

PROJET PROGRAMMATION SYSTEME

LIVRABLE 0

FODIL Nel | MARCELLI Enzo | GOUADFEL Rayan | ARRIGHI Fabien

28 janvier 2024

Table des matières

I.	Intro	4
II.	Contexte	4
III.	Organisation du projet	5
1.	Méthodologie PDCA	5
2.	Rôles d'Équipe	6
3.	Planification	6
4.	Diagramme de GANTT	7
IV.	Technologies utilisées	8
1.	.NET	8
2.	C#	9
3.	Environnement de développement	9
4.	Git et Versioning	10
5.	Tests / Déploiement	10
5.1.	Utilisation de Docker pour les Tests	11
5.2.	Mise en Place du Serveur Jenkins pour les Tests et le Déploiement	11
5.3.	Configuration des Pipelines CI/CD dans Jenkins	11
5.4.	Configuration finale	12
1.	Diagramme UML	13
2.	ReadMe	18
3.	Code et Architecture Logicielle	19
V.	Pipeline CI/CD	22
1.	Intégration Continue	22
2.	Déploiement Continue	23
3.	Github Actions	23
VI.	Conclusion	25
VII.	Webographie	26

Table des figures

Figure 1 : PDCA	5
Figure 2 : GANTT.....	7
Figure 3 : Logo .NET	8
Figure 4 : Logo C#	9
Figure 5 : Exemple diagramme UML	13
Figure 6 : Diagramme UML d'utilisation.....	14
Figure 7: Diagramme UML d'activité.....	15
Figure 8 : Diagramme UML de séquence	16
Figure 9 : Diagramme UML de classe	17
Figure 10 : Sommaire ReadMe	18
Figure 11 : Commentaires Code	21
Figure 12 : Logo Jenkins.....	23
Figure 13 : Extrait fichier YAML	25

I. Intro

Ce rapport a pour objectif de faire office de « Fil Rouge » durant le projet. Il sert à maintenir la cohérence et l'alignement tout au long du projet. En cas de changement ou d'ajustement nécessaire pendant le projet, ce rapport servira de référence pour évaluer l'impact de ces changements sur les objectifs globaux.

II. Contexte

ProSoft, éditeurs de logiciels, a intégré notre équipe pour gérer le projet « EasySave » sous la supervision du DSI. Ce projet vise à développer un logiciel de sauvegarde intégré à la politique tarifaire de ProSoft, avec un prix unitaire de 200€HT. Le contrat de maintenance annuel inclut mises à jour et est fixé à 12% du prix d'achat, avec une reconduction automatique basées sur l'indice SYNTEC.

Les responsabilités de notre équipe incluent le développement, la gestion des versions et la documentation. Cette documentation comprend un manuel d'utilisation pour les utilisateurs et les informations nécessaires pour le support client. Cela permettra de faciliter la compréhension du logiciel pour le client, et de faciliter le travail du support client.

Afin de garantir une reprise de notre travail par d'autres équipes, la direction de l'entreprise nous impose de travailler en suivant certaines contraintes. Notamment le développement du logiciel en C#, l'utilisation de Git mais encore une certaine lisibilité et maintenabilité du code. L'objectif est de diriger ce projet de manière à réduire les coûts de développement des futures versions. Et surtout d'être capable de proposer rapidement des mises à jour en cas d'éventuels dysfonctionnement au sein du logiciel.

III. Organisation du projet

1. Méthodologie PDCA

La gestion du projet EasySave intègre l'approche itérative PDCA, un modèle de gestion de la qualité pour un contrôle et une amélioration continue des processus et des produits. Cette méthode, également connue sous le nom de "roue de Deming", est structurée en quatre phases :

Plan (Planifier) : Identification des objectifs, analyse des processus actuels et planification des changements nécessaires. Dans cette phase, nous définirons les exigences du logiciel EasySave, établirons les spécifications et concevrons les solutions pour répondre aux besoins identifiés.

Do (Faire) : Mise en œuvre des plans, souvent à petite échelle d'abord, pour tester leur viabilité. L'équipe développera une version bêta du logiciel, en appliquant les plans définis et en intégrant les fonctionnalités essentielles.

Check (Vérifier) : Suivi et évaluation des résultats par rapport aux attentes, analyse des données pour vérifier l'efficacité des changements. Nous évaluerons le logiciel en termes de fonctionnalité, de performance et de retour utilisateur.

Act (Agir) : Action basée sur ce qui a été appris dans la phase de vérification. Si les changements n'ont pas fonctionné, ils seront revus et affinés. Si les résultats sont satisfaisants, les changements seront implémentés à plus grande échelle et le logiciel sera finalisé pour la production.

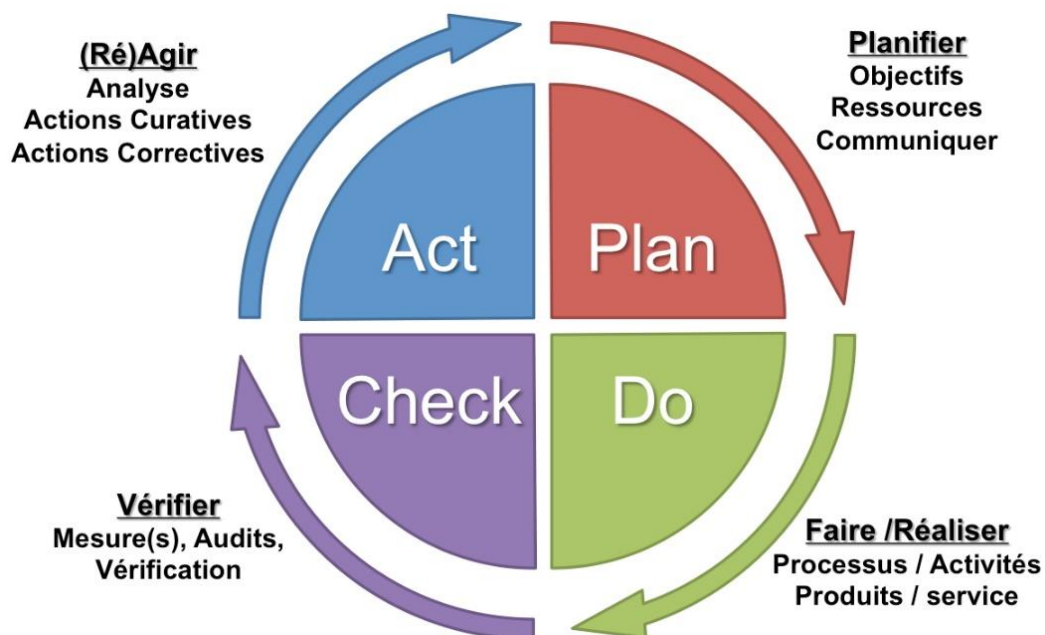


Figure 1 : PDCA

2. Rôles d'Équipe

Les rôles au sein de notre équipe ont été définis pour favoriser une collaboration optimale et efficace :

- **Animateur (Fabien)** : Assumera le rôle de facilitateur, s'assurant que toutes les réunions et sessions de travail collaboratif se déroulent sans heurts et que chaque membre de l'équipe est engagé et contribue pleinement.
- **Gestionnaire de Temps (Nel)** : Prendra en charge le suivi du planning et veillera à ce que le projet reste dans les temps. Il sera responsable de rappeler les échéances et de s'assurer que les objectifs sont atteints en temps voulu.
- **Secrétaire (Enzo)** : S'occupera de la documentation du projet, y compris la mise à jour et l'archivage des comptes-rendus, des décisions prises et des modifications du cahier des charges.
- **Scrib (Rayan)** : Aura pour mission de documenter les discussions clé, rédiger les synthèses et s'assurer que les points importants sont communiqués à toute l'équipe.

3. Planification

Le projet EasySave a été planifié en trois phases principales, avec une attention particulière portée à la gestion efficace du temps et des ressources. La première phase, Initiation et Conception, se déroule du 29 janvier au 8 février 2024. Pendant cette période, l'équipe se concentrera sur la compréhension approfondie du cahier des charges et la mise en place de l'environnement de développement, suivies par la planification initiale et le début de la programmation de la version 1.0 du logiciel, incluant la création des premiers diagrammes UML.

La seconde phase, Développement et Amélioration, du 9 au 16 février 2024, permettra l'élaboration des versions 1.1 et 2.0. Cette période sera consacrée à l'amélioration des fonctionnalités existantes et au développement de nouvelles fonctionnalités, tout en continuant à documenter le processus à travers des diagrammes UML.

La troisième et dernière phase, Finalisation et Préparation de la Soutenance, prévue du 19 février au 1er mars 2024, se concentrera sur le peaufinage du livrable final, la version 3.0, et la préparation de la présentation du projet. Cette étape cruciale comprend la finalisation du développement, la mise à jour des diagrammes UML pour le livrable 3 et les préparatifs pour la soutenance du projet.

Tout au long du projet, une approche agile sera adoptée pour garantir flexibilité et réactivité face aux changements. Cela signifie que le projet peut évoluer en fonction des retours, des révisions du cahier des charges et des demandes des parties prenantes. L'objectif est d'assurer que le logiciel EasySave non seulement répond aux besoins initiaux mais est également adaptable aux exigences futures.

4. Diagramme de GANTT

EasySave

2F Roaming

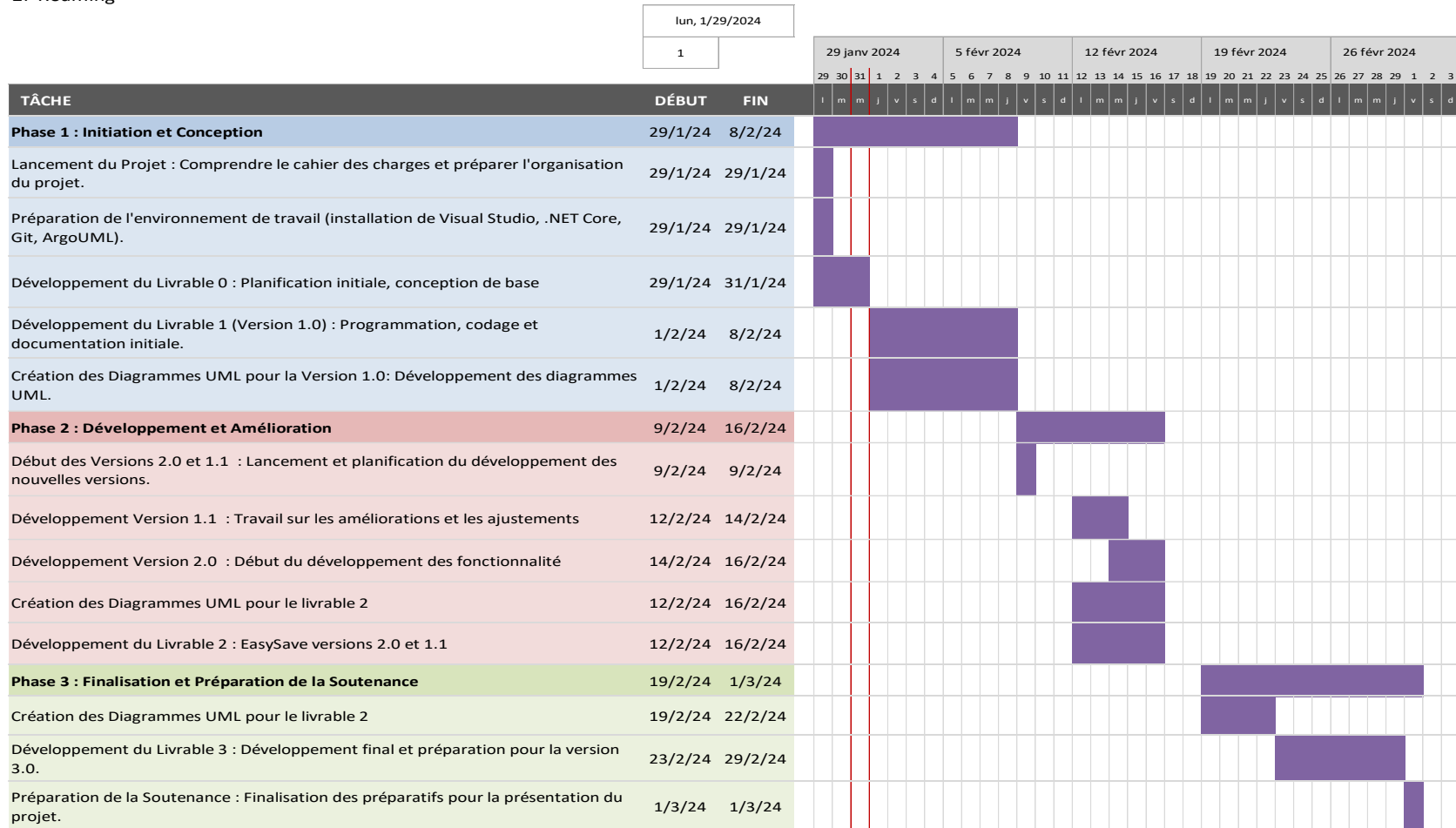


Figure 2 : GANTT

IV. Technologies utilisées

1. .NET

Pour réaliser le projet EasySave qui nous a été confié nous allons utiliser le Framework .NET sous sa dernière version .NET 8.0

Initialement .NET est sorti sous le nom de .NET Framework, ayant pour objectif d'être une solution de développement sous Windows exclusivement. Par la suite a été créé .NET Core avec le but d'offrir une meilleure interopérabilité, conçu pour être multiplateforme, il peut être utilisé sur Windows, Linux et MacOS. .NET Core a été distribué avec un ensemble de CLR compatible sur ces différents systèmes d'exploitation. De plus, Microsoft a Open-Sourcé .NET Core pour que chacun puisse y contribuer.

Finalement arrive la création de .NET STANDART, avec le but d'être un sous-ensemble du Framework comme un standard accessible aussi bien en .NET Core qu'en .NET Framework.

Il y avait alors trois Frameworks, et cela commençait à causer certaines confusions au sein de la communauté. Pour pallier ce problème Microsoft a tenté une unification pour apporter plus de cohérence. Ils ont donc créé l'unique .NET avec pour première version .NET 5.0, qui est le socle de toutes les versions de .NET à venir.

Avec .NET on retrouve de nombreuses bibliothèques et de Frameworks qui existe en .NET, par exemple WinForms qui est un ensemble de bibliothèque graphique pour le développement d'UI pour les applications Windows. Il existe le Framework ML.NET pour le machine learning, les Frameworks ASP.NET et Blazor pour le Web.



Figure 3 : Logo .NET

.NET prend en charge plusieurs langages de programmation tels que C#, F# et Visual Basic, permettant aux développeurs de choisir le langage qui convient le mieux à leurs compétences et à leurs besoins.

Pour fonctionner, .NET s'appuie sur son composant central, le CLR (Common Language Runtime). Le CLR est une machine virtuelle qui gère l'exécution des programmes .NET. Il offre des fonctionnalités telles que la gestion de la mémoire, la compilation à la volée, la gestion des exceptions, etc. Le CLR assure également la sécurité en exécutant le code dans un environnement isolé, appelé bac à sable.

Pour assurer la compatibilité il existe le CIL (Common Intermediate Language), c'est un langage intermédiaire utilisé par le CLR. Lorsqu'on compile un programme .NET, il se retrouve

traduit en CIL au lieu du langage machine spécifique à une plateforme. Cela permet au code d'être portable et de s'exécuter sur n'importe quelle plateforme compatible avec le CLR.

- 1- Plus concrètement le code source est écrit dans un langage tel que le C#
- 2- Le code source est compilé en code intermédiaire CIL à l'aide du compilateur de langage (compilateur C#)
- 3- Lors de l'exécution, le CLR charge le code intermédiaire CIL et le compile en code machine spécifique à la plateforme.

2. C#

Comme nous l'avons vu .NET propose de nombreux langages, pour ce projet nous allons utiliser C#. Ce dernier est un langage de programmation orienté objet créé en 2000, il a été conçu pour développer sur la plateforme Microsoft .NET.

C# est un langage dérivé du C++ et avec une certaine ressemblance à Java. Il propose des caractéristiques comme un typage sûr, de l'encapsulation, de l'héritage et du polymorphisme. Il comporte également un ramasse-miettes et un système de gestion d'exceptions.

C'est donc avec C# que nous allons développer notre application Windows.



Figure 4 : Logo C#

3. Environnement de développement

Pour ce projet, toute l'équipe travaillera sur Visual Studio 2022. Visual Studio est un environnement de développement intégré (IDE) qui regroupe plusieurs outils permettant de créer, éditer, déboguer et gérer le code plus facilement. Dans notre cas, nous allons devoir par la suite réaliser une interface graphique, Visual Studio possède également les fonctionnalités nécessaires pour réaliser ce genre de projet.

Nous n'utiliserons aucune extension pour ce projet mais un paramétrage commun sera fait pour le formatage du code.

4. Git et Versioning

Afin d'assurer le suivi de chaque version de notre code et de faciliter la collaboration entre les développeurs, nous utiliserons Git et GitHub. Le premier outil est un logiciel de gestion de versions qui permettant d'avoir une vue complète de la base de code ainsi que de son historique sur tous les postes de travail des développeurs. GitHub quant à lui, est un service web permettant de sauvegarder un dépôt de manière distante, favorisant ainsi la collaboration au sein de l'équipe.

Nous utiliserons également GitHub afin d'appliquer des règles de sécurités à notre branche principale. En effet, il ne sera pas possible de pousser du code directement sur la branche principale. Au lieu de cela, les développeurs devront d'abord effectuer un commit sur une autre branche, puis soumettre une pull request qui devra être validée par une ou plusieurs personnes.

Notre dépôt sera constitué de deux branches principales :

- ``main`` qui contiendra le code de production stable.
- ``dev`` où les fonctionnalités sont fusionnées au fur et à mesure de leur achèvement.

D'autres branches seront créées au fur et à mesure de l'avancement du projet pour travailler sur le développement des fonctionnalités ou la correction de bug.

Une convention de nommage pour les commits sera également mise en place avec la rédaction du message de commit en anglais et avec l'utilisation de mots-clés ou autres afin d'identifier facilement les modifications apportées par les commits.

Bien que la grande majorité des manipulations puissent être faites directement via ligne de commande avec Git, l'utilisation d'une interface graphique se révélera être utile dans certains cas tels que la gestion de conflit. Pour cela, nous utiliserons l'application GitHub Desktop.

5. Tests / Déploiement

L'utilisation de Jenkins et Docker ensemble nous permettra de créer un processus de développement, de test et de déploiement robuste, automatisé et efficace, aligné avec les meilleures pratiques de DevOps. Cette configuration contribue non seulement à l'efficacité opérationnelle, mais améliore également la fiabilité et la qualité du logiciel que nous déployons pour EasySave.

5.1. Utilisation de Docker pour les Tests

Dans notre projet EasySave, Docker jouera un rôle essentiel pour créer des environnements de test isolés et reproductibles. En encapsulant notre application et son environnement dans des conteneurs Docker, nous assurons que nos tests sont effectués dans des conditions constantes, réduisant ainsi les variables et les problèmes de "ça marche sur ma machine".

Nous utiliserons des images Docker pour définir les configurations nécessaires des systèmes d'exploitation et des piles logicielles. Ceci garantit que chaque membre de l'équipe, indépendamment de son environnement de développement local, peut exécuter des tests fiables et cohérents. De plus, nous envisageons d'utiliser les extensions disponibles pour les frameworks de test, comme NUnit ou xUnit pour .NET, qui peuvent être intégrées dans nos conteneurs Docker pour automatiser les tests unitaires et d'intégration.

5.2. Mise en Place du Serveur Jenkins pour les Tests et le Déploiement

Jenkins, un serveur d'intégration continue (CI) et de déploiement continu (CD), sera au cœur de notre pipeline d'automatisation. Nous configurerons Jenkins pour qu'il surveille notre dépôt Git et déclenche des builds automatiques à chaque nouvelle soumission de code (commit).

Chaque build sera accompagné d'une série de tests automatisés qui valideront les modifications. Si les tests sont concluants, le build pourra être marqué comme prêt pour le déploiement. En cas d'échec, l'équipe sera alertée pour corriger les problèmes avant de procéder.

5.3. Configuration des Pipelines CI/CD dans Jenkins

Nos pipelines CI/CD dans Jenkins seront configurés pour effectuer les actions suivantes :

- **Extraction du Code** : À chaque commit poussé sur la branche de développement ou à chaque pull request, Jenkins extraira le code du dépôt Git pour débiter le processus de CI/CD.
- **Build et Rapports de Tests** : Après les tests, Jenkins compilera le code et générera des rapports de tests qui seront mis à disposition de l'équipe.
- **Déploiement Automatisé** : Une fois validé, le code sera automatiquement déployé dans l'environnement de staging pour des tests plus approfondis ou directement en production selon la stratégie de déploiement définie.

5.4. Configuration finale

Toutes les parties précédemment abordées ont été traitées en tout début de projet afin de proposer nos premières potentielles solutions. Nous les avons laissés dans ce livrable afin de bien laisser en évidence nos différentes propositions le long de ce projet, étant l'essence même de ce livrable fil rouge. Elles n'ont donc pour certaines pas été réalisées ou bien plus d'actualité, cette partie a donc pour objectif de décrire notre configuration actuelle en fin de projet.

Dans un premier temps, nous avons travaillé sur Docker et Jenkins dans le cadre de la version 1.0 et 1.1. Avec Docker, nous avons mis en place des images sur le Docker Hub, directement téléchargeable pour chaque version afin de pouvoir lancer l'application dans un container.

Docker nous a également servis à mettre en place notre serveur Jenkins, dans lequel on retrouve 2 configurations, une pour la version 1.0 et la deuxième pour la version 1.1.

Finalement, durant notre projet et après avoir réalisé la version 1.1, nous avons abandonnée Docker et Jenkins pour laisser place à GitHub Actions. Nous expliquerons en détails nos arguments justifiant ce changement dans la partie « Pipeline CI/CD ».

Avec GitHub Actions, nous avons pu mettre en place une pipeline complète, réalisant le build, le lancement des tests unitaires (ils sont aux nombres de 17 dans la solution « EasySaveGUI_UnitTests »), le publish ainsi que le release sur Git. Cela pour la version 2.0 ainsi que la 3.0, avec pour cette dernière un deuxième publish et release pour la console déportée. Tous les détails concernant les différentes étapes à suivre ainsi que les commandes sont expliqués dans le ReadMe, qu'on abordera par la suite dans ce livrable.

1. Diagramme UML

Les Diagrammes UML (Unified Modeling Language) sont un élément clé dans la planification et la documentation de notre projet EasySave. Pour la création de ces diagrammes, notre choix initial s'est porté sur Draw.io. Cet outil offre les fonctionnalités nécessaires pour créer des diagrammes UML complets.

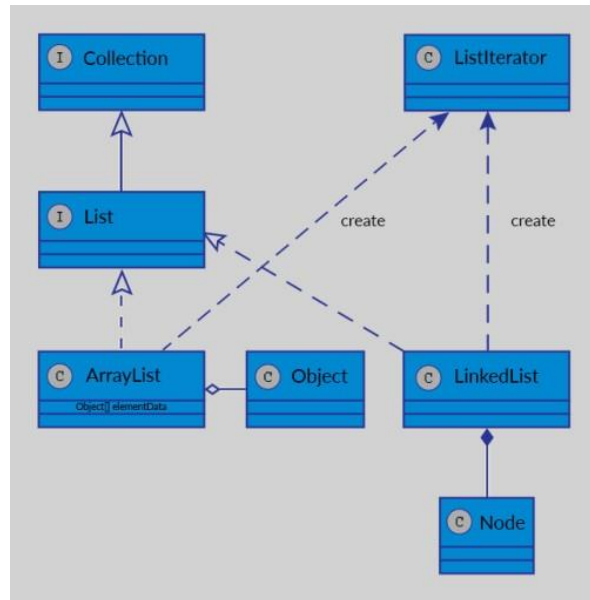


Figure 5 : Exemple diagramme UML

Ces diagrammes nous permettent de visualiser, de spécifier, de construire et de documenter les parties du système de logiciels en développement. Dans le cadre de notre projet, les UML joueront plusieurs rôles cruciaux :

- Conception de l'Architecture
- Communication entre Équipe
- Analyse des Besoins
- Documentation du Projet

Dans le cadre de ce projet, afin de garantir qualité, cohérence et compréhension de notre application, nous avons choisis quatre sortes de diagrammes UML.

Tout d'abord le diagramme d'utilisation permettant de décrire les interactions entre les utilisateurs et notre système. En identifiant les différents cas d'utilisation et en décrivant leurs relations, nous pourrions garantir que notre application répond aux besoins réels du client tout en optimisant l'expérience.

Ci-dessous en exemple, le diagramme d'utilisation de la version 2.0 :

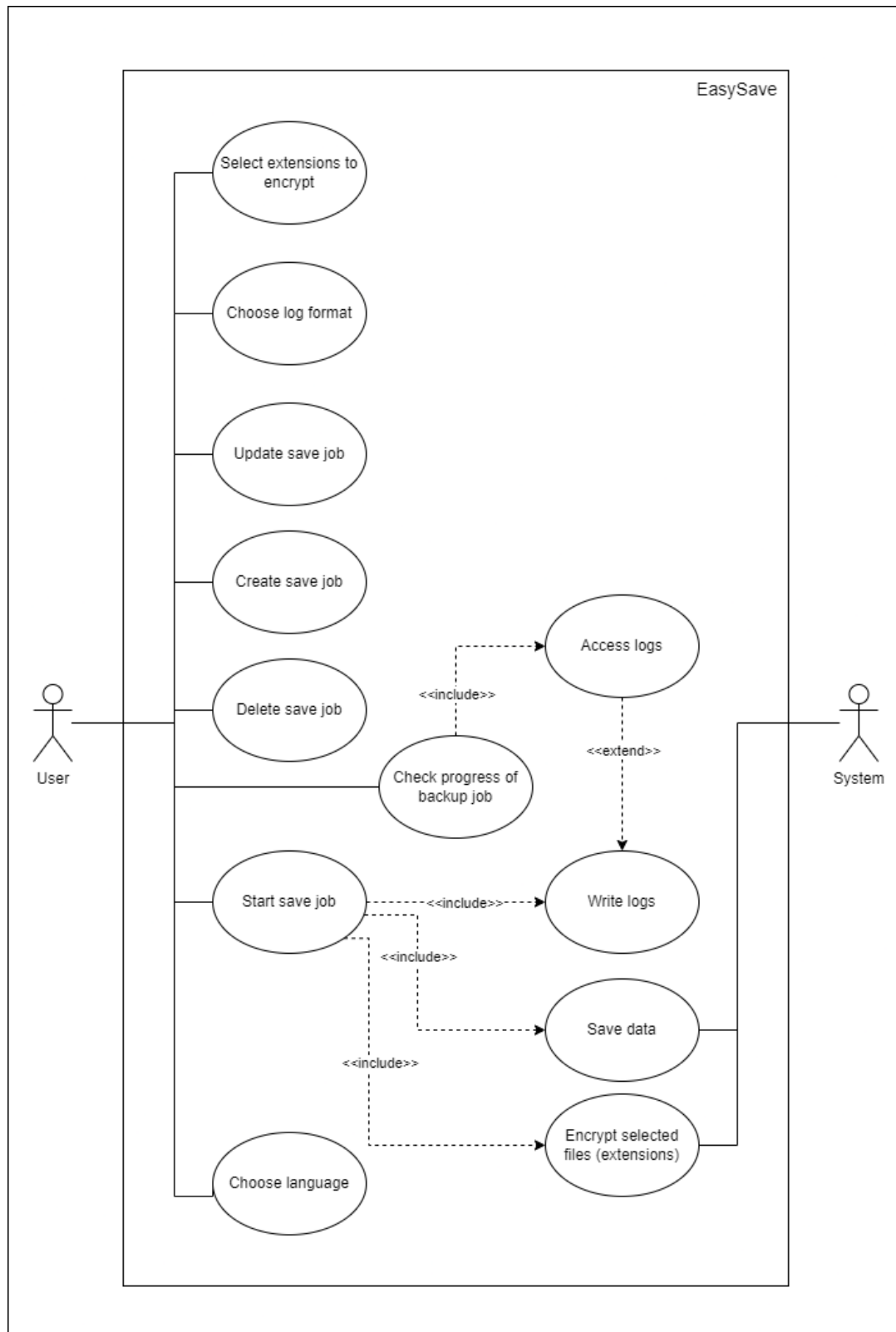


Figure 6 : Diagramme UML d'utilisation

En complément, nous avons choisis le diagramme d'activité pour décrire le flux de contrôle au sein de notre application. En représentant graphiquement les étapes séquentielles, les décisions et les conditions, ce diagramme a permis une compréhension approfondie des différents scénarios d'utilisation.

Ci-dessous en exemple, le diagramme d'activité de la version 2.0 :

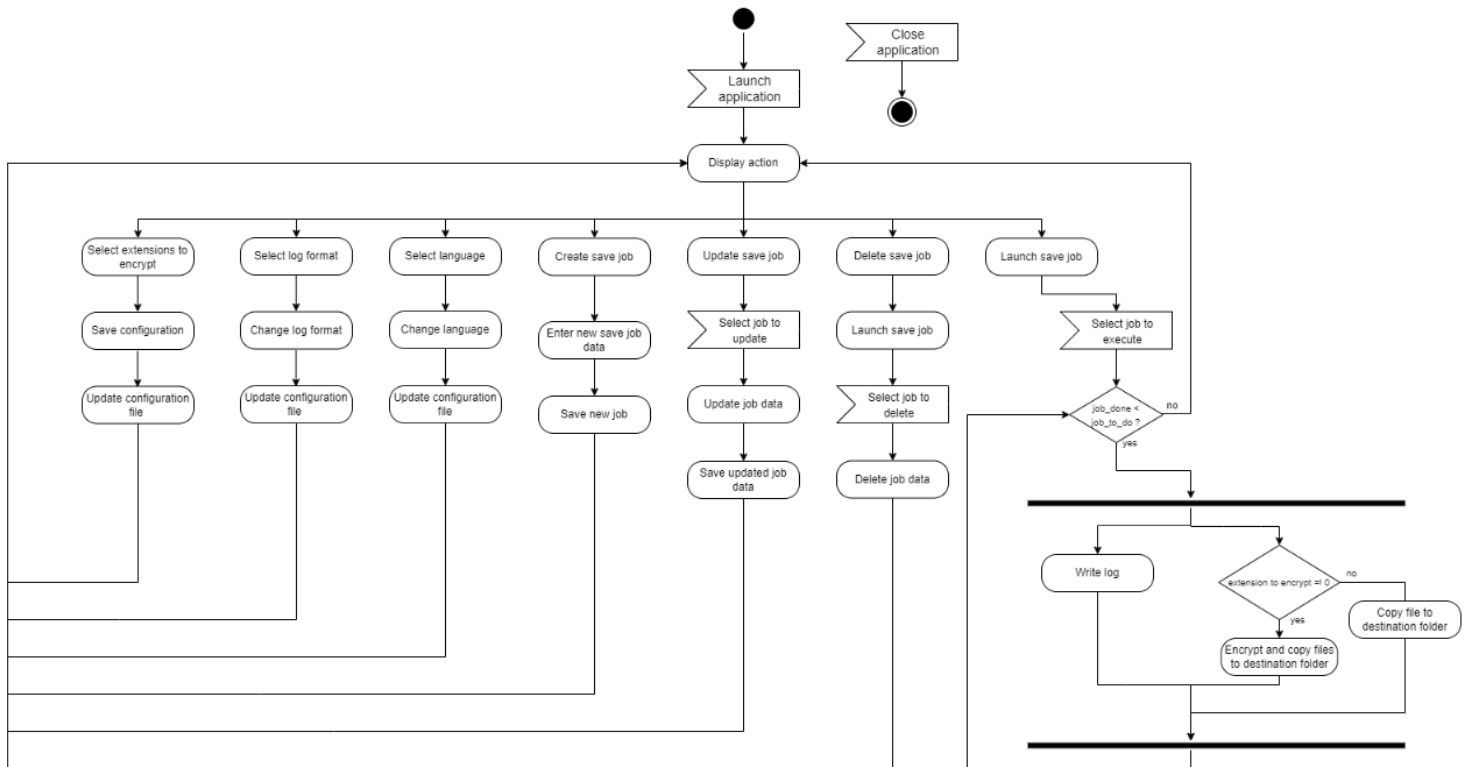


Figure 7: Diagramme UML d'activité

Également, le diagramme de séquence a été utilisé pour représenter les interactions entre les différents composants de notre système. En décrivant séquentiellement les messages échangés entre les objets, ce diagramme a permis une visualisation précise du comportement dynamique de notre application.

Ci-dessous en exemple, le diagramme de séquence de la version 2.0 :

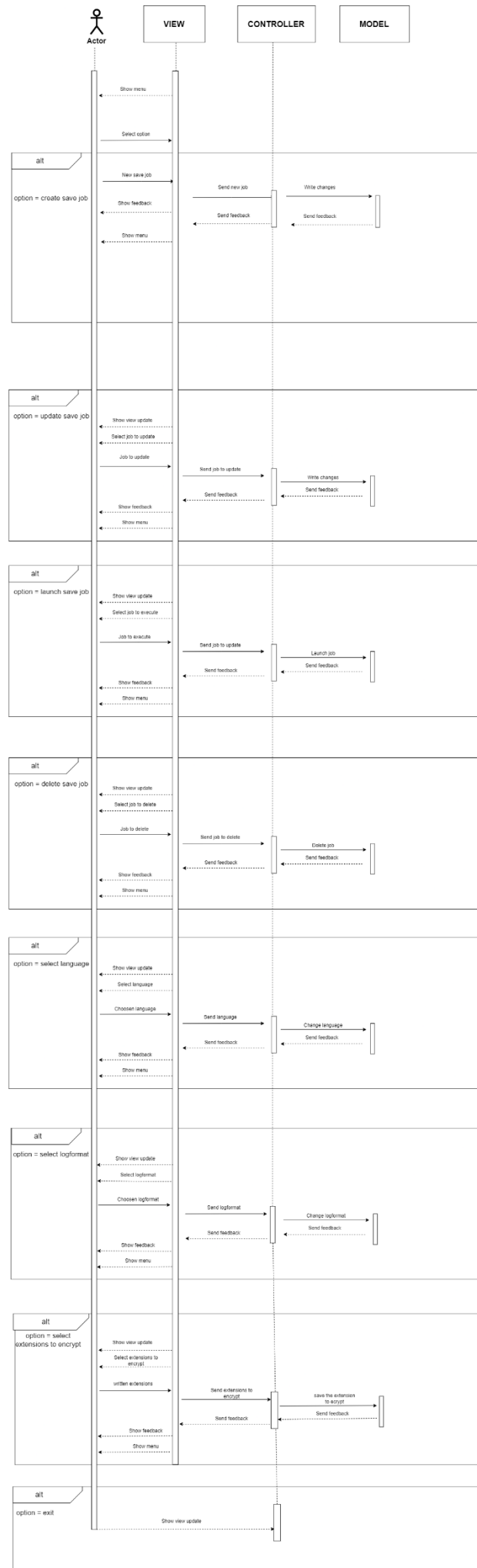


Figure 8 : Diagramme UML de séquence

Enfin, le diagramme de classe que nous avons utilisé pour modéliser la structure de notre code application. En identifiant les classes principales, leurs attributs et leurs relations, nous avons pu conceptualiser clairement les différents objets et leur organisation au sein du système.

Ci-dessous en exemple, le diagramme de classe de la version 2.0 :

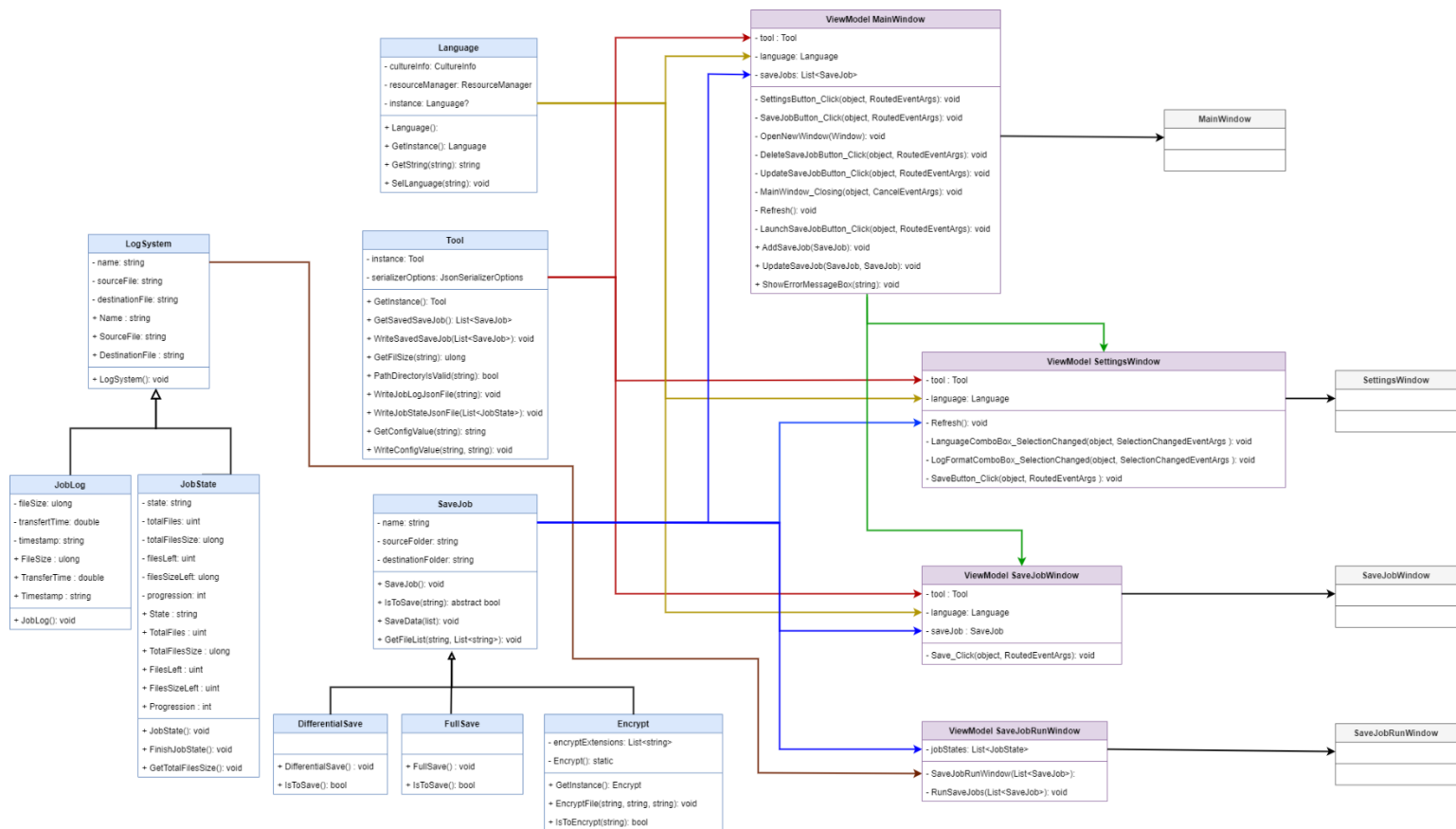


Figure 9 : Diagramme UML de classe

2. ReadMe

Un README est un document essentiel dans tout projet de développement logiciel. Il s'agit généralement d'un fichier texte qui fournit des informations importantes sur le projet, telles que son but, comment l'utiliser et le configurer. Le README se trouve généralement à la racine du dépôt du projet, le rendant facilement accessible pour toute personne qui consulte le projet.

Dans un premier temps, on trouve dans notre ReadMe une introduction concise qui présente EasySave. Ensuite, un sommaire organisé offre une vue d'ensemble claire des différentes sections, facilitant la navigation à travers le document.

Ci-dessous le sommaire :

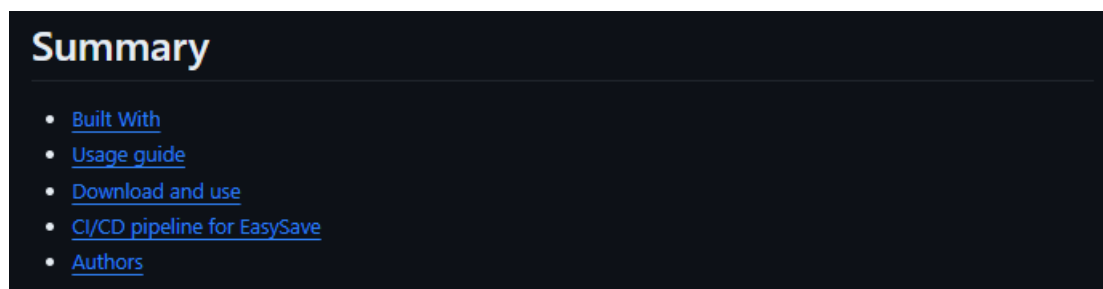


Figure 10 : Sommaire ReadMe

Dans « Built With » on retrouve les technologies et les frameworks utilisés pour développer l'application. Ici, on découvre que le projet est construit avec .NET et C#, offrant une idée claire des compétences et des environnements nécessaires pour contribuer ou comprendre le code du projet. Cela établit également les prérequis techniques pour toute personne souhaitant installer ou modifier l'application.

Ensuite, « Usage guide » fournit une explication détaillée de la manière d'utiliser EasySave, allant de la navigation dans l'interface utilisateur à l'exécution des fonctions clés comme la création, l'exécution, la mise à jour, et la suppression des travaux de sauvegarde. Elle décrit également comment ajuster les paramètres de l'application, comme la langue et le format des logs. Cette partie est essentielle pour aider les utilisateurs finaux à tirer le meilleur parti de notre application.

Puis « Download and use », cette section du README guide les utilisateurs à travers le processus de téléchargement et d'installation d'EasySave. Elle fournit des instructions spécifiques pour les utilisateurs de Windows et Linux, et des précisions des différentes étapes à suivre. De plus, des informations sur l'utilisation de l'application via Docker sont incluses, pour ceux qui préfèrent des environnements de conteneurisation.

Pour « CI/CD Pipeline for EasySave », nous avons une description de la pipeline CI/CD mise en place pour EasySave en utilisant GitHub Actions. Cette section explique comment chaque modification sur la branche principale passe par un processus de construction, de test, et de déploiement, assurant ainsi la qualité et la fiabilité du logiciel. Elle donne un aperçu des différentes étapes de la pipeline, y compris la construction, les tests, la publication, et la création de releases.

Finalement, on retrouve la sections « Authors ». Cette dernière nous rend hommage en tant que contributeurs au projet EasySave. Elle liste nos noms et nos alias GitHub, permettant une reconnaissance pour notre travail et facilitant la communication directe pour les feedbacks ou les contributions de la communauté. Car en tout état de cause, nous sommes ouverts à tous retours constructifs de la part des utilisateurs de l'application.

Chaque partie du sommaire de notre README est méticuleusement conçue pour offrir une compréhension complète du projet, depuis sa construction technique jusqu'à son utilisation pratique.

3. Code et Architecture Logicielle

Dans notre projet de développement logiciel, l'adaptabilité et l'évolution jouent un rôle crucial. Nous avons dû faire preuve d'une certaine remise en question quant à nos précédents choix, notamment notre choix d'architecture logicielle.

En effet, nous avons dans un premier temps choisi d'utiliser l'architecture MVC, cette dernière était idéale pour notre application alors qu'elle fonctionnait uniquement en console. Cette architecture divise l'application en trois composants principaux : le Modèle (gestion des données et de la logique métier), la Vue (présentation des données à l'utilisateur), et le Contrôleur (traitement des entrées utilisateurs et liaison entre la Vue et le Modèle). L'avantage principal de MVC est sa simplicité et son efficacité pour des applications en console, où l'interaction utilisateur est relativement simple et directe.

Cependant, avec l'évolution de notre application vers une interface graphique WPF plus sophistiquée, l'architecture MVC n'était plus aussi adapté pour le projet. C'est ici que nous avons opté pour un changement d'architecture pour choisir du MVVM, offrant une solution plus adaptée pour les applications WPF. En effet, WPF est fortement orienté vers le binding de données, une caractéristique qui permet de lier les éléments de l'interface utilisateur aux propriétés du modèle de données. MVVM prend pleinement en charge cette caractéristique en permettant un binding de données facile et efficace, réduisant ainsi le code nécessaire pour synchroniser la vue et les données. On retrouve aussi des avantages quant à la séparation des préoccupations, dans MVVM la Vue est complètement séparée du VueModèle, permettant de se concentrer sur l'interface utilisateur sans affecter la logique métier. Mais également la testabilité, les modifications dans la logique d'affichage ou dans la logique métier peuvent être effectuées avec moins d'impact sur l'autre partie, ce qui est particulièrement utile pour les applications WPF.

Comme nous le voyons, pour un projet WPF, MVVM est généralement plus adapté que MVC pour ces différentes raisons.

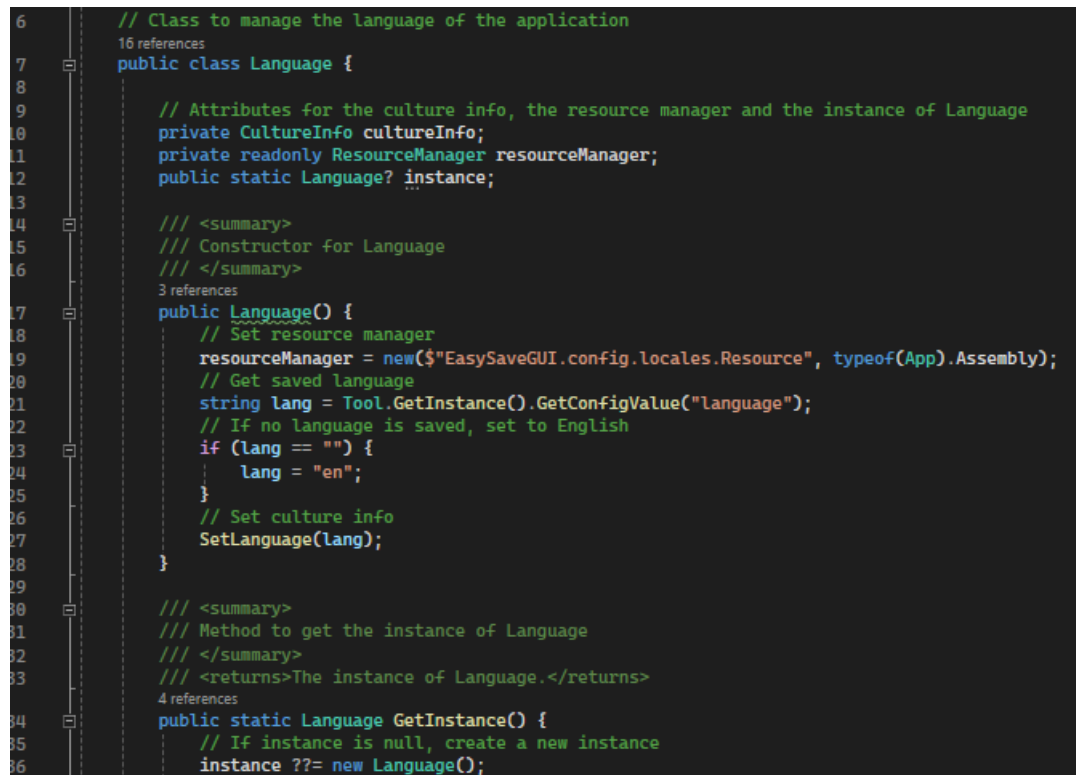
Pour comprendre un peu plus en détail le fonctionnement de MVVM, il faut connaître son organisation structurelle. MVVM se compose de trois éléments clés : le Modèle (données et logique métier), la Vue (interface utilisateur), et le VueModèle (un intermédiaire qui relie la Vue au Modèle). Cette architecture tire parti de la liaison de données (data binding) de WPF, permettant une communication fluide entre la Vue et le VueModèle. Le VueModèle agit comme un intermédiaire non seulement pour traiter les données, mais aussi pour gérer les commandes et les événements, facilitant ainsi une séparation plus nette entre la logique de l'interface utilisateur et la logique métier.

Notre transition de MVC à MVVM était donc une décision stratégique pour s'aligner sur les exigences qu'une interface graphique WPF demande. Cette évolution nous a donc permis d'améliorer à la fois la structure et la qualité de notre code.

Outre la migration d'architecture, nous avons également porté une attention particulière à la documentation et aux commentaires du code. Commenter le code consiste à écrire des explications claires et concises dans le code source, tandis que la documentation fournit une vue d'ensemble et une compréhension de haut niveau de l'architecture et des composants de l'application. Cela présente de nombreux avantages comme faciliter la maintenance, et ainsi permettre aux nouveaux développeurs ou membres de l'équipe de comprendre plus facilement le code. Mais aussi une amélioration de la collaboration, une bonne documentation et des commentaires clairs permettent une meilleure collaboration entre les différents membres de l'équipe. Une réduction d'erreurs est également notable, la

compréhension claire de chaque partie du code peut aider à prévenir les erreurs et les malentendus.

Ci-dessous un exemple de nos commentaires dans notre code :



```

6 // Class to manage the language of the application
7 public class Language {
8
9     // Attributes for the culture info, the resource manager and the instance of Language
10    private CultureInfo cultureInfo;
11    private readonly ResourceManager resourceManager;
12    public static Language? instance;
13
14    /// <summary>
15    /// Constructor for Language
16    /// </summary>
17    public Language() {
18        // Set resource manager
19        resourceManager = new($"EasySaveGUI.config.locales.Resource", typeof(App).Assembly);
20        // Get saved language
21        string lang = Tool.GetInstance().GetConfigValue("language");
22        // If no language is saved, set to English
23        if (lang == "") {
24            lang = "en";
25        }
26        // Set culture info
27        SetLanguage(lang);
28    }
29
30    /// <summary>
31    /// Method to get the instance of Language
32    /// </summary>
33    /// <returns>The instance of Language.</returns>
34    public static Language GetInstance() {
35        // If instance is null, create a new instance
36        instance ??= new Language();
  
```

Figure 11 : Commentaires Code

Nous avons fait en sorte de débiter chaque classe et méthode par un commentaire résumé, encapsulé dans les balises « <summary> » pour fournir une vue d'ensemble de son rôle. Sinon nous avons laissé des commentaires là où nous avons jugé qu'il était nécessaire d'apporter plus de précision et de compréhension. Nos commentaires sont en anglais afin d'offrir une meilleure accessibilité, de plus l'anglais est la langue principale de la programmation et est largement utilisé dans la communauté informatique à travers le monde. Concernant la documentation tout se trouve dans le ReadMe que nous avons vu précédemment (également écrit en anglais).

L'ordonnancement ainsi que le fonctionnement de nos commentaires et de notre documentation sont conçus pour assurer la clarté, la maintenabilité et la facilité de compréhension pour tous les membres de l'équipe de développement. En suivant ces principes, nous nous assurons que chaque partie de notre code comme le fonctionnement global de notre application soit expliqué de manière appropriée. Contribuant ainsi à une base de code saine, facile à gérer et appréhender.

V. Pipeline CI/CD

Le principal objectif d'une pipeline CI/CD (Continuous Integration/Continuous Delivery) est d'automatiser et d'optimiser le processus de développement logiciel, du moment où le code est modifié jusqu'à son déploiement en production.

Durant notre projet, nous allons donc devoir développer notre application. Afin d'assurer un environnement de travail professionnel permettant le bon développement de notre projet, nous allons devoir mettre en place certaines bonnes pratiques. Ces dernières auront pour objectif de suivre l'intégration des parties développées par chaque membre de l'équipe. Permettant ainsi d'éviter des conflits de versions, de valider le bon fonctionnement du code et en général de garantir une certaine qualité, rapidité et fiabilité tout au long du cycle de vie de notre application.

1. Intégration Continue

L'intégration continue est la première étape cruciale d'une pipeline CI/CD. Son objectif principal est d'automatiser et de rationaliser le processus d'intégration des modifications apportées au code source dans un référentiel partagé. Lorsqu'un développeur pousse des modifications, la CI déclenche automatiquement des étapes telles que la compilation du code, l'exécution de tests automatisés, et l'analyse statique du code. L'objectif principal de cette automatisation est de détecter rapidement les potentielles erreurs de développement et/ou incompatibilité, pour ainsi assurer une base stable pour les phases de développement ultérieur.

Cette pratique vise à garantir une intégration harmonieuse des contributions individuelles des membres de l'équipe au sein du projet. Ainsi, la collaboration entre les membres se retrouve simplifiée. En détectant par le biais de test les problèmes dès leur apparition, la CI continue à réduire les risques liés à l'intégration tardive, permettant une progression rapide du projet et des cycles de développement plus courts. D'un point de vue financier, résoudre un problème en phase de développement est bien moins coûteux qu'une fois l'application lancée, mais aussi bien plus simple.

2. Déploiement Continue

Le déploiement continu constitue la phase suivante, se concentrant sur la livraison automatisée et fiable du logiciel après la phase d'intégration. Il consiste à déployer les nouvelles modifications de code dans un environnement de production dès qu'elles passent les étapes de test et de construction. Cela permet d'avoir une chaîne d'automatisation rapide et efficace pour ajouter à l'application des nouvelles fonctionnalités et des correctifs de bugs par exemple.

La CD automatisée comprend généralement des étapes telles que la création de builds déployables, la validation de ces builds à travers des tests de déploiement, et finalement le déploiement automatisé dans l'environnement de destination. L'objectif est de minimiser les interventions manuelles dans le processus de déploiement, réduisant ainsi les erreurs potentielles et accélérant le cycle de livraison.

Le déploiement continu est essentiel pour répondre au besoin croissant de livraison version dans le code, tout en garantissant une certaine fiabilité de test. Pouvoir déployer rapidement des fonctionnalités ou des correctifs est important, mais le faire en gardant la stabilité du système l'est tout aussi. C'est donc le déploiement continu qui va garantir cette certaine rapidité tout en évitant d'ajouter d'éventuels problèmes à la suite du déploiement.

3. Github Actions

Nous l'avons vu, mettre en place une pipeline CI/CD est une pratique obligatoire à mettre en place pour effectuer un projet de développement dans les meilleures conditions possibles. Dans un premier temps nous avons utilisé Jenkins qui est un outil open source d'intégration et de déploiement continu développé en Java. Jenkins se charge automatiquement de recompiler et de tester chaque modification de code. Lors de la détection d'une erreur, il alerte le développeur afin qu'il résolve le problème.



Figure 12 : Logo Jenkins

Nous avons initialement opté pour Jenkins comme solution d'intégration continue et de déploiement continu (CI/CD). Cependant, nous avons rencontré des limitations significatives dues à la nécessité d'un environnement Windows pour le build et le test de notre application, alors que Jenkins fonctionne principalement sur un environnement Linux.

Pour contourner ce problème, nous aurions pu configurer Jenkins pour qu'il s'exécute sur Windows en utilisant un conteneur Docker non officiel fourni par Microsoft. Cette solution aurait impliqué le lancement de Jenkins via Remote Desktop Protocol (RDP) et le partage de la configuration via Docker Hub.

Toutefois, cette approche présentait un inconvénient majeur : la nécessité de télécharger régulièrement une image Docker de 5 Go, ce qui s'avérait peu pratique et coûteux en temps.

Face à ces défis, nous avons décidé de migrer vers GitHub Actions. Ce dernier, est un outil d'automatisation puissant intégré à l'écosystème GitHub, permettant aux développeurs de créer, tester et déployer leurs applications directement depuis GitHub. En tant qu'outil d'intégration et de déploiement continu (CI/CD), GitHub Actions offre une flexibilité et une facilité d'utilisation pour automatiser des workflows de développement logiciel.

Ce changement nous a offert plusieurs avantages décisifs. Premièrement, GitHub Actions supporte nativement les environnements Windows, éliminant ainsi le besoin de configurations complexes et de solutions de contournement. De plus, l'utilisation de GitHub Actions nous a libérés de la nécessité de maintenir un serveur dédié pour le CI/CD, réduisant ainsi notre charge de travail et nos coûts d'infrastructure même si un seul PC suffisait à combler cette tâche.

Avec GitHub Actions, la configuration de notre pipeline CI/CD est gérée simplement via un fichier YAML. Ce fichier définit les différentes étapes de notre processus CI/CD de manière claire et concise. Finalement, l'ensemble du processus est devenu plus facile à gérer, plus flexible et plus intégré à notre environnement de développement existant sur GitHub.

Pour mettre en place une pipeline CI/CD avec GitHub Actions, il est nécessaire de créer un ou plusieurs fichiers de workflow dans le répertoire `.github/workflows` du dépôt. Ces fichiers sont écrits en format YAML et définissent les étapes du workflow.

Chaque fichier de workflow suit une structure de base :

name: Nom du workflow.

on: Événements déclencheurs du workflow (push, pull_request, etc...).

jobs: Ensemble de jobs que le workflow doit exécuter. Chaque job peut s'exécuter sur un runner différent et peut contenir plusieurs étapes.

steps: Étapes d'un job, pouvant inclure l'exécution de scripts, l'utilisation d'actions préconstruites, la mise en place d'environnements, etc.

Ci-dessous en exemple, le début de notre fichier YAML (cicd.yml) :

```

1  name: EasySave_CI-CD_Pipeline_v2.0
2
3  on:
4    push:
5      branches: [ main ]
6
7  jobs:
8    build-and-release:
9      runs-on: windows-latest
10
11     steps:
12     - uses: actions/checkout@v2
13
14     - name: Setup .NET
15       uses: actions/setup-dotnet@v1
16       with:
17         dotnet-version: '8.x'
18

```

Figure 13 : Extrait fichier YAML

Nous avons ici qu'une partie du fichier, mais on peut voir que ce fichier définit une pipeline CI/CD qui se déclenche lors de chaque « push » sur la branche « main ». On y retrouve les différents jobs qui s'exécutent sur la dernière version de Windows, comme premier job on retrouve des étapes pour initialiser le dépôt et configurer l'environnement .NET nécessaire pour le build.

VI. Conclusion

En conclusion du projet "EasySave", nous avons efficacement combiné des méthodes de gestion de projet rigoureuses, comme PDCA, et une organisation d'équipe claire, avec des rôles bien définis pour chaque membre. L'utilisation judicieuse de technologies modernes telles que .NET, C# et des outils comme Docker, Jenkins et Github Action a renforcé la qualité et l'efficacité de notre processus de développement. Notre capacité, à créer et à gérer une pipeline CI/CD robuste a assuré un déploiement et une intégration continue efficaces.

Durant le projet, nous avons été confrontés à de nouvelles problématiques, cela nous a donc amenés à réfléchir à de nouvelles solutions. On retrouve comme exemple notre changement d'architecture, de MVC à MVVM, ou encore du passage de Jenkins à GitHub Action.

Ce projet ne cherche non seulement pas à répondre aux exigences techniques et commerciales de ProSoft, mais a également posé des bases pour l'évolution future du logiciel. Cela par le biais de la documentation comme le ReadMe, et les commentaires du code, assurant la clarté, la maintenabilité et la facilité de compréhension pour toute personne extérieure souhaitant travailler sur le projet. Tout au long de ce projet, nous avons consolidé nos compétences en naviguant dans des environnements technologiques complexes tout en atteignant des objectifs clairs.

VII. Webographie

<https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment>

<https://searchsoftwarequality.techtarget.com/feature/Visual-Studio-IDE-offers-many-advantages-for-developers>

<https://docs.microsoft.com/en-us/dotnet/framework/>

<https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners>

https://en.wikipedia.org/wiki/Release_notes

<https://rogerdudler.github.io/git-guide/index.fr.html>

https://tropars.github.io/downloads/lectures/DevOps/devops_11_Integration_Continue.pdf

<https://www.youtube.com/watch?v=jmn1EgMYE3Q>

<https://www.lebigdata.fr/integration-continue-definition>

<https://www.lebigdata.fr/docker-definition>

<http://www.morere.eu/IMG/pdf/virtualisation.pdf>

https://www.wavestone.com/app/uploads/2017/09/2017-SyntheseDEVOPS_VF_WEB.pdf