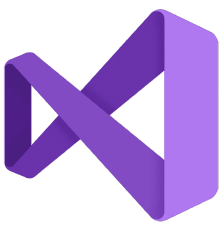
Documentation : Environnement & solutions - Projet EasySave



Introduction

Suite à la demande de Franck et Francis, nous avons été missionné afin de résoudre les problèmes qui touchent **2F Roaming Engine**. Notre travail est d'améliorer l'organisation de travail, de faciliter le développement, et la maintenance de l'application.

Nous avons les **problématiques suivantes** :

- **Baisse de productivité** car certains développeurs doivent travailler simultanément sur le même fichier source, et donc doivent attendre que l'autre ait fini de travailler sur le dit fichier source.
- Provoque des **régressions fréquentes** lors des mises à jour, où la correction d'un bug pour certains clients entraîne l'apparition de nouveaux problèmes dans d'autres fonctionnalités.
- **Fonctionnalité manquante** à la livraison.
- **Difficultés** à bien **distinguer les versions** en production, et celles en développement.
- **Difficultés de déploiement** chez les clients.

Pour résoudre cela, nous avons établis les points suivants ainsi que les applications qui peuvent venir régler ces problèmes :

- Travail en collaboration entre développeurs | **Azure DevOps, Git, Docker**
- Environnement de développement | **Azure DevOps, Visual Studio, Git**
- Gestion des versions ou versioning | **Git, Docker**
- Gestion des "build" et construction des versions | **Docker**
- Automatisation des tests unitaires | **Docker**

Rappel du cahier de charges

Notre équipe travaillera dans un environnement **.NET Core** pour développer une application console **"EasySave"** qui permettra de réaliser des sauvegardes de manière à être "User Friendly" et faciliter son utilisation tout en apportant de nouvelles fonctionnalités au fil des versions.

Le cahier de charge du projet est déjà présenté, en voici une reformulation avec quelques explications et détails :

Une sauvegarde est un répertoire possèdant un nom, un type, une source et une cible.

Un utilisateur peut exécuter une ou plusieurs tâches de sauvegarde qui seront exécutées l'une après l'autre (séquentiellement) selon l'ordre qu'il a défini.

Le programme peut être exécuté par ligne de commande (Sujet : '1-3' > *exécuter automatiquement les sauvegardes de 1 à 3* – '1;3' > *les sauvegardes 1 et 3*).

Supporter au moins le Français et l'Anglais comme langues d'interaction utilisateur.

Les répertoires sources et cibles pourront être sur disque local, externe ou lecteurs réseaux.

La sauvegarde d'un répertoire englobe l'ensemble des fichiers qu'il contient, ainsi que tous les sous-répertoires associés.

Logs journaliers (Log Exemple) : Date et heure – Nom de sauvegarde – Adresse fichier source (UNC) – Adresse fichier destination (UNC) – Taille fichier – Temps transfert fichier (ms).

Fichier État temps réel pour le suivi d'avancement des travaux de sauvegarde – Nom du travail de sauvegarde – Horodatage – État (Actif/Passif).

Étudier de l'utilisation des emplacements temporaires (C:/temp/) dans les serveurs clients pour y placer les fichiers de log et d'état.

Mettre en forme les fichiers log et état qui sont en format JSON avec une pagination.

Garder en tête qu'il y aura une potentielle version 2.0 au futur qui sera basée sur la structure MVVM pour permettre une interaction graphique avec WPF.

État du travail de sauvegarde :

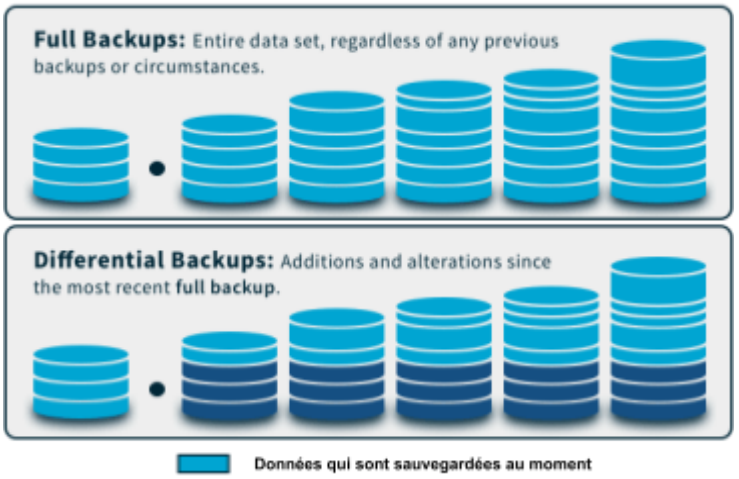
Dans le fichier d'état de sauvegarde en temps réel, afficher des détails additionnels si l'état de la sauvegarde est actif (Log Exemple) :

Nombre de fichiers éligibles (Calcul du nombre de fichiers) – Taille des fichiers (Totale) – Nombre et taille des fichiers restants – Adresse du fichier source en cours de sauvegarde + Adresse du fichier de destination.

Types de sauvegarde :

Une sauvegarde dans EasySave aura l'un des deux types suivant :

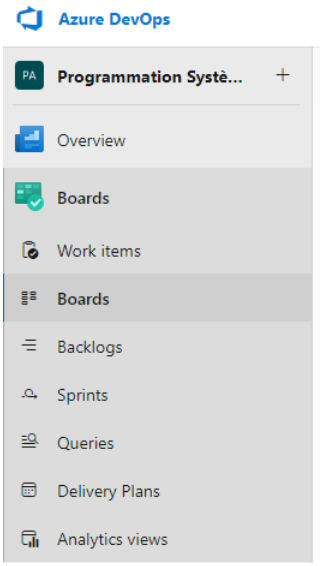
- Complète : Toutes les données disponibles sont enregistrées - Réalisée occasionnellement au vu des coûts temporels et spatiaux élevés.
- Différentielle : Uniquement les données nouvelles ou modifiées depuis la dernière sauvegarde complète sont enregistrées.



Gestion de projet

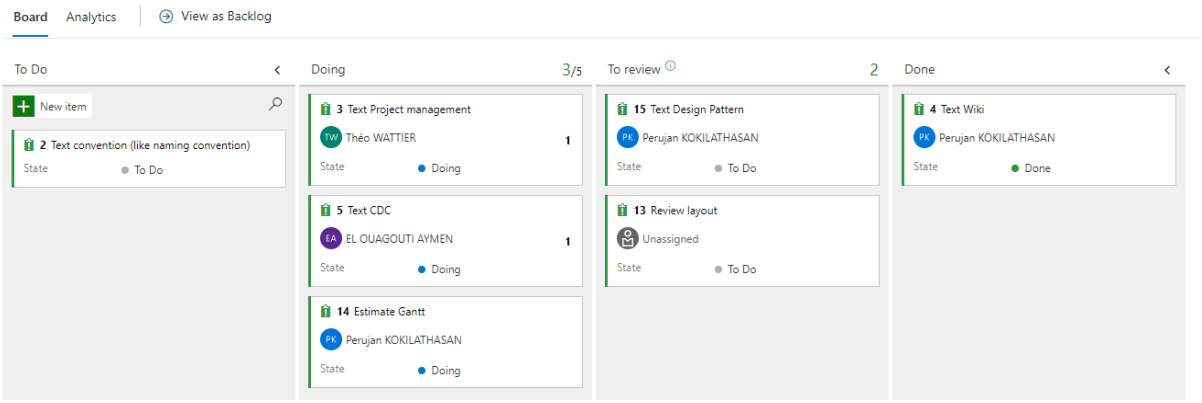
Gérer notre projet est essentiel pour assurer le succès du projet EasySave. Cela nous permet de planifier et d'organiser le projet de manière à le réaliser au mieux et dans les délais impartis, tout en minimisant les coûts et les retards.

Pour cela, nous avons mis en place une méthodologie de gestion de projet. Nous utilisons le gestionnaire de projet Azure DevOps, disponible dans l'onglet "Boards" → "Boards", où nous pouvons ajouter les différentes tâches à exécuter, les attribuer et leur donner un état selon les catégories suivantes :



- **To Do** = Tâches à faire, non commencées
- **Doing** = Tâches en cours
- **To Review** = Tâches à revoir/vérifier avec les pairs
- **Done** = Tâches terminées

Chaque utilisateur peut créer un nouvel élément (tâche), lui attribuer une priorité et la classer dans l'état correspondant à son avancement. Une fois les tâches terminées, il revient au développeur de placer sa tâche dans la colonne "To Review" ou "Done".



L'utilité de prendre en main cet outil, c'est que celui-ci est aussi bien utile pour le développement de l'application, ainsi que la gestion des erreurs et le suivi des problèmes de l'application.

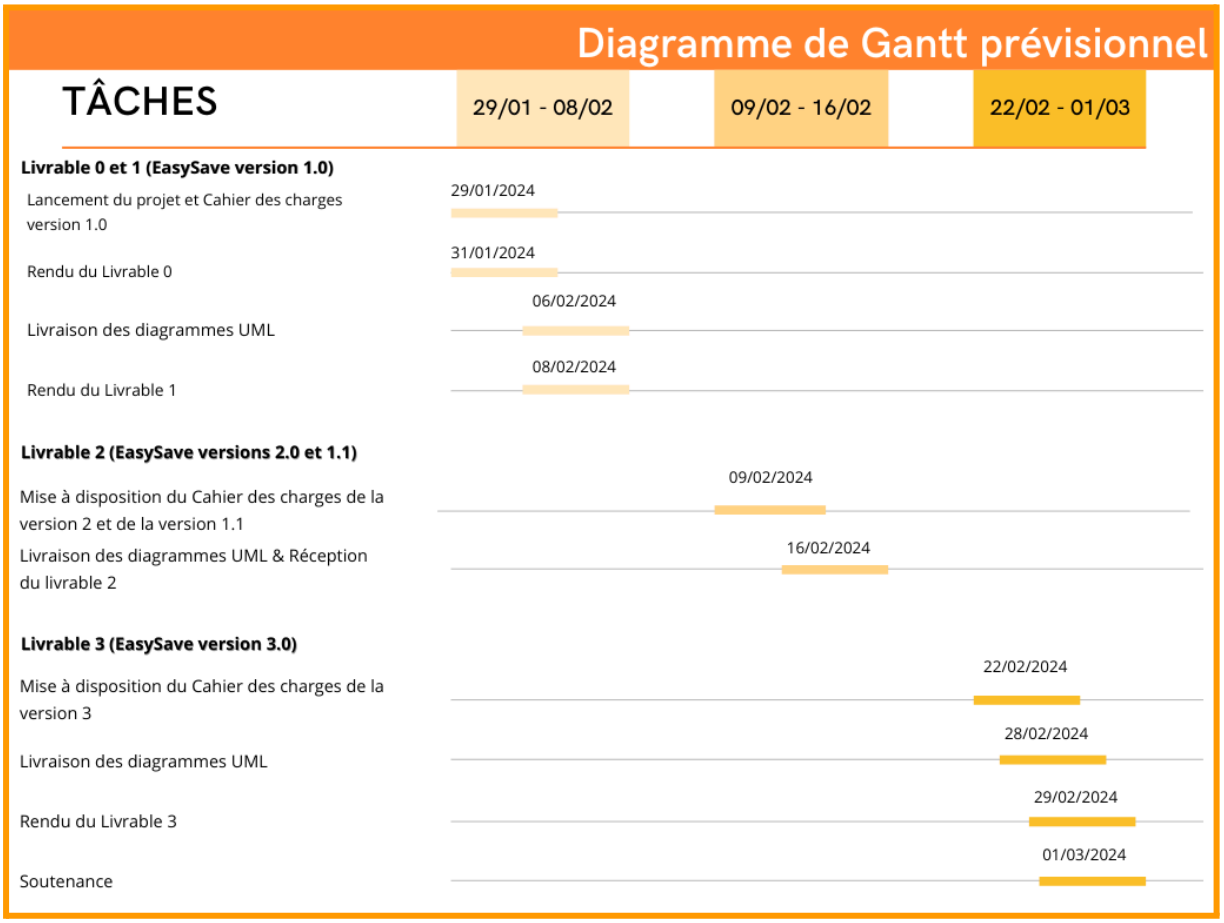
Chaque membre prenant part au projet est attribué un rôle, définissant clairement ses responsabilités, ses tâches et ses interactions avec les autres membres de l'équipe.

Au sein du groupe, nous avons au moment où nous écrivons ce document, le groupe de projet est composé de 3 membres :

- Perujan KOKILATHASAN - **Développeur**
- Aymen EL OUAGOUTI - **Développeur**
- Théo WATTIER - **Développeur & Chef de projet**

Gantt prévisionnel & Gantt actualisé

Un diagramme de Gantt est un outil de gestion de projet qui représente visuellement les tâches dans le temps. Il liste toutes les choses que nous devons faire pour un projet. C'est un plan qui permet une représentation rapide du projet. Maintenant, pour être sûrs que tout se passe comme prévu, et pour pouvoir comparer la réalisation des tâches estimées et réelles, nous allons tenir à jour un diagramme actualisé. Cela nous permettra de savoir où nous en sommes, si nous avons pris du retard quelque part, et si on doit ajuster son plan pour rester sur la bonne voie.



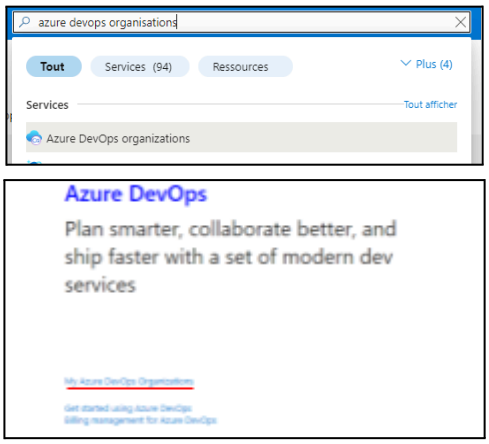
Environnement & solutions choisies

Azure DevOps offre un ensemble d'outils collaboratifs qui permettent aux développeurs de travailler efficacement en équipe. En utilisant les fonctionnalités telles que les demandes de tirage, les commentaires sur le code, et les notifications, les membres de l'équipe peuvent collaborer de manière transparente, examiner et approuver les modifications, et résoudre les problèmes plus rapidement.

Dans cette documentation, nous allons explorer la mise en place d'Azure DevOps avec Git et son intégration dans Visual Studio pour faciliter le développement logiciel collaboratif.

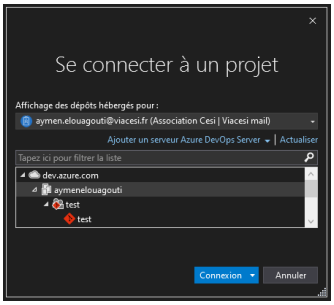
I. Configuration d'Azure DevOps

1. Création d'un projet Azure DevOps



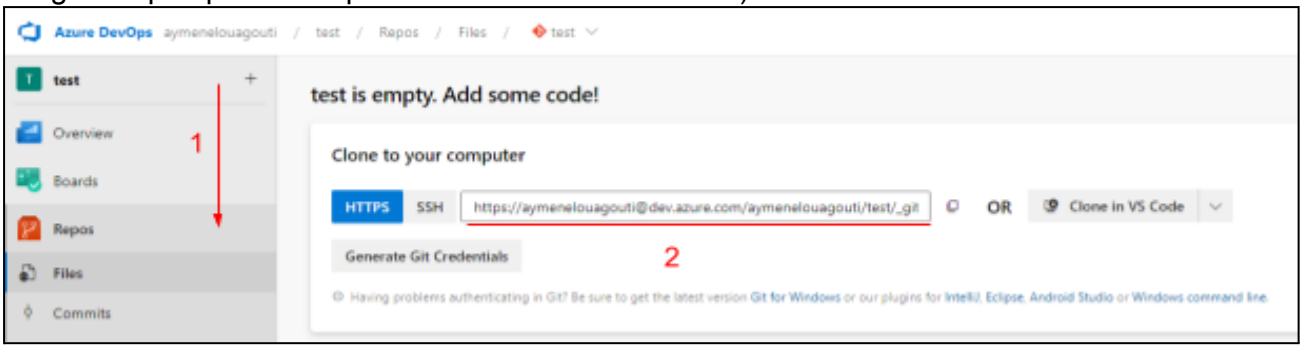
- Se connecter à son compte Microsoft Azure.
- Taper sur la barre de recherche : "Azure DevOps organizations" et y accéder.
- Suivre le lien My Azure DevOps Organizations.
- Créer une organisation ou sélectionner celle dans laquelle on souhaite créer le projet.
- Cliquer sur "Nouveau projet" et remplir les détails requis puis créer le projet.

2. Configuration de Visual Studio pour Git et Azure DevOps

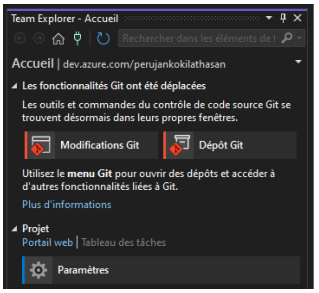


- Ouvrir Visual Studio et sélectionner "Cloner un dépôt".
- Sélectionner Azure DevOps dans "Parcourir un dépôt" et connecter son compte Azure.
- Actualiser et sélectionner l'organisation créée puis appuyer sur "Connexion".
- Choisir le chemin local pour y cloner le dépôt.
- Une fois le dépôt cloné, on peut commencer à travailler sur le code du projet.

(Il est aussi possible de cloner le dépôt à partir de l'URL, à partir de Azure DevOps dans l'onglet Repos puis en copiant le lien sur Visual Studio).



3. Utilisation de Git dans Visual Studio



Utiliser Team Explorer pour gérer les modifications, valider les changements, créer des branches, fusionner des branches, etc.

Synchroniser régulièrement le code avec le référentiel distant en poussant (push) les modifications.

Utiliser les fonctionnalités de suivi des problèmes (Git issues) et de demandes de tirage (Pull Requests) dans Azure DevOps pour une collaboration efficace au sein de l'équipe.

Utilisation et intégration de Docker

Docker est une plateforme open-source qui permet de créer, déployer et exécuter des applications dans des conteneurs. L'intégration de Docker avec Azure DevOps et Visual Studio offre de nombreux avantages. Elle assure la portabilité des applications, simplifie la gestion des dépendances et des environnements, et facilite le déploiement ainsi que la mise à l'échelle.

De plus, Docker facilite la collaboration entre développeurs en fournissant un environnement de développement cohérent et isolé pour chaque membre de l'équipe. Chaque conteneur Docker peut encapsuler une application avec ses dépendances, garantissant ainsi que tous les développeurs travaillent dans le même environnement, indépendamment de leur configuration locale. Cela réduit les conflits liés aux différences d'environnement et améliore la cohérence du code développé en équipe.

1. Configuration de Docker dans Visual Studio

Installation de Docker Desktop : Assurez-vous d'avoir Docker Desktop installé sur votre machine de développement. Vous pouvez le télécharger depuis le site officiel de Docker.

2. Création de fichiers Docker

Dockerfile : Créez un fichier Dockerfile à la racine de votre projet pour définir l'environnement d'exécution de votre application, y compris les dépendances, les variables d'environnement, et les commandes d'exécution.

docker-compose.yml : Si votre application est composée de plusieurs services Docker, vous pouvez utiliser un fichier docker-compose.yml pour automatiser le lancement, et la configuration des services utilisés.

3. Intégration avec Azure DevOps

Création de pipelines Docker : Utiliser Azure DevOps pour créer des pipelines CI/CD (Continuous Integration/Continuous Deployment) pour vos applications Dockerisées. Configurez les étapes du pipeline pour construire, tester et déployer vos conteneurs Docker.

Intégration avec les registres Docker : Azure DevOps offre une intégration native avec les registres Docker tels que Docker Hub ou Azure Container Registry. Vous pouvez publier vos images Docker dans ces registres à partir de vos pipelines Azure DevOps.

4. Avantages de l'utilisation de Docker

Portabilité des applications : Les conteneurs Docker offrent un environnement d'exécution isolé et léger, ce qui permet de déployer facilement des applications sur différentes plateformes sans se soucier des dépendances.

Gestion des environnements : Avec Docker, vous pouvez créer des images conteneurisées contenant toutes les dépendances de votre application, ce qui facilite la gestion des environnements de développement, de test et de production.

Déploiement simplifié : Les conteneurs Docker peuvent être déployés rapidement et facilement sur n'importe quelle infrastructure compatible avec Docker, qu'il s'agisse de serveurs locaux, de machines virtuelles ou de services cloud.

En intégrant Docker dans votre flux de développement avec Azure DevOps et Visual Studio, vous pouvez bénéficier d'une gestion simplifiée des applications, d'un déploiement efficace et d'une flexibilité accrue dans le développement et le déploiement de vos applications.

.NET Core & Framework

Dans notre projet EasySave, nous utiliserons .NET Core, une plateforme de développement open-source de Microsoft. .NET Core nous permettra de créer notre logiciel de sauvegarde multiplateforme, car il peut fonctionner sur différents systèmes d'exploitation comme Windows, Linux et macOS. Cela signifie que notre logiciel pourra être utilisé sur une variété d'ordinateurs, ce qui est idéal pour répondre aux besoins divers de nos utilisateurs.

.NET Core est un ensemble d'outils et de bibliothèques qui aident les développeurs comme nous à construire des applications robustes et performantes. Par exemple, .NET Core fournit des fonctionnalités pour gérer les entrées utilisateur, effectuer des calculs complexes et interagir avec d'autres logiciels. De plus, .NET Core est conçu pour être compatible avec les versions antérieures du framework .NET, ce qui signifie que nous pourrons facilement adapter notre logiciel à de nouvelles versions de .NET Core à l'avenir, sans avoir à réécrire tout notre code.

Environnement de développement

Visual Studio fournit un environnement de développement intégré (IDE) puissant qui prend en charge une large gamme de langages de programmation et de frameworks. En utilisant Visual Studio avec Azure DevOps, les développeurs peuvent bénéficier d'une intégration transparente avec les fonctionnalités de gestion de projet, de suivi des problèmes et de gestion de versions, tout en travaillant dans un environnement familier et productif.

Docker offre un moyen efficace de gérer l'environnement de développement. En utilisant des conteneurs Docker, les développeurs peuvent encapsuler leur application ainsi que toutes ses dépendances dans un environnement isolé et portable. Cela signifie que les développeurs peuvent facilement configurer leur environnement de développement en utilisant des images Docker prédéfinies, ce qui accélère le processus de configuration et garantit une cohérence entre les environnements de développement.

Bonnes pratiques

Pour assurer une gestion des versions propre et une collaboration efficace au sein de notre équipe, nous avons défini plusieurs pratiques et outils que nous allons mettre en place dans notre processus de développement.

Commits atomiques

Division des modifications en commits atomiques (commits qui respectent 3 règles : ne concerne qu'un seul et unique sujet, ne doit pas rendre incohérent le dépôt, doit avoir un message clair et concis) pour une gestion des versions plus propre.

Branching model

Adoption d'un modèle de branching comme GitFlow pour une gestion efficace des branches.
Dans notre cas, vous avons prévu de fonctionner avec 5 branches :

- Une branche **DEV**, disponible chez tous les développeurs, afin de venir développer les nouvelles fonctionnalités, donc on créera une branche par fonctionnalité qui partira ensuite sur la branche PRÉ-PROD.
- Une branche de **PRÉ-PROD**, permettant de faire un premier test du regroupement des fonctionnalités développées par les développeurs**.**
- Une branche **PROD**, dont le but est de regrouper l'application finale, que l'on va fournir au client.

Intégration continue

Configuration des builds automatisés dans Azure DevOps pour tester et déployer votre code à chaque modification. Pour plus d'informations, veuillez consulter ce lien.

Revue de code

Utilisation des demandes de tirage dans Azure DevOps pour examiner et approuver les modifications avant de merge le tirage.

Design Patterns

Les design patterns sont des solutions efficaces pour résoudre des problèmes récurrents rencontrés lors de la conception de logiciels. En adoptant ces modèles de conception, nous structurons notre code de manière à le rendre plus efficace, réutilisable et facile à maintenir. Ces modèles nous aident à résoudre des problèmes courants de manière standardisée et évitent ainsi la réinvention de la roue à chaque fois qu'un problème similaire se pose.

1. **Réutilisation du Code**

Les design patterns favorisent la réutilisation du code en proposant des solutions éprouvées à des problèmes de conception communs. Plutôt que de réinventer la roue à chaque fois qu'un problème se pose, les design patterns nous permettent d'appliquer des solutions déjà éprouvées et optimisées, ce qui réduit le temps de développement et minimise les risques d'erreurs.

2. **Mutualisation du Code**

En utilisant des design patterns, nous mutualisons le code en regroupant des fonctionnalités similaires ou récurrentes dans des composants réutilisables. Par exemple, les design patterns tels que le Singleton ou le Factory regroupent la logique de création d'objets ou de gestion de ressources dans des composants uniques, ce qui permet d'éviter la duplication de code et de simplifier la maintenance.

3. **Modularisation du Code**

Les design patterns favorisent également la modularisation du code. Par exemple, les design patterns architecturaux comme MVC (Modèle-Vue-Contrôleur) séparent clairement la logique métier, la présentation et le contrôle, ce qui facilite la gestion et l'évolution du code.

Méthode Agile

Nous avons décidé d'adopter une méthode de développement agile pour plusieurs raisons importantes.

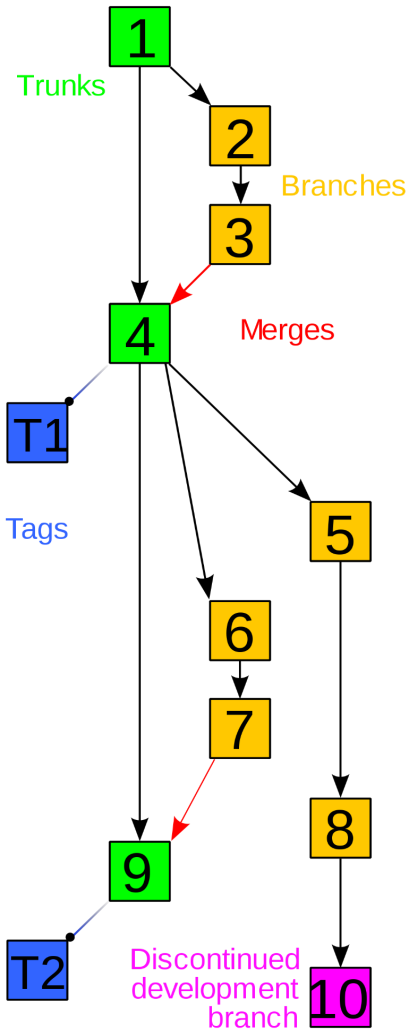
Tout d'abord, la méthode agile favorise une approche dans laquelle nous pouvons livrer des versions fonctionnelles du logiciel à des intervalles réguliers et courts, plutôt que d'attendre la fin du développement pour obtenir un produit final. Cela nous permet d'ajuster notre travail.

De plus, l'approche agile met l'accent sur la collaboration entre les membres de l'équipe et les parties prenantes du projet. En favorisant une communication régulière, ainsi qu'une collaboration active, nous pouvons identifier et résoudre rapidement les problèmes et optimiser la valeur de notre travail.

Enfin, la méthode agile offre une grande flexibilité et une capacité d'adaptation aux changements. Elle nous permet de répondre rapidement aux évolutions des exigences du projet, des priorités des utilisateurs ou des contraintes externes, en ajustant notre planification.

Gestion des versions (Versioning)

La gestion des versions, également connue sous le nom de versioning, est essentielle dans le développement logiciel pour suivre et historiser les modifications apportées au code source au fil du temps.



En utilisant Git dans Azure DevOps, les développeurs peuvent créer des commits pour enregistrer les modifications. Il peuvent créer des branches pour travailler sur des fonctionnalités isolées, puis fusionner les branches une fois les modifications terminées, et gérer les conflits de fusion de manière efficace.

Cela permet une traçabilité complète des modifications et facilite la collaboration entre les membres de l'équipe.

L'intégration de Docker dans le processus de gestion des versions offre une traçabilité complète des applications et de leur configuration.

Les fichiers Dockerfile, qui définissent l'environnement d'exécution de l'application, sont versionnés avec le code source de l'application. Cela permet de garantir que chaque version de l'application est associée à une configuration d'environnement spécifique, ce qui facilite la reproductibilité et la résolution des problèmes liés aux différences d'environnement entre les versions.

Ici, en vert, nous avons le tronc, c'est-à-dire les versions principales (version de production) de l'application.

En jaune, nous avons les versions de développement, sur les quels nous venons créer, et mettre en place des nouvelles fonctionnalités, ou de la correction de bug. Puis une fois la version validée, nous pouvons merge (fusionner) les deux versions, afin de n'en faire plus qu'une, et de la fournir au client.

Wiki

Nous avons l'intention de mettre en place un wiki pour centraliser toutes les informations importantes liées au développement de notre logiciel de sauvegarde. Ce wiki nous permet de documenter nos réunions, nos décisions architecturales avec les différents diagrammes, nos guides d'utilisation, nos procédures d'installation, des outils utilisés, références, bibliothèques et frameworks. Il s'agit d'une ressource essentielle pour notre équipe, car elle nous permet de rester organisés et de collaborer efficacement tout au long du projet.

Conclusion

Ce document à pour objectif de présenter l'environnement dans lequel nous prévoyons de développer EasySave. Pour ce faire, nous avons envisagé l'utilisation d'outils de gestion de projet adaptés au travail en équipe ainsi que des environnements spécialisés pour le développement impliquant plusieurs développeurs. De plus, nous avons évoqué des outils facilitant le déploiement et la maintenance de l'application.

Cela nous a également permis de peaufiner certaines de nos méthodes de travail, afin que tout le monde soit sur la même longueur d'onde, favorisant une collaboration plus fluide et une gestion plus intuitive de l'application.

La prochaine étape consistera à débuter le développement de l'application en utilisant les outils et les méthodologies que nous avons sélectionnés pour répondre à nos besoins.

Ressources

Création du repository et ajout dans Visual Studio :
https://youtu.be/Tpd_TUCuk8Y?si=Q_ry2swUF015-V6B

Qu'est ce qu'un commit atomique :
https://www.codeheroes.fr/2021/10/25/git-pourquoi-ecrire-des-commits-atomiques/

Qu'est ce que l'intégration continue :
https://learn.microsoft.com/fr-fr/devops/develop/what-is-continuous-integration