Génie Logiciel

PROJET

EasySave : Diagrammes

Remy MOREEL

Léo SCHAEFFER

Axel DELAR

Mohammed EL ORFALI

CESI
ÉCOLE D'INGÉNIEURS

Table des matières

1) Introduction

Dans le cadre du projet EasySave, notre équipe intègre l'éditeur de logiciels ProSoft sous la responsabilité du DSI. Ce projet s'inscrit dans une démarche de développement accéléré avec la livraison de trois versions successives du logiciel de sauvegarde EasySave, destiné aux entreprises.

L'objectif de ce projet est d'assurer un développement structuré et évolutif du logiciel, en respectant les contraintes techniques imposées par ProSoft. Nous devrons donc utiliser des outils comme Visual Studio 2020, GitHub ainsi qu'assurer le développement en C# avec .NET 8.0.

Ce projet met en avant des aspects clés du développement logiciel, comme le versionning, la maintenabilité et la documentation du code.

Ce premier livrable contient l'ensemble des éléments nécessaires au développement du logiciel, y compris les diagrammes UML servant de base à sa conception et à son architecture.

2) Présentation de l'équipe

Voici notre équipe en charge du projet pour le logiciel EasySave, avec Rémy MOREEL en tant que chef de projet.

Remy MOREEL

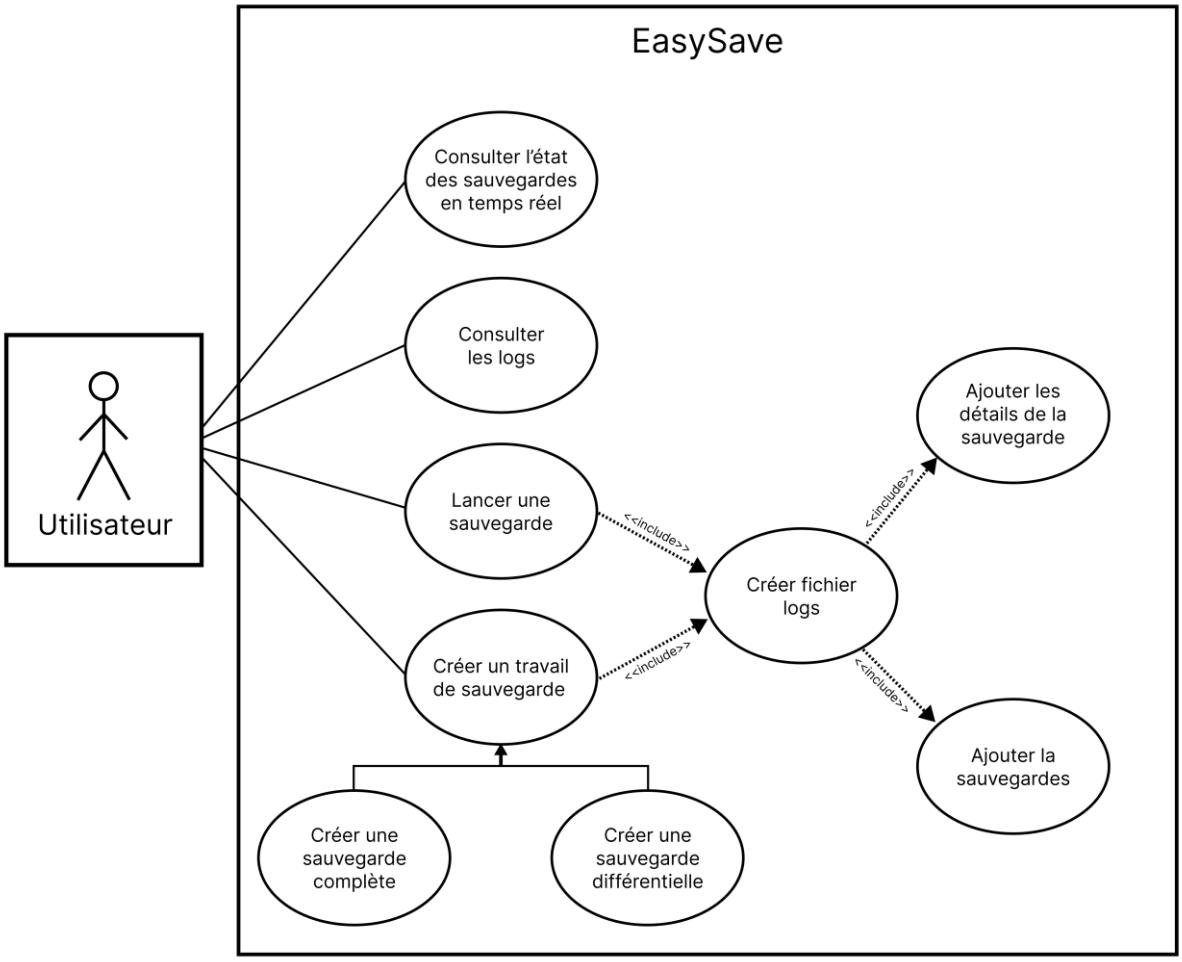
Axel DELAR

Léo SCHAEFFER

Mohammed EL ORFALI

3) Diagramme Use Case

Le diagramme de cas d'utilisation (Use Case) permet de représenter visuellement les interactions entre les utilisateurs et le système. Il met en évidence les différentes fonctionnalités du logiciel EasySave ainsi que les rôles des acteurs impliqués, facilitant ainsi la compréhension des besoins et du fonctionnement global de l'application.

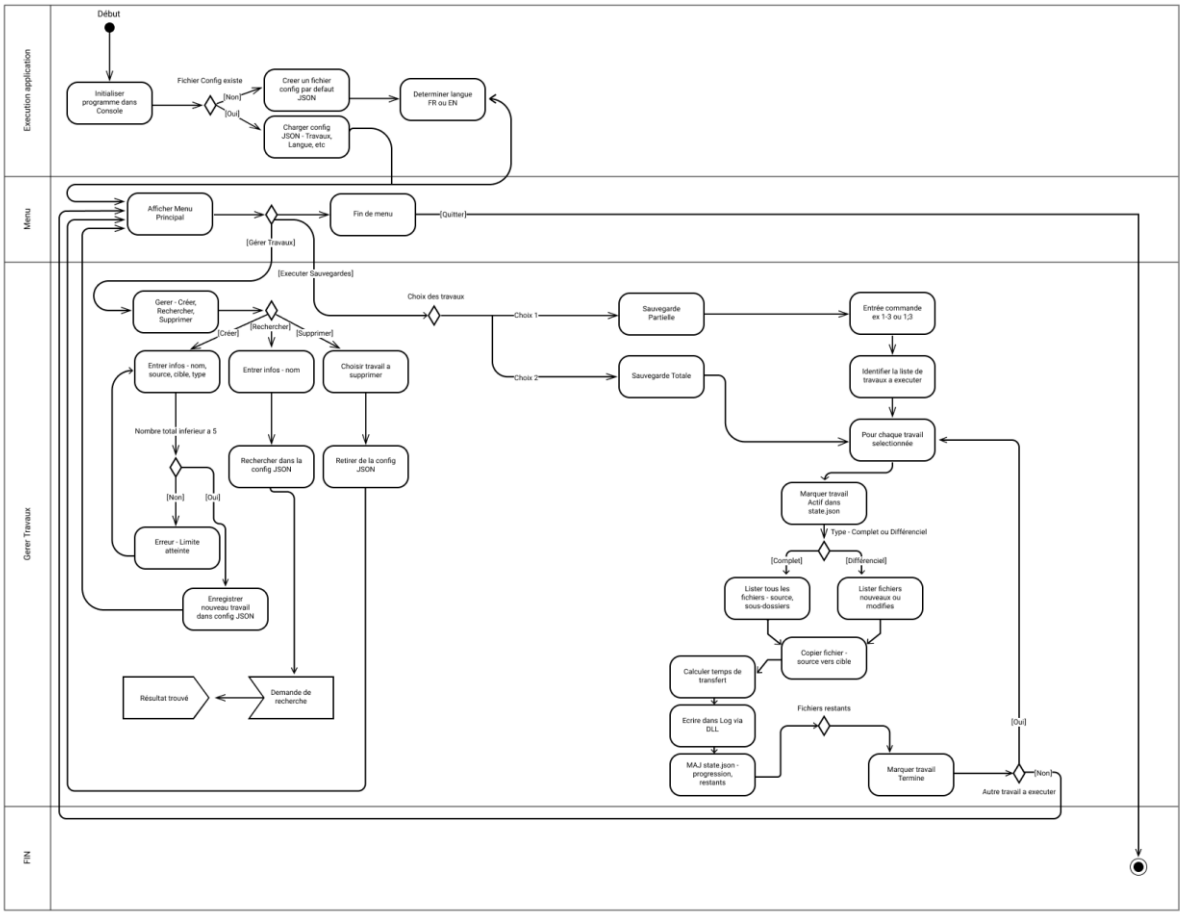


Dans notre cas, l'utilisateur peut faire 4 actions principales. Il peut consulter l'état des sauvegardes en temps réel, consulter les logs, créer un travail de sauvegarde et lancer une sauvegarde.

Pour créer un travail de sauvegarde on peut choisir entre une sauvegarde complète et une sauvegarde différentielle. On voit également que créer un fichier logs est inclus à deux autres actions, cette action est réalisée en parallèle. La fonction de créer un log se fait automatiquement quand l'utilisateur lance ou crée une sauvegarde, que ce soit le log journalier ou le log état réel.

4) Diagramme d'activité

Le diagramme d'activité nous permettra de visualiser clairement le déroulement des actions et des processus au sein du projet, ainsi que les décisions qui peuvent influencer ces actions. Il nous aidera à identifier les différentes étapes à réaliser, à comprendre les interactions entre les acteurs et à anticiper les éventuels points de blocage. En utilisant ce diagramme, nous pourrons améliorer la coordination des tâches, optimiser les flux de travail et nous assurer que chaque étape est bien définie et suivie de manière efficace.



L'exécution démarre par l'initialisation du programme dans la console, où il vérifie l'existence d'un fichier de configuration JSON. S'il n'existe pas, un nouveau fichier est créé, contenant des informations comme la langue et les paramètres du programme.

Ensuite, l'utilisateur navigue dans le menu principal, où il peut gérer les travaux en les créant, recherchant ou supprimant. Lorsqu'un travail est ajouté, l'utilisateur saisit des informations comme le nom, la source, la cible et le type. Une vérification est effectuée pour éviter de dépasser la limite de 5 travaux enregistrés. La recherche et la suppression des travaux se font via le fichier de configuration JSON.

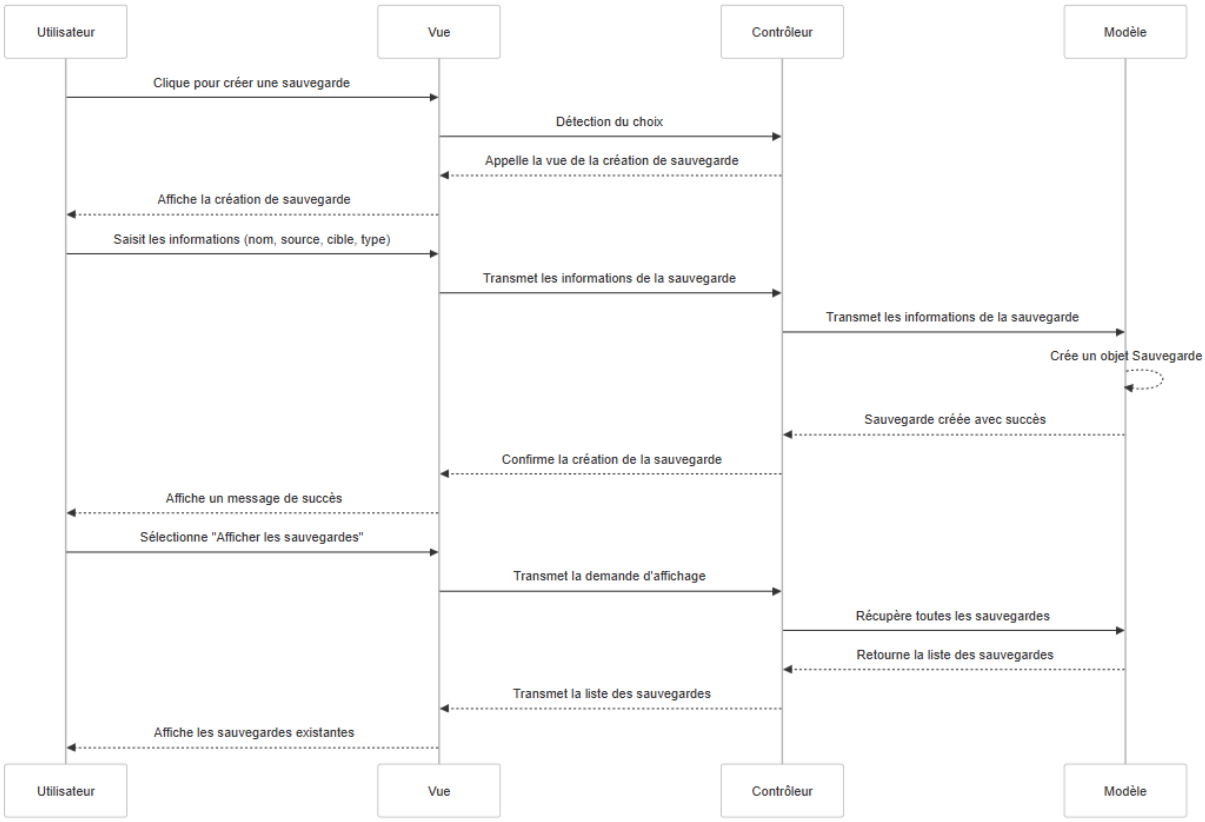
La section exécution des sauvegardes permet à l'utilisateur de choisir entre une sauvegarde totale ou partielle. Pour chaque travail sélectionné, le programme identifie les fichiers à sauvegarder selon qu'il s'agit d'une sauvegarde complète ou différentielle. Il copie les fichiers de la source vers la cible, met à jour le fichier JSON, et journalise les opérations dans un fichier log avant de marquer le travail comme terminé. Enfin, le programme vérifie s'il y a d'autres travaux à exécuter avant de se terminer.

5) Diagramme de séquence

Dans notre projet, le diagramme de séquence va nous aider à visualiser et à comprendre l'interaction entre les différents composants du système, en montrant clairement l'ordre des actions et des messages échangés. Cela nous permettra de mieux planifier et d'identifier d'éventuels problèmes de communication ou de logique.

Nous avons décidé de présenter ce qui concerne la sauvegarde, dans un premier temps la création d'un travail de sauvegarde, puis dans un second temps le lancement d'une sauvegarde.

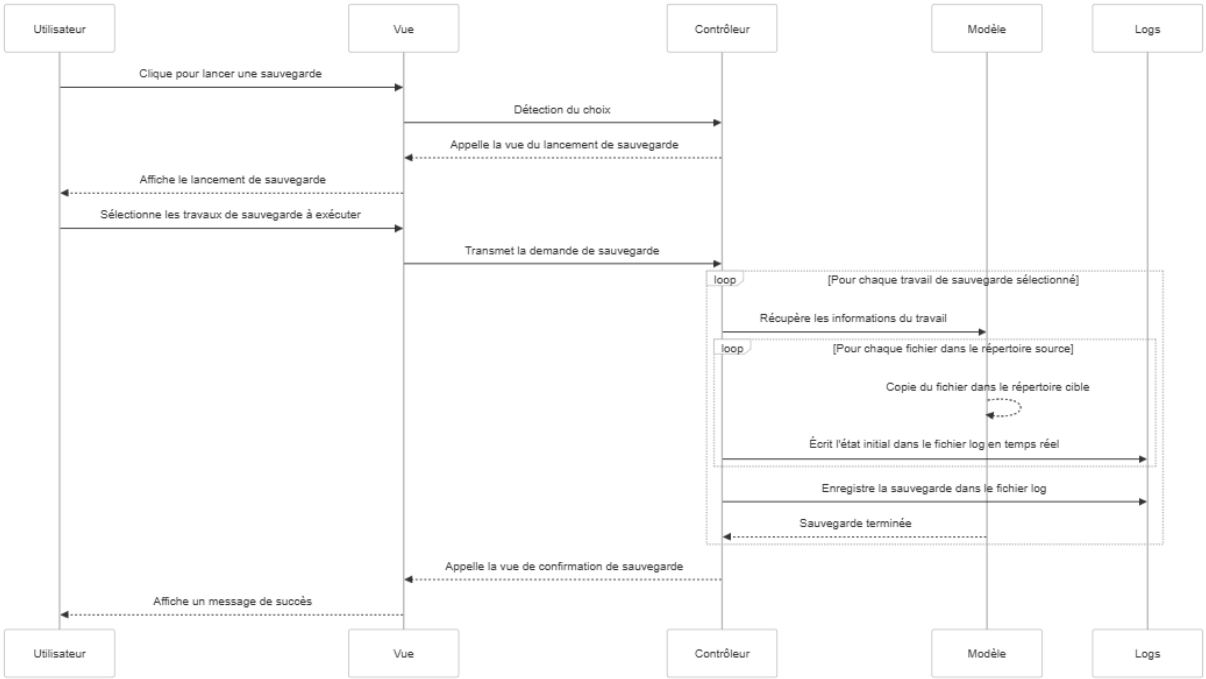
Voici donc le premier diagramme qui concerne la création d'un travail de sauvegarde :



Lorsqu'un utilisateur souhaite créer une sauvegarde, il commence par cliquer sur l'option correspondante dans l'interface. La Vue détecte cette action et transmet la demande au Contrôleur, qui appelle ensuite la vue dédiée à la création de sauvegarde. La Vue affiche alors un formulaire permettant à l'utilisateur de saisir les informations nécessaires, telles que le nom, la source, la cible et le type de sauvegarde. Une fois ces données renseignées, la Vue les transmet au Contrôleur, qui les envoie à son tour au Modèle pour créer l'objet Sauvegarde.

Le Modèle crée cet objet et informe le Contrôleur du succès de l'opération. Le Contrôleur notifie la Vue, qui affiche à l'utilisateur un message de confirmation. Si l'utilisateur choisit ensuite de visualiser ses sauvegardes existantes, il sélectionne l'option correspondante dans l'interface. La Vue transmet cette demande au Contrôleur, qui sollicite le Modèle pour récupérer la liste des sauvegardes. Une fois la liste renvoyée par le Modèle, le Contrôleur la transmet à la Vue, qui l'affiche à l'utilisateur, lui permettant ainsi de consulter ses sauvegardes.

Voici maintenant le diagramme qui concerne le lancement d'une sauvegarde :



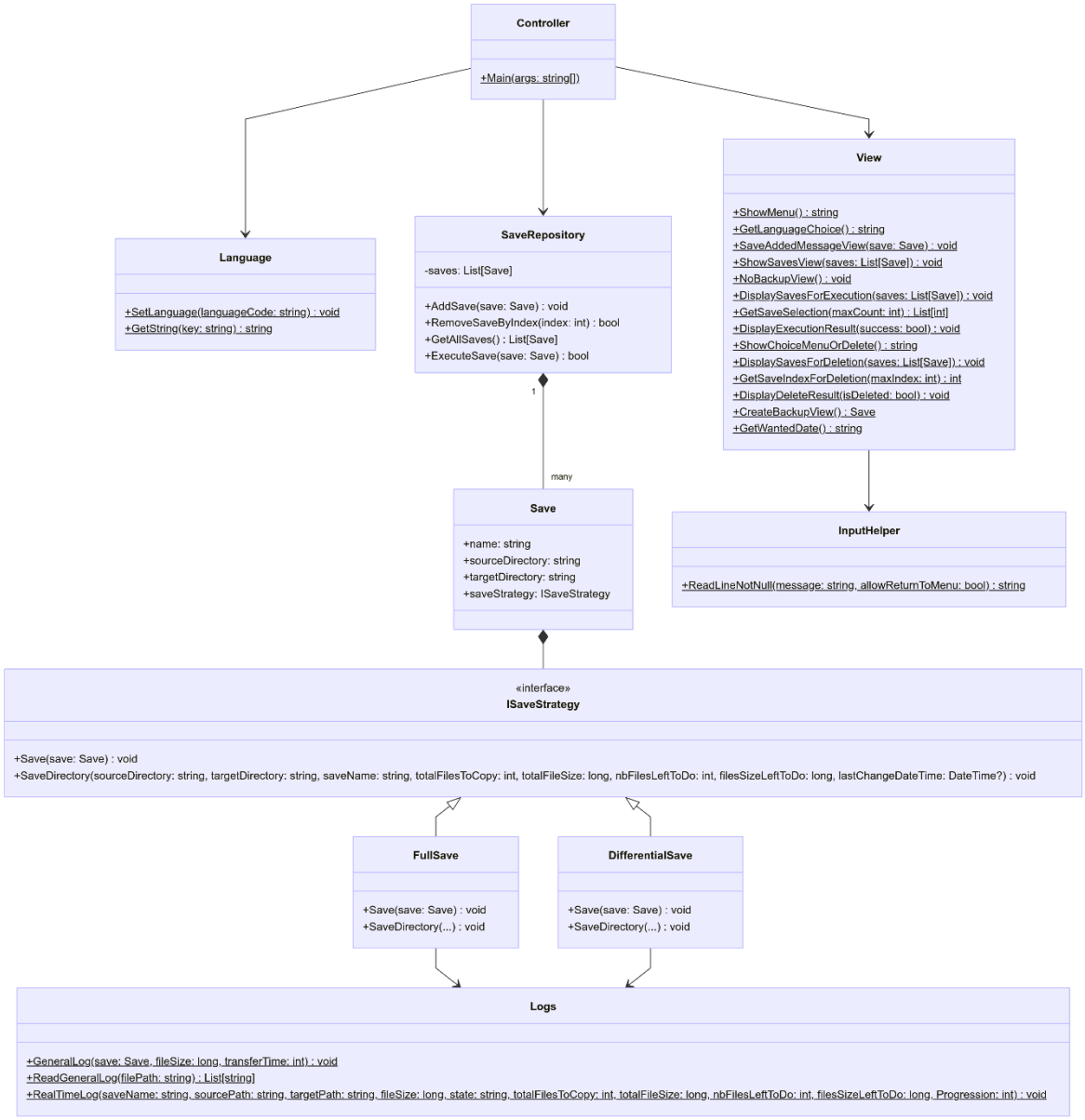
Lorsqu'un utilisateur souhaite lancer une sauvegarde, il commence par cliquer sur l'option appropriée dans l'interface. La Vue détecte l'action et transmet la demande au Contrôleur, qui appelle la vue de lancement de la sauvegarde et l'affiche à l'utilisateur.

Ensuite, l'utilisateur sélectionne les travaux de sauvegarde à exécuter, et ces informations sont envoyées du Vue au Contrôleur. Le Contrôleur, pour chaque travail de sauvegarde sélectionné, récupère les informations nécessaires du Modèle. Pour chaque fichier situé dans le répertoire source, le Modèle effectue une copie du fichier vers le répertoire cible, et l'état de cette opération est écrit en temps réel dans le fichier log grâce à l'interaction avec le service Logs.

Après avoir traité tous les fichiers d'un travail de sauvegarde, le Contrôleur enregistre l'opération complète dans le fichier log et informe le Contrôleur que la sauvegarde est terminée. Une fois toutes les sauvegardes effectuées, le Contrôleur appelle la vue de confirmation de la sauvegarde, et la Vue affiche à l'utilisateur un message de succès, indiquant que la sauvegarde a été réalisée avec succès.

DT1 Deployment - Activity Diagram.pdf

6) Diagramme de classes



Le diagramme de classes permet pendant la phase de modélisation, de conception, puis de réalisation d'avoir une idée claire concernant l'architecture de l'application. Celui-ci peut évoluer au fur et à mesure des différentes versions de l'application, tout en permettant à l'utilisateur technique de mieux comprendre son environnement de développement.

Dans notre application, nous avons pris la décision que les classes qui n'auront jamais plus d'une seule instance auraient des méthodes static, nous permettant ainsi de n'appeler que le nom de la classe avec sa méthode, plutôt que d'en créer un objet (exemple, InputHelper, View, Logs, Language).

Ensuite sur le diagramme ci-dessus, on peut bien voir l'architecture MVC (modèle View-Controller), avec la classe Controller qui fait le lien entre les classes du modèle (SaveRepository, composée de 0 à 5 Save) et la vue.

Pour le choix des types de sauvegardes, nous avons décidé d'implémenter une stratégie via une interface, et d'assigner un objet stratégie (FullSave ou DifferentialSave) en tant qu'attribut d'une sauvegarde (Save). Cela permet de limiter la création des objets FullSave/DifferentialSave et de favoriser la réutilisation du code. Un des points d'amélioration possible serait d'implémenter une factory, permettant la création des stratégies de sauvegardes.

Un autre point d'amélioration serait d'implémenter un observateur, afin de ne déclencher l'écriture dans les logs que via des évènements générés par l'observateur, lorsqu'un fichier est sauvegardé.

Tous ces changements seront étudiés dans les futures versions de l'application, afin d'améliorer l'efficacité et la réutilisabilité du code de l'application, et rendre le projet le plus modulable possible.

7) Conclusion

Grâce à l'ensemble des diagrammes UML réalisés, nous disposons désormais d'une vision claire et structurée du fonctionnement et de l'architecture du logiciel EasySave 1.0. Ces représentations nous permettent de bien comprendre les interactions entre les différents composants, les cas d'utilisation, ainsi que la gestion des sauvegardes et des logs.

En conclusion, la construction de ces différents diagrammes a permis de réfléchir et d'avoir une vision claire sur l'ensemble du projet pour correctement coder EasySave.