

Livrable 0 :

Environnement de travail



**Chef de groupe : Milan Sangare
Eugénie Hariniaina
Alexandre De Jesus Correia
Pierre Grossin
Date : 10/10/2022**

**Ecole d'ingénieur du CESI
Etudiant en Première année du cycle ingénieur informatique
93 Bd de la Seine, 92000 Nanterre**

Sommaire

Introduction	1
Les Contraintes de l'Environnement de Travail	1
.NET	1
.NET Framework :	2
.NET Core :	2
Agilité de l'équipe	2
Intégration Continue :	3
La gestion de Git (Versioning)	3
Git	3
Azure DevOps	5
Les diagrammes UML	10
UML	10
La qualité du code	10
Conclusion	11
Ressources	11
Bilan personnel	12

Introduction

Le but de notre projet est le développement de 3 versions du logiciel de sauvegarde "EasySave", en utilisant l'éditeur de logiciels ProSoft. Il faudra garantir son développement, la gestion des versions (majeurs et mineurs) et la documentation. Pour garantir la prise en main du projet par notre équipe, la direction nous impose des contraintes de travail.

En particulier, notre équipe doit installer un environnement de travail respectant ces contraintes. Le bon usage de l'environnement de travail et des contraintes imposées par la direction seront évalués tout au long du projet.

Les Contraintes de l'Environnement de Travail

.NET

La plateforme logicielle Microsoft .NET, laquelle permet en outre de gérer tous les aspects de l'exécution d'une application dans un environnement d'exécution dit « managé » :

- allocation de mémoire pour le stockage des données et des instructions du programme ;
- autorisation ou refus des droits à l'application ;
- démarrage et gestion de l'exécution ;
- gestion de réallocation de la mémoire pour les ressources qui ne sont plus utilisées.

Elle est composée de deux principaux blocs : une bibliothèque logicielle .NET et une machine virtuelle compatible Common Language Infrastructure (CLI), sous les noms de Common Language Runtime (CLR) et Dynamic Language Runtime (DLR). CLR est le composant de machine virtuelle pour le cadre .NET framework. Il s'agit de l'implémentation par Microsoft du standard Common Language Infrastructure (CLI) qui définit l'environnement d'exécution des codes de programmes. Le CLR fait tourner un bytecode nommé Common Intermediate Language (CIL).

La direction nous impose d'utiliser .NET, et nous utiliserons plus spécifiquement .NET Core.

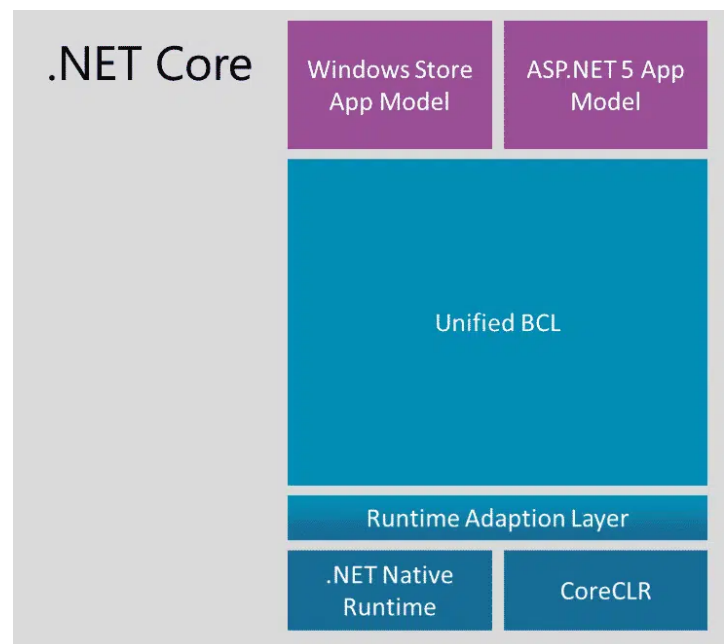
Les deux modes proposés par .NET sont .NET Core et .NET Framework, qui servent pour la création d'applications mobiles, Web et de bureau.

.NET Framework :

C'est une infrastructure logicielle développée par Microsoft qui inclut un environnement d'exécution permettant de créer des applications Windows et des services Web. Il y a des situations où son utilisation n'est pas pertinente, en particulier lorsque plusieurs plates-formes d'OS sont nécessaires ou des performances et une évolutivité élevées sont nécessaires.

.NET Core :

C'est une infrastructure open source et multiplate-forme permettant de créer des applications pour tous les systèmes d'exploitation, notamment Windows, Mac et Linux. Travailler avec des conteneurs Docker, les conteneurs et l'architecture de microservices sont souvent utilisés ensemble. Parce qu'il est léger et modulaire, .NET Core fonctionne très bien avec les conteneurs. Nous pouvons déployer des applications multi plateformes dans des conteneurs Docker, .NET Framework fonctionne avec les conteneurs, mais la taille de l'image est plus importante. C'est parce qu'il est plus léger à déployer et qu'il permet de développer en multi-plateforme que nous allons préférer utiliser .NET Core.



Agilité de l'équipe

L'agilité de l'équipe peut être atteinte en adoptant la philosophie du DevOps, c'est-à-dire mettre en place un espace de collaboration entre les équipes du développement logiciel et ceux des opérations d'infrastructure. Dans notre cas la philosophie sera pour le début plus axé Dev que Ops, tout particulièrement l'intégration continue.

Intégration Continue :

On définit cela comme une pratique de développement de logiciels où les membres d'une équipe intègrent leur travail fréquemment, généralement chaque personne intègre au moins quotidiennement.

Dans notre cas, on s'intéresse surtout au contrôle du code source car le contrôle du code source est un moyen pour garder trace et historiser les changements apportés à un code source et de collaborer sur ce dernier et Il protège notamment le code source des dommages soudains, des erreurs humaines et des différents conflits.

Git est un contrôleur de code source gratuit et open source, il permet de faire un historique complet des modifications à long terme de chaque fichier ainsi que Branching et merging mais aussi de la traçabilité.

La gestion de Git (Versioning)

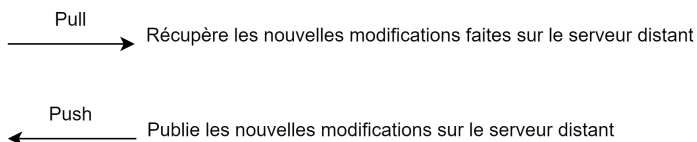
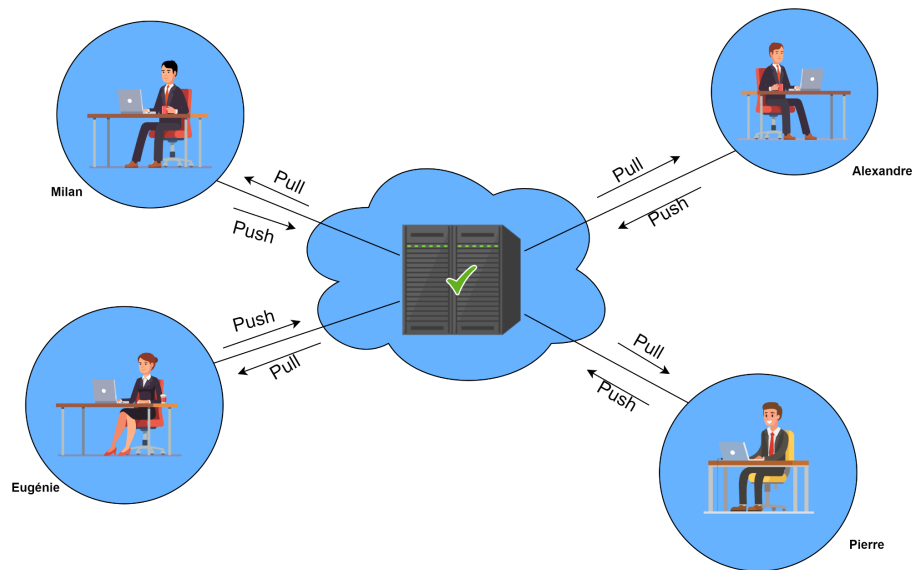
Git

Git est un logiciel de versioning permettant de conserver un historique des modifications effectuées sur un projet. Il facilite aussi le travail en groupe en s'assurant que :

- Que les fonctionnalités sur lesquelles travaillent un développeur ne rentrent pas en conflit avec les fonctionnalités sur lesquelles travaillent des autres développeurs ;
- Que chaque développeur sache sur quoi travaillent les autres développeurs afin de travailler correctement ;
- que chaque développeur possède une version actualisée du site pour tester et implémenter ses fonctionnalités.

Dans le cadre de ce projet, nous allons utiliser Git en tant que gestionnaire de code source décentralisé. Ce dernier va permettre d'historiser l'ensemble de notre code source mais de façon décentralisée.

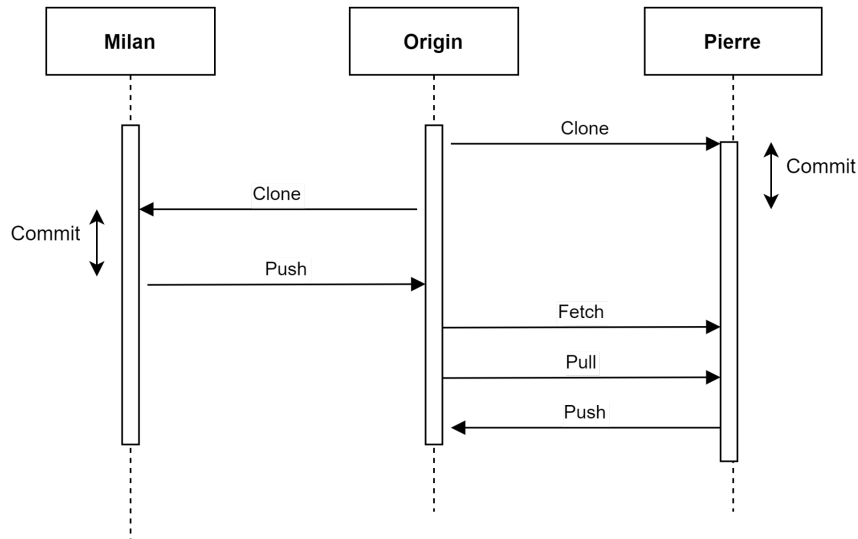
Gestionnaire de code source décentralisé



Le serveur représenté au milieu du schéma, qui va s'appeler "**origin**" par convention, va stocker l'entièreté du code source et chacun des postes de développeurs va faire une copie complète de ce dépôt présent sur le serveur.

Par exemple, Milan va récupérer l'entièreté du code développé par tous les autres développeurs et vice-versa.

Lorsqu'on parle de Git, il y a plusieurs mots clés importants, le schéma suivant va expliquer les principales commandes à devoir réaliser pour travailler dans notre environnement de travail :



En premier lieu, chaque développeur doit récupérer l'ensemble du projet par la commande “**git clone+url du chemin du projet**” pour l'avoir en local. À noter que l'url est fourni par le serveur lui-même.

De leur côté chacun va travailler donc développer ensuite faire un “**git commit**” pour confirmer que son travail pourra être remonté vers le serveur.

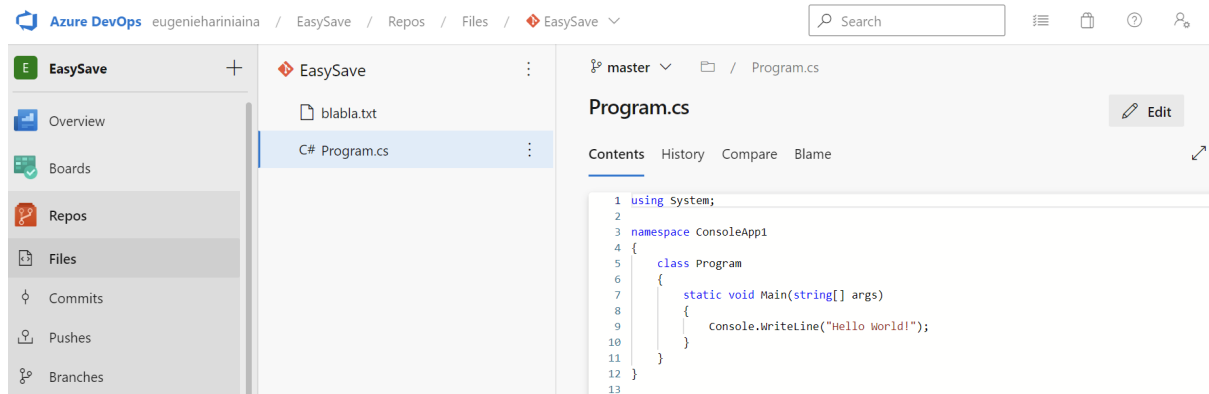
La commande “**git fetch**” télécharge des commits, des fichiers et des réfs d'un dépôt distant vers votre dépôt local. Le fetch est l'opération à réaliser lorsque vous souhaitez voir ce sur quoi tout le monde travaille. Pour pouvoir ensuite les récupérer à l'aide de la commande “**git pull**” et enfin (après avoir fini notre travail), publier ce dernier dans le serveur par la commande “**git push**”.

Azure DevOps

DevOps est un ensemble de pratiques qui met l'accent sur la collaboration et la communication entre les développeurs de logiciels et les professionnels des opérations informatiques, en automatisant le processus de livraison de logiciels et les changements d'infrastructure. DevOps permet de travailler en cohésion, en tant qu'équipe pluridisciplinaire, en se servant des mêmes outils et des mêmes référentiels.

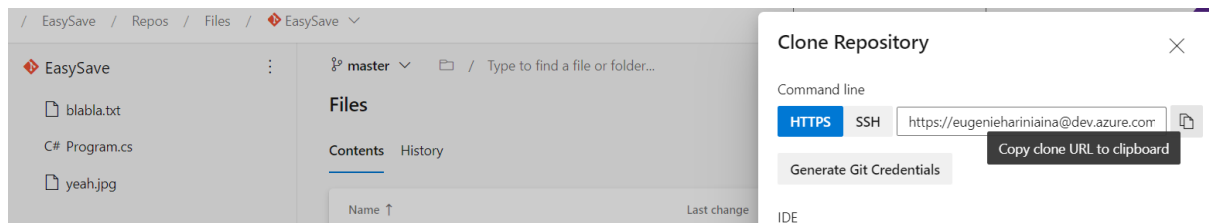
Microsoft Azure fait partie des systèmes qui permettent de faire du DevOps, notamment avec Azure DevOps.

Ci-dessous nous avons notre serveur avec Azure DevOps :



Pour avoir le projet “EasySave” en local, nous allons devoir le cloner à l’aide de la commande **git clone + url**

Nous allons d’abord récupérer l’url sur le serveur :



Ouvrir la commande Git pour exécuter le clonage :

```
C:\Users\eugen>git clone https://eugeniehariniaina@dev.azure.com/eugeniehariniaina/EasySave/_git/EasySave
Cloning into 'EasySave'...
remote: Azure Repos
remote: Found 14 objects to send. (4 ms)
Unpacking objects: 100% (14/14), 449.72 KiB | 1.28 MiB/s, done.
C:\Users\eugen>
```

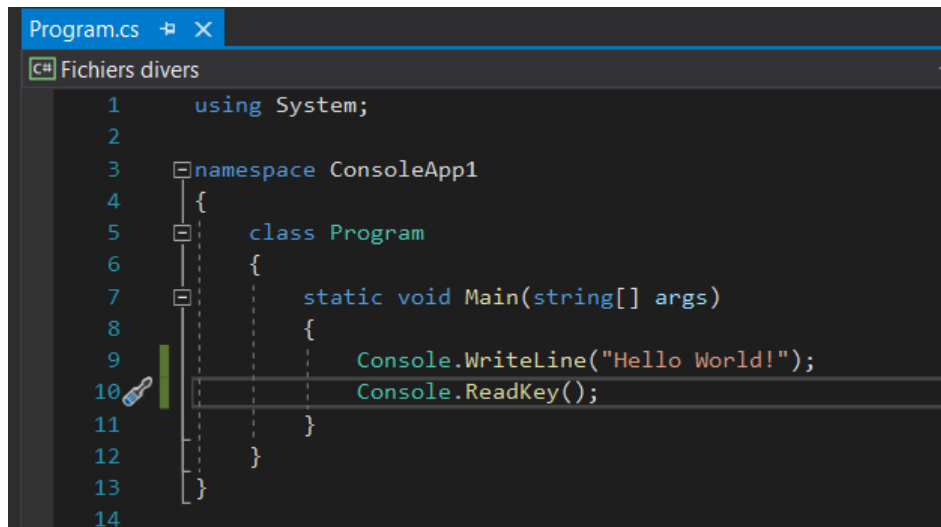
Le dossier est donc actuellement présent sur le PC local du développeur. Pour le vérifier il faut entrer dans le chemin où il se trouve puis vérifier le statut :

```
C:\Users\eugen>cd EasySave
C:\Users\eugen\EasySave>git status
On branch master
Your branch is up to date with 'origin/master'.
nothing to commit, working tree clean
C:\Users\eugen\EasySave>
```

Nous pouvons voir ici que le fichier est bien présent en local :

Ce PC > Windows-SSD (C:) > Utilisateurs > eugen > EasySave				
Nom	Modifié le	Type	Taille	
blabla	16/11/2022 19:17	Document texte	1 Ko	
Program.cs	16/11/2022 19:17	C# Source File	1 Ko	
yeah	16/11/2022 19:17	Fichier JPG	730 Ko	

Supposons maintenant que le développeur travaille sur le projet en local en ajoutant une ligne de code au fichier “Program.cs” présent dans le projet “EasySave”. En local, nous allons ajouter la ligne de code “Console.ReadKey();” qui n’est pas présent sur le serveur.



```

1      using System;
2
3      namespace ConsoleApp1
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  Console.WriteLine("Hello World!");
10                 Console.ReadKey();
11             }
12         }
13     }
14

```

Après cette modification, en exécutant la commande “**git status**” Git va afficher qu’un changement a été fait et nous invite alors soit à ajouter ce changement soit à restaurer la dernière version du projet :

```

C:\Users\eugen\EasySave>git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
       modified:   Program.cs

no changes added to commit (use "git add" and/or "git commit -a")

```

Nous allons donc l’ajouter pour pouvoir le commiter pour qu’il puisse être remonté à l’aide de la commande “**git commit -m “Ajout d’une ligne de code”**”

```

C:\Users\eugen\EasySave>git add Program.cs

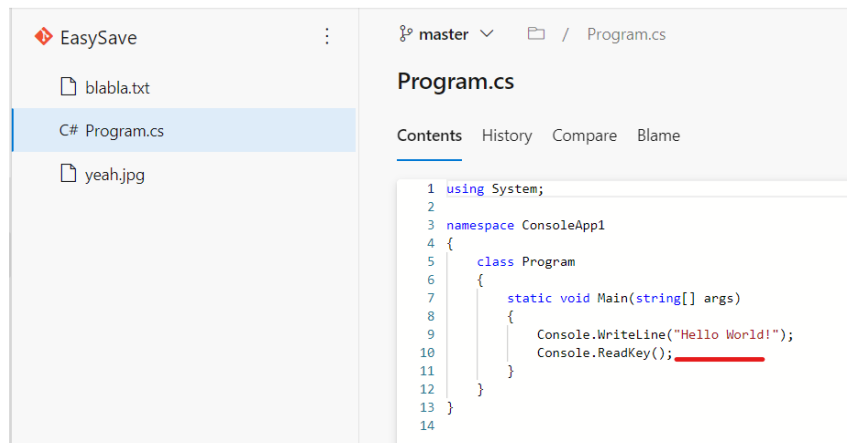
C:\Users\eugen\EasySave>git commit -m "Ajout d'une ligne de code"
[master c68b942] Ajout d'une ligne de code
1 file changed, 1 insertion(+)

```

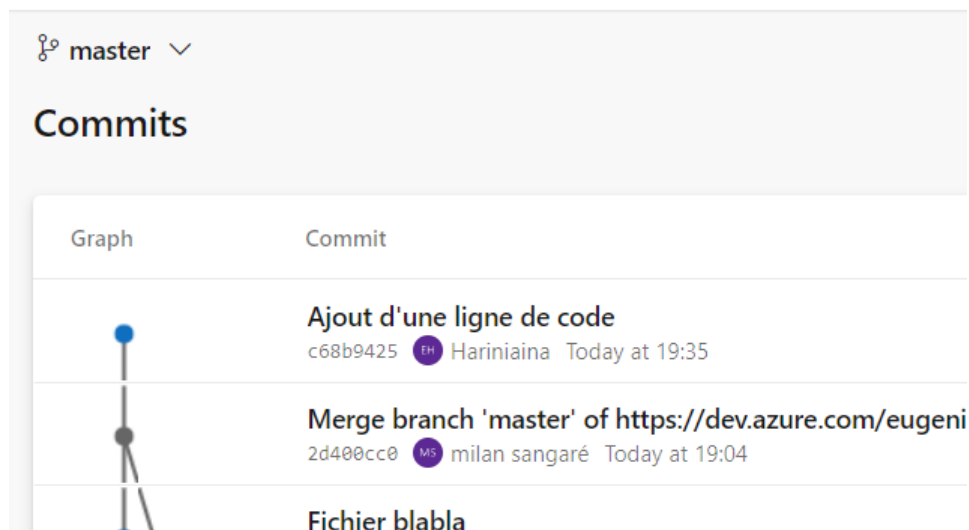
La dernière étape est donc de le publier sur le serveur en utilisant la commande “**git push origin master**” (**origin** étant notre serveur et **master** la branche correspondant à notre projet)

```
c:\Users\eugen\EasySave>git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 374 bytes | 124.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Analyzing objects... (3/3) (6 ms)
remote: Storing packfile... done (77 ms)
remote: Storing index... done (50 ms)
To https://dev.azure.com/eugeniehariniaina/EasySave/_git/EasySave
2d400cc..c68b942 master -> master
```

En revenant sur Azure DevOps, nous pouvons alors voir que la ligne de code a bien été ajoutée :



Ainsi que le dernier “commit” fait dans la branche “master” :



Supposons à présent que le nom de l'image yeah.jpg a été changé en yyeah.jpg par un autre développeur. Localement, avant chaque publication vers le serveur (push) nous devons interroger Git si quelconque changement a été fait par la commande **git fetch** et **git status** :

```
c:\Users\eugen\EasySave>git fetch
remote: Azure Repos
remote: Found 2 objects to send. (0 ms)
Unpacking objects: 100% (2/2), 315 bytes | 18.00 KiB/s, done.
From https://dev.azure.com/eugeniehariniaina/EasySave/_git/EasySave
c68b942..4444322 master -> origin/master

c:\Users\eugen\EasySave>git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)

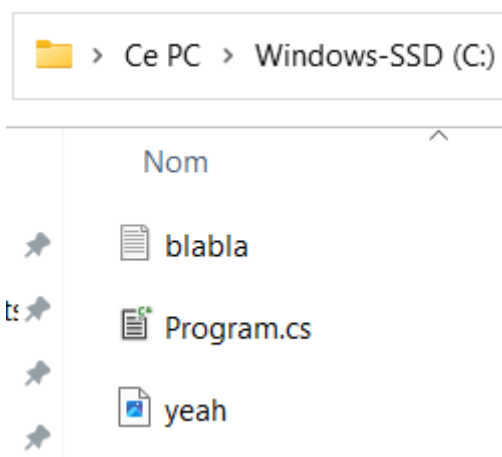
nothing to commit, working tree clean
```

La réponse indique clairement qu'un changement a été fait et nous invite à récupérer celui-ci par la commande **git pull** :

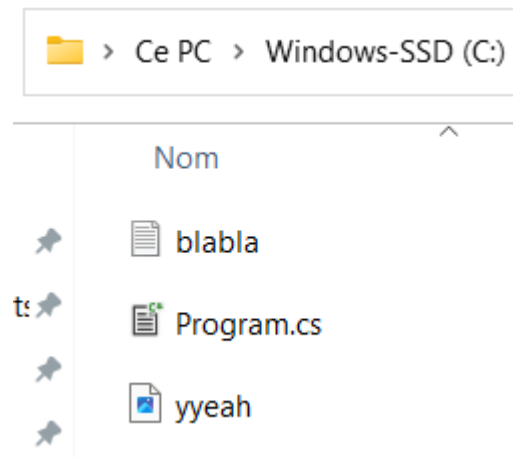
```
c:\Users\eugen\EasySave>git pull
Updating c68b942..4444322
Fast-forward
 yeah.jpg => yyeah.jpg | Bin
1 file changed, 0 insertions(+), 0 deletions(-)
rename yeah.jpg => yyeah.jpg (100%)
```

Après la récupération, il est bien indiqué que le changement de nom a été récupéré avec succès et en local cela donne ce qui suit :

Avant le **pull**



Après le **pull**



Git joue donc un rôle primordial pour la synchronisation de chaque développeur qui travaille sur le même projet, peu importe leur nombre.

Les diagrammes UML

UML

Un diagramme UML (Unified Modeling Language) fournit une représentation visuelle d'un aspect d'un système. C'est un outil permettant de décrire la structure d'un système en modélisant ses classes, ses attributs, ses opérations et ses relations.

Il existe 9 types de diagrammes UML, réparties dans 3 familles :

- les diagrammes statiques (diagrammes de classes, d'objet et de cas d'utilisation)
- les diagrammes dynamiques (diagrammes d'activité, de collaboration, de séquence, d'état-transitions et de cas d'utilisation)
- les diagrammes d'architecture : (diagrammes de composants et de déploiements)

Le langage UML, avec lequel sont construit les diagrammes, permet de :

- simplifier la complexité.
- assurer une bonne communication.
- automatiser la production de logiciels et les processus.
- résoudre les problèmes architecturaux récurrents.
- améliorer la qualité du travail.
- diminuer les coûts et le temps de mise sur le marché.

Pour pouvoir créer des diagrammes UML, on peut installer l'outil ArgoUML.

La qualité du code

La qualité du code doit faire partie des objectifs principaux d'un projet. Un code sans erreurs qui exécute de manière fiable sa fonction prévue peut la définir.

Cependant, certaines propriétés garantissent une haute qualité de code : il ne faut donc pas les omettre. On peut alors citer la maintenabilité du code, sa robustesse, sa testabilité, sa complexité ou encore sa sûreté.

Ces propriétés ont toutes un impact sur le succès du produit, la qualité du logiciel et la longévité, y compris les coûts de main-d'œuvre et le délai de mise sur le marché : rationaliser le processus de développement, augmenter les normes de qualité et améliorer l'analyse du code permet un déroulement efficace du projet.

Il faudra notamment être vigilant sur la maintenabilité et la lisibilité du code :

- l'ensemble des lignes de codes doit être exploitable par les filiales anglophones (ceci sera valable aussi pour la documentation et les commentaires).
- le nombre de lignes de code dans une fonction doit être raisonnable.
- la redondance (ou répétition inutile) des lignes de code est interdite (en particulier, il faudra faire attention aux copier-coller).
- respecter des conventions de nommage.

Ainsi, la programmation est claire, cohérente et bien documentée, de sorte que l'on peut facilement et efficacement le déboguer, le mettre à jour ou lui ajouter des fonctionnalités.

Conclusion

Dans ce livrable, il est donc question d'exposer notre environnement de travail selon les contraintes de l'entreprise ProSoft. Commençant par la gestion de versionning à l'aide de l'outil Git qui va nous permettre de synchroniser nos travaux de développement via un serveur décentralisé Azure Devops. Passant par le choix d'outil de diagramme UML utile à l'établissement d'une architecture logicielle adéquate. Et terminant par les détails de l'importance de la qualité de code.

Ressources

-.NET

https://fr.wikipedia.org/wiki/.NET_Framework

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

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

-DevOps

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

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

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

- Gestion du Git / Versionning

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

[Git - git Documentation \(git-scm.com\)](https://git-scm.com/docs)

<https://www.youtube.com/watch?v=h86A6vAJWos>

- Diagramme UML

[Easy Way to Install ArgoUML - YouTube](#)

[Download Java for Windows](#)

<https://www.jmdoudoux.fr/java/dej/chap-uml.htm>

http://www.lsv.fr/~schmitz/teach/2003_AP_Java/UML/

- Qualité du Code

<https://fr.parasoft.com/solutions/code-quality/#:~:text=La%20qualit%C3%A9%20du%20code%2C%20objectifs%20principaux%20d'un%20projet.>

<https://fr.wikipedia.org/wiki/Maintenabilit%C3%A9#:~:text=En%20informatique%2C%20l'indice%20de%20logiciels%20faciles%20%C3%A0%20entretenir.>

<https://www.futura-sciences.com/tech/definitions/tech-code-redondant-1166/>

Bilan personnel

Milan Sangare:

Pour ce livrable ma contribution est les contraintes de l'environnement de travail, ainsi qu'une participation à la réalisation des autres parties.

Pierre Grossin:

Au cours de ce projet, j'ai fait la partie sur les diagrammes UML, tout en apportant mon aide sur les autres parties.

Eugénie Hariniaina:

Dans le cadre de ce projet, je me suis chargé principalement de la partie gestion de Git tout en contribuant aux autres parties.

Alexandre De Jesus Correia:

Lors de ce projet, j'ai effectué la partie qualité du code, mais j'ai aussi participé et aidé pour les autres parties.