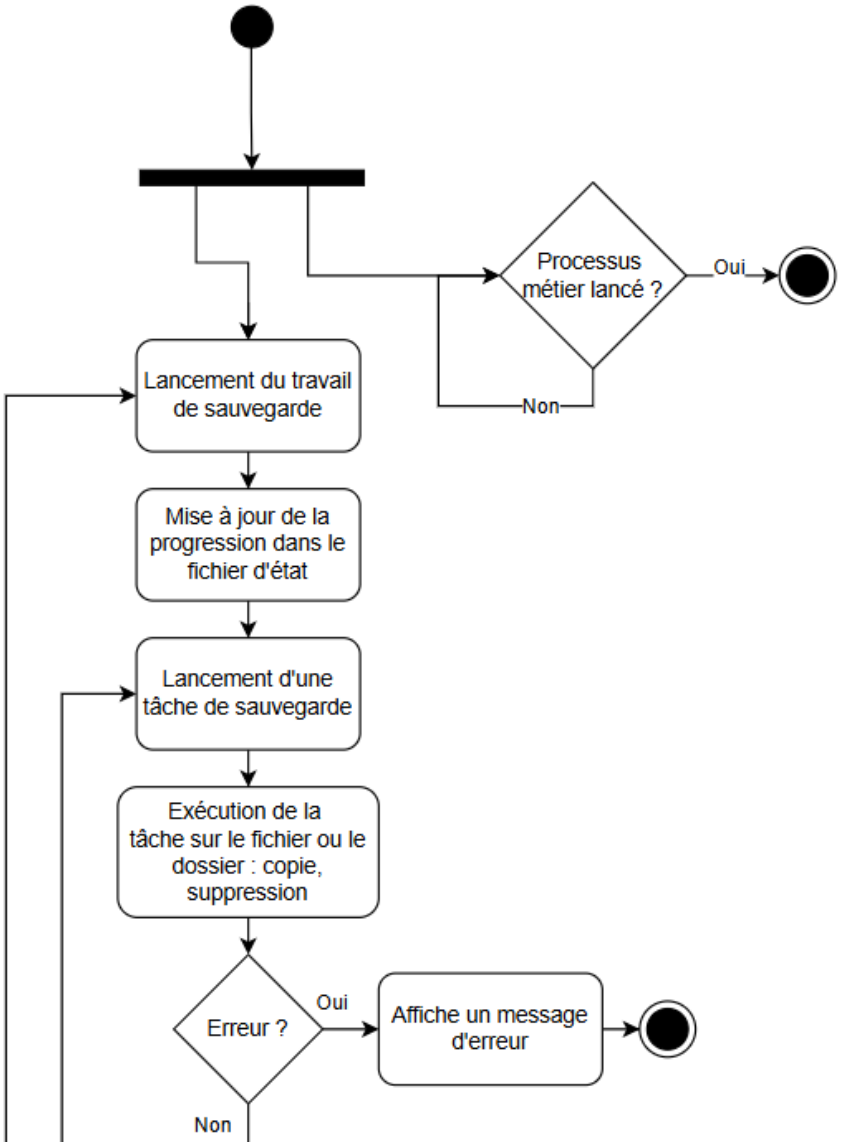

Figure 7 : Diagramme d'activité - Lancement d’un travail de sauvegarde - Partie 1

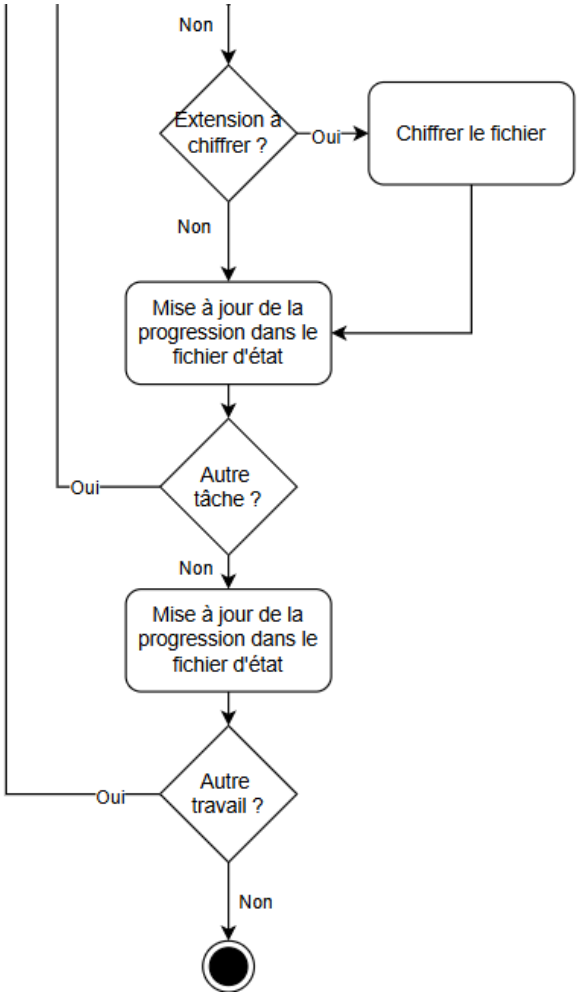

Figure 8 : Diagramme d'activité - Lancement d'un travail de sauvegarde - Partie 2

L'objectif d'un diagramme d'activité est de décrire un fonctionnement logique précis d'une tâche. Cela se fait par une syntaxe bien précise avec les rectangles qui représentent les actions, les losanges qui représentent les choix et les points noirs qui représentent le début et les points noirs entourés pour la fin.

Ce diagramme déroule le fonctionnement d'un lancement d'un travail de sauvegarde. Donc, le travail démarre, et chaque tâche est exécutée successivement, avec une mise à jour régulière de l'état d'avancement dans un fichier dédié. Après chaque action, le système vérifie la présence d'erreurs, puis s'assure qu'il reste d'autres tâches ou travaux à traiter. Le cycle se répète jusqu'à la fin complète de toutes les sauvegardes. Dès qu'une erreur est repérée, la tâche s'arrête.

Fonctionnalités apportées avec la version 2.0 :

Le système de chiffrement d'extension de fichier (au choix de l'utilisateur) sera effectué après chaque exécution, tandis que l'arrêt du système lié aux processus métier est surveillé en permanence et intervient dès le lancement de ceux-ci.

Diagramme de séquence

Initialisation

Ce diagramme de séquence représente les actions effectuées de l'initialisation de l'application jusqu'à l'exécution des différentes commandes.

Les commandes sont au nombre de 5 :

- **Run** : Commande permettant d'exécuter des travaux de sauvegarde.
- **List** : Commande permettant d'afficher une liste des travaux de sauvegarde enregistrés dans la configuration.
- **Add** : Commande permettant d'ajouter un travail de sauvegarde à la configuration (seulement 5 travaux de sauvegarde peuvent être enregistré au maximum)
- **Remove** : Commande permettant de supprimer un travail de sauvegarde à partir de sa position dans la configuration (à partir de 1) ou de son nom ?
- **Language** : Commande permettant de changer la langue de l'application.

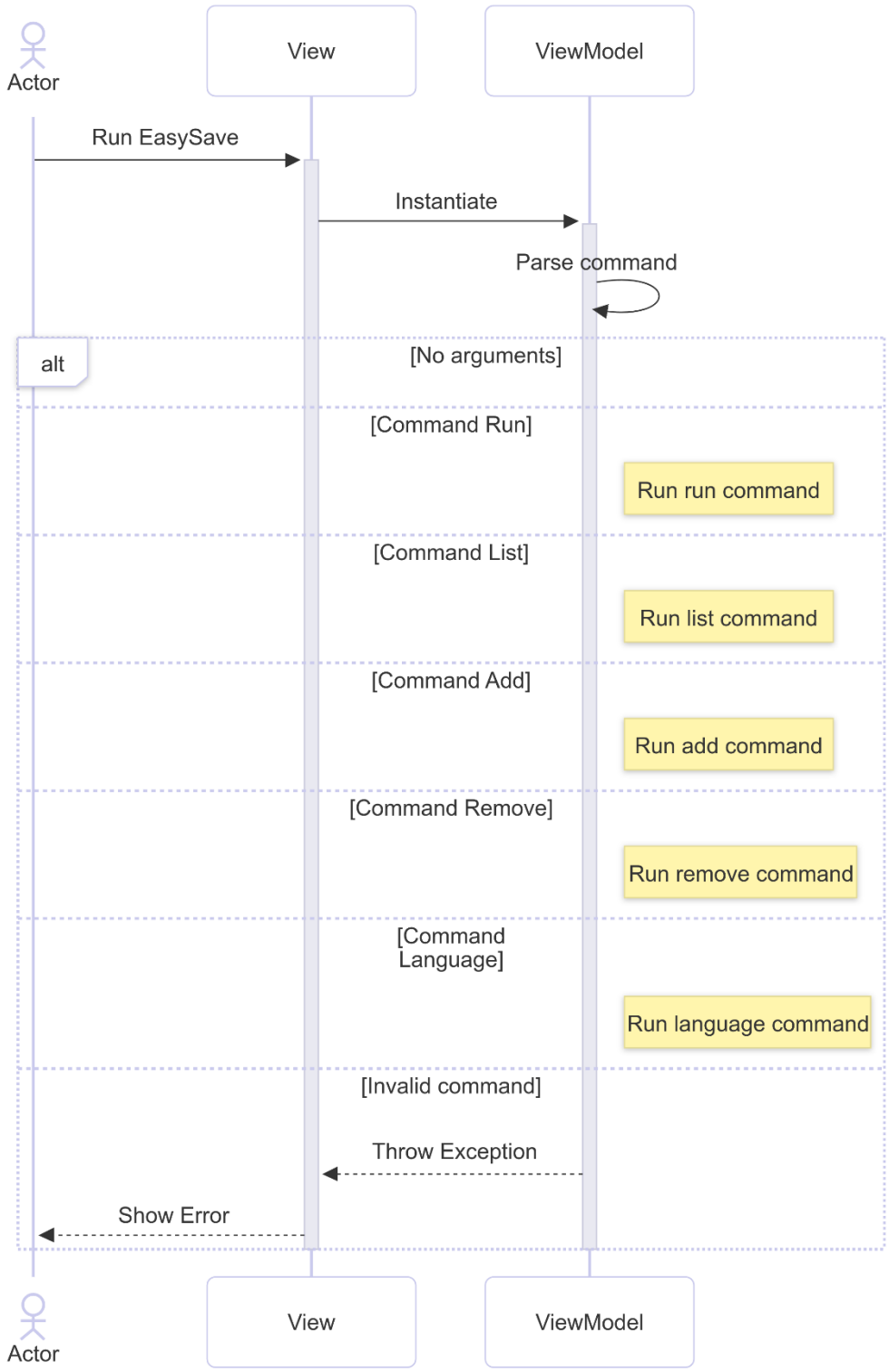

Figure 9 : Diagramme de séquence – Initialisation

Exécution des travaux de sauvegarde

Ce diagramme représente le déroulé des actions effectuées lors de l'exécution d'un ou plusieurs travaux de sauvegarde.

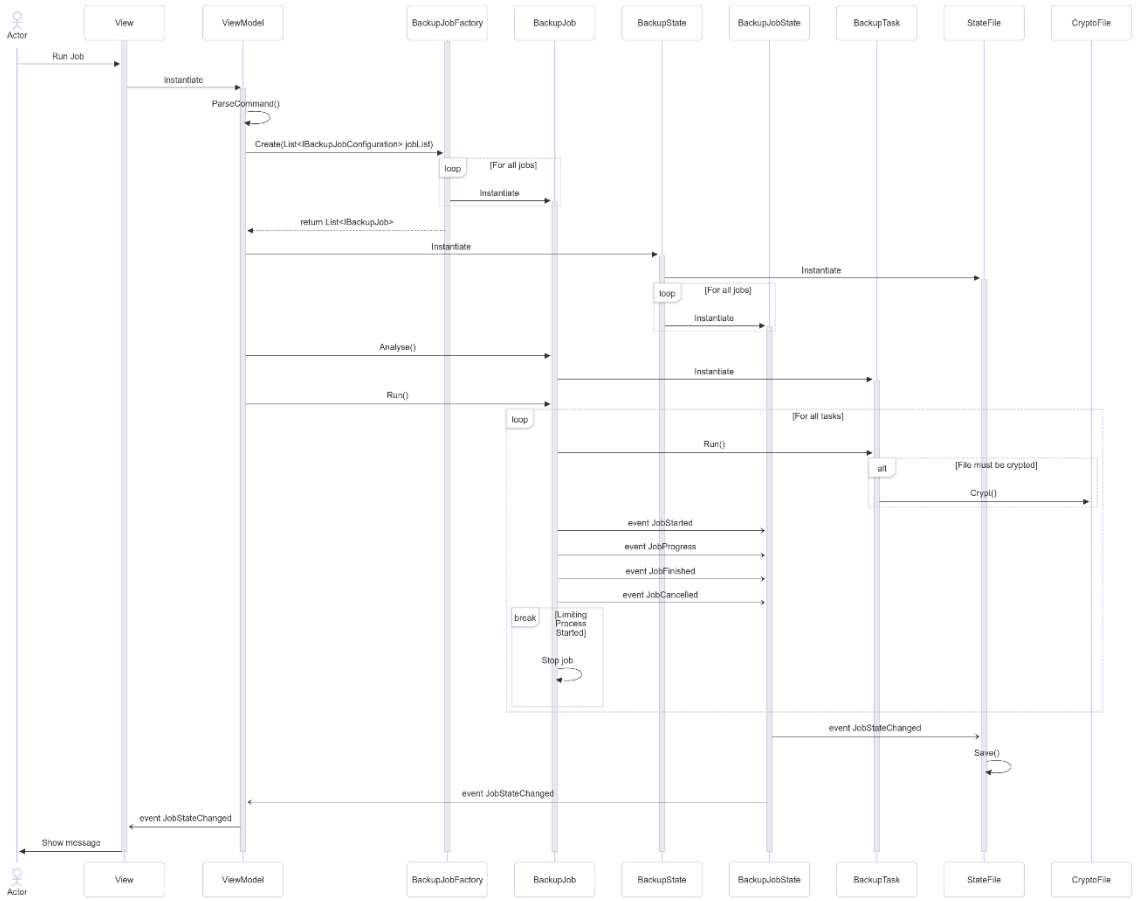

Figure 10 : Diagramme de séquence - Exécution des travaux de sauvegarde

Modification de la langue

Ce diagramme représente les étapes de modification de la langue au sein de l'application.

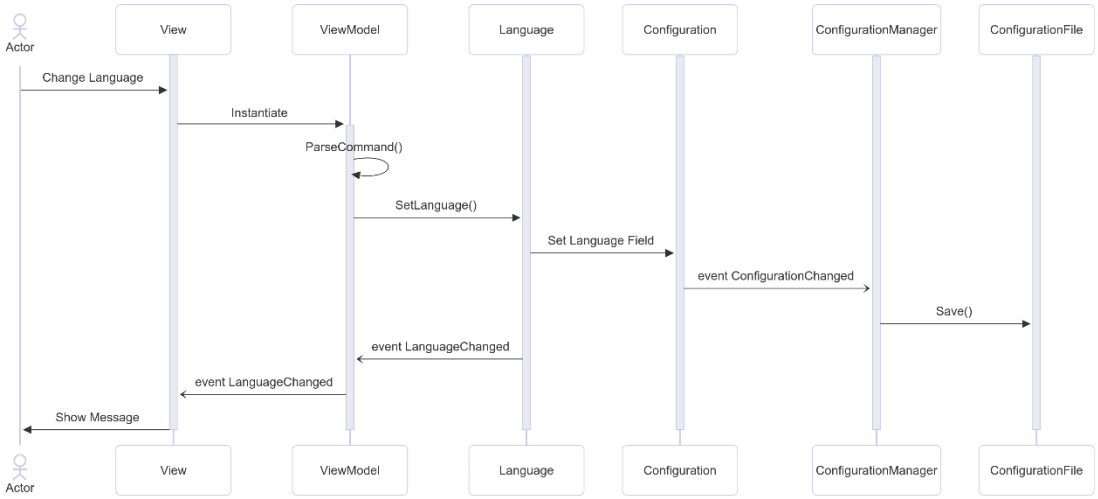

Figure 11 : Diagramme de séquence - Modification de la langue

Diagramme cas d’utilisation

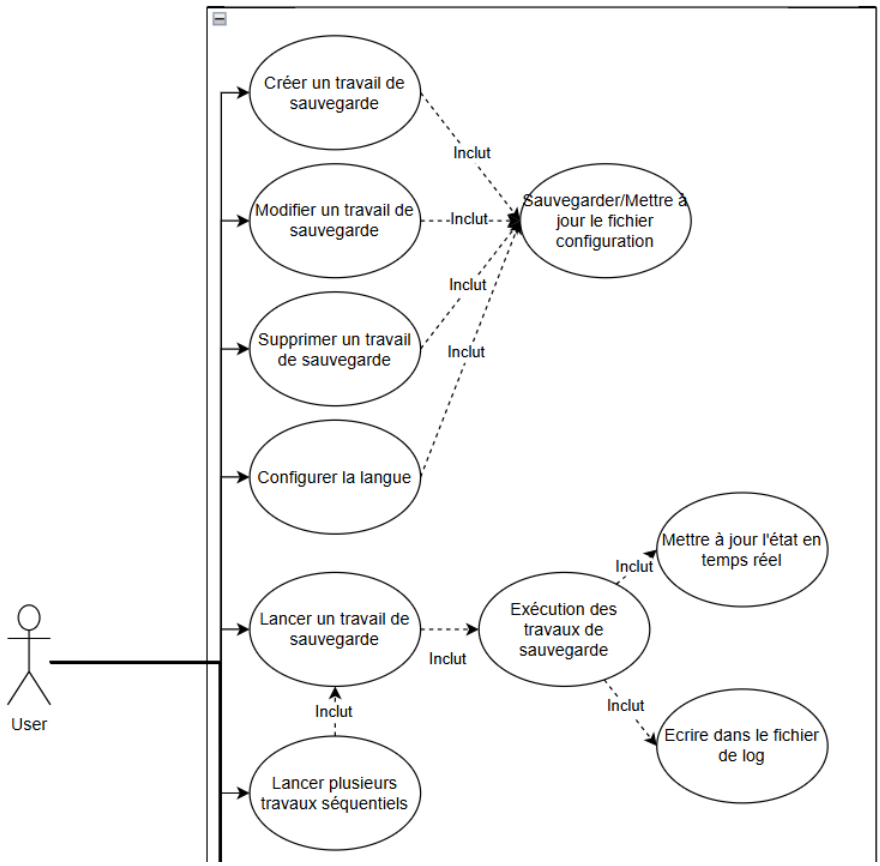

Figure 12 : Diagramme de cas d'utilisation - Partie 1

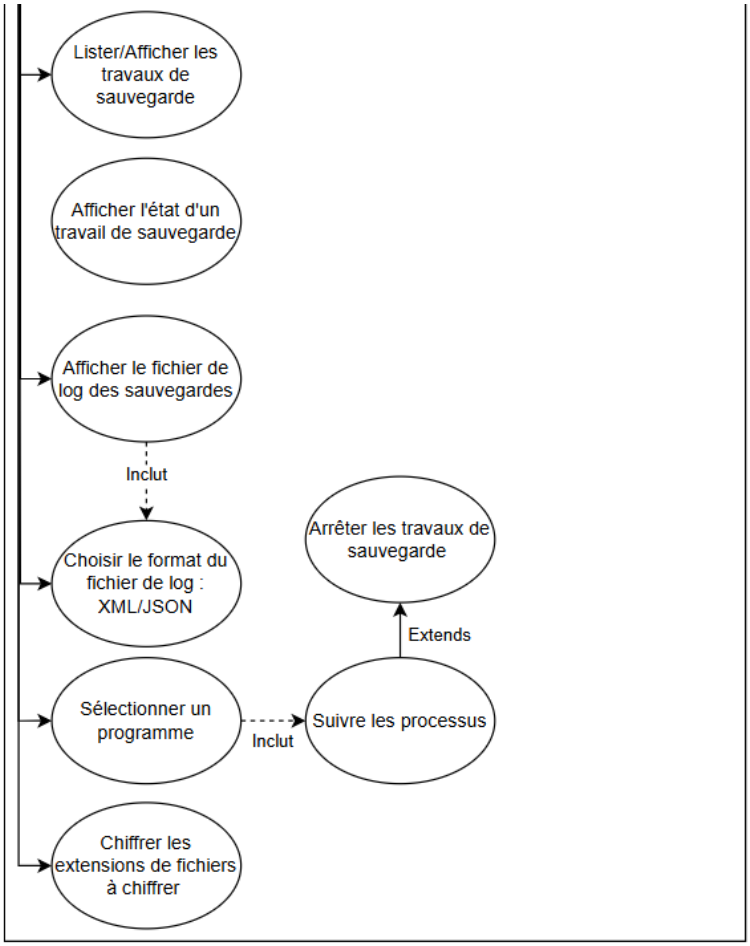

Figure 13 : Diagramme de cas d'utilisation – Partie 2

L'objectif du diagramme de cas d'utilisation est de représenter les différentes actions que le projet met à disposition des acteurs. Les cas d'utilisation accessibles à l'utilisateur sont illustrés par des bulles (ou ovales) directement reliées à lui. Certaines actions peuvent en déclencher d'autres, ce qui est représenté par des liens d'inclusion (*inclut*).

Fonctionnalités apportées avec la version 2.0 :

Les actions de sélectionner un programme ce qui peut entrainer l'arrêt des travaux nouveauté ainsi que chiffrer une extension de fichier sont des nouveautés apportées par la version 2.0.

Diagramme de composant

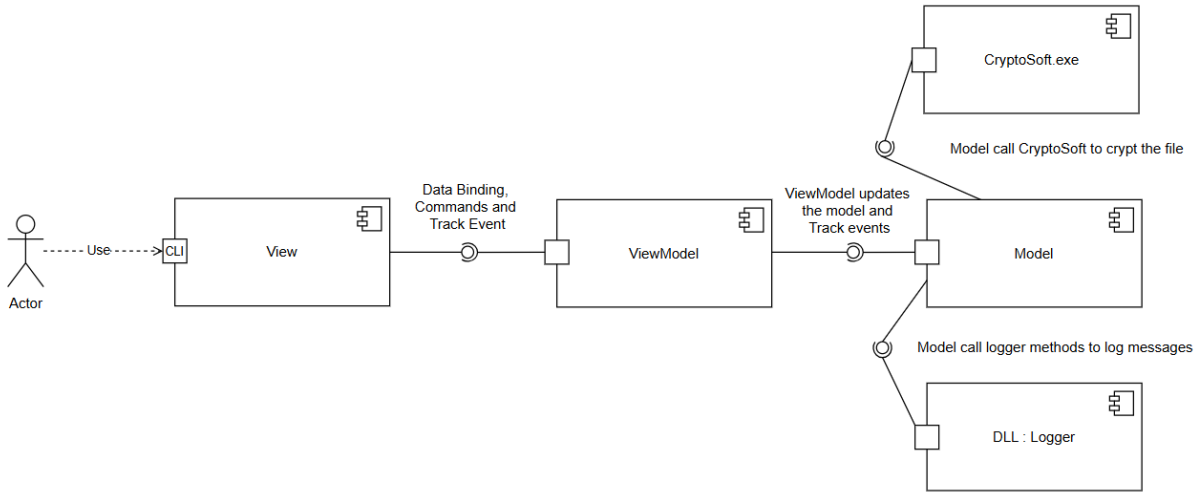

Figure 14 : Diagramme de composant

L'objectif du diagramme de composants est de représenter les relations entre les différents éléments du système. Dans notre cas, en utilisant le pattern MVVM (Model-View-ViewModel), nous avons identifié les composants *View*, *ViewModel* et *Model*, et représenté leurs interactions à l'aide de liaisons. Les points d'entrée sont également indiqués afin de préciser le sens des échanges (entrant/sortant). Ainsi, l'utilisateur saisit une commande via l'interface en ligne de commande (CLI), la *View* transmet ensuite l'information au *ViewModel*, qui agit à son tour sur le *Model*. Un dernier composant, la DLL, est également présent : il est appelé par le *Model* lorsqu'une action ou une modification doit être enregistrée dans les logs.

Fonctionnalités apportées avec la version 2.0 :

Un nouveau composant vient s'ajouter : il se nomme *CryptoSoft*. Celui-ci va chiffrer certaines extensions de fichiers qui seront définies par l'utilisateur. Il est relié à *Model*, car ce dernier va l'appeler afin de chiffrer les extensions de fichiers.