**Utilisation de Git et GitHub**

**Présentation :**

Git est un logiciel de gestion de versions décentralisé open source. Il permet de stocker un code informatique pas uniquement en local sur l'ordinateur de l'utilisateur, celui-ci peut aussi être stocké sur un serveur dédié.

Github quant à lui est une plateforme d'hébergement de projet / de code open source elle aussi. Il permet de faciliter la collaboration autour d'un projet entre différents utilisateurs qui sont alors en mesure de le récupérer, dupliquer ou de proposer des modifications sur du code si le propriétaire du projet accepte.

**Quels sont les utilités d'un système de gestion de version ?**

L'idée est d'utilisé un serveur distant qui contiendrait un historique de toute les modifications réalisées par chaque contributeur d'un projet et ainsi mettre en commun les avancées de chacun. Le système distribué que propose Git et Github permet à chaque utilisateur d'héberger l'entièreté du code du projet sur lequel il travail sans avoir besoin d'être constamment connecté au serveur distant. De plus chaque collaborateur ayant une version propre du projet celles-ci peuvent être utilisé en tant que backup dans le cas d'une corruption totale du projet par un autre collaborateurs.

**Gestion d'un projet grâce à github :**

Il est nécessaire d'avoir git installer sur notre ordinateur comme prérequis.

Dans un premier temps nous allons nous intéresser au fonctionnement de git à l'aide de ligne de commande nous verrons par la suite qu'il existe d'autres pour se faire.

Principe de fonctionnement de git :

Les données dans git sont assimilés en réalité à un flux de snapshots, à chaque validation/enregistrement d'une session de travail sur notre projet git, celui-ci va créer un snapshot de notre espace de travail et en créer une référence afin de nous y donner accès par la suite. Ce snapshot est stocké sur notre ordinateur en local dans une base de données.

Les fichiers dans git on plusieurs états possibles dont 2 grand états à savoir : suivi (fichier enregistré sur la base de de donnée) et non suivi (non enregistré sur la base de données et donc n'appartient pas au dernier snapshot).

Chaque fichier suivi peut lui-même avoir un autre sous état :

* Modifié (modified) : lorsque le contenue du fichier et modifié comparé à celui du dernier snapshot.
* Indexé (staged) : lorsque que l'on indique à git que le fichier fera partie du prochain snapshot/
* Validé (commited) : lorsque les fichiers du snapshots sont stockés dans la base de données locale.

Chaque projet Git est composé de 3 sections distinctes :

* Le répertoire de travail (Working Tree) : Correspond au dossier qui contient réellement nos fichiers extrait de la base de données sur laquelle sont stocké nos snapshots afin de pouvoir les modifiés et/ou les utiliser.
* La zone d'index (Staging Area) : Fichier qui stocke le contenu du prochain snapshot.
* Le répertoire Git : Stocke la base de données de notre projet.

Les branches permettent de développer des fonctionnalités isolées les unes des autres. Par défaut la branche principale est la master, on en crée d'autre pour le développement que l'on fusionne ensuite à la branche principale une fois terminé.

* **Création d'un nouveau dépôt** :

Il s'agit de l'importation et d'une copie des fichiers de notre projet dans git.

* Tout d'abord on se place dans le dossier où l'on souhaite développer notre projet puis on exécute la commande : **git init**, qui va créer un sous répertoire .git qui contiendra l'ensemble des fichiers. Il est ensuite possible de commencer à coder.

* **Ajout/modification de fichier et validation de fichier** :

* Il est ensuite possible de proposer des changements sur notre code lors de la modification (ajout à l'index) de celui-ci en utilisant la commande **git add <filename>**  ou  **git add \***, pour ensuite valider ces changements on utilise la commande  **git commit -m "Message de validation"** pour ajouter le fichier tel quel au répertoire git.
* A l'aide de la commande **git status** nous pouvoir directement voir le repertoire de travail.

* **Envoyer/recevoir des changements au serveur distant :**

* Dès lors que nos changements sont dans notre repertoire de travail, il est possible de les envoyer à notre serveur distant à l'aide de la fonction **git push origin <nom de la branche>** le nom de la branche à spécifier correspond à celle dans laquelle nous avons effectué des changements.
* Pour mettre à jour notre dépôt local en fonction des derniers commits on utilise la fonction **git pull** .

* **Gestion des branches :**

* On utilise la fonction **git branch <nom de branche>** pour créer une branche.
* On utilise la fonction **git checkout -b <nom de branche>** pour se créer et se placer dans la branche en question.
* On utilise la fonction **git checkout master** pour se resituer dans la branche principale.

Dans le cas où l'on souhaite fusionner 2 branches on se place dans la branche 1 puis on utilise la fonction **git merge <nom de la branche 2>** qui va permettre de faire avancer la branche 1 au dernier commit de la branche 2 on appel cette action **fast forward**.

* A la suite d'un merge d'une branche 1 dans une branche 2 on peut supprimer la branche 2 en utilisant la fonction **git branch-d** .

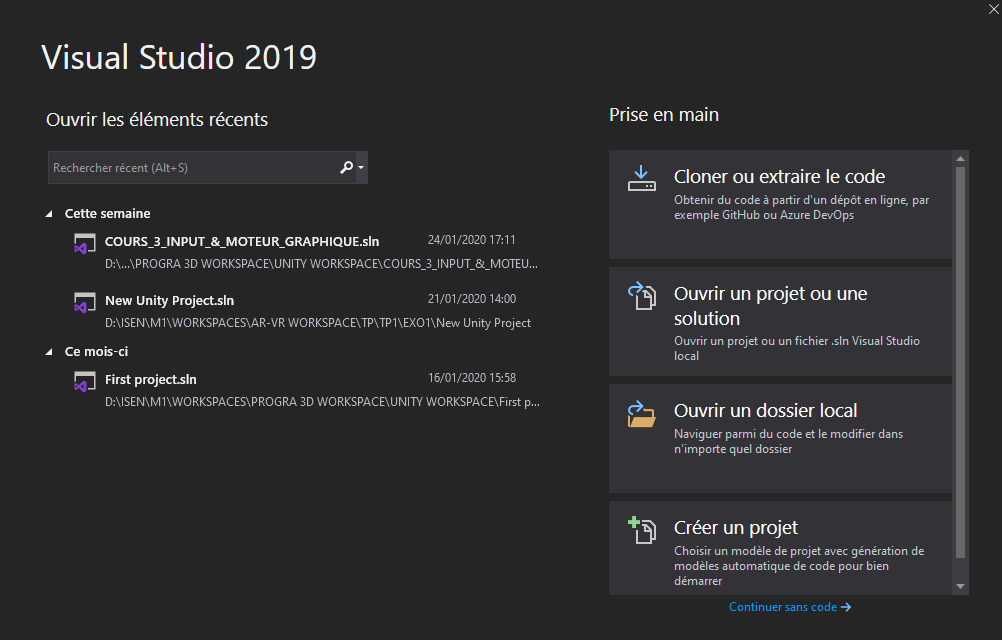
* **Consultation et gestion de l'historique :**

* On utilise la commande **git log** pour afficher la liste de tous les commits, du plus récent au plus ancien. Il est possible de rajouter l'argument **-p** à cette commande pour voir les différences entre chaque validation. Il exsite d'autres arguments possible tel que **--pretty / --since / --author**.
* Il est possible d'annuler la validation d'un commit pour se faire on utilise **git commit -- amend** qui généré un commit qui remplacera le précédant en l'écrasant.
* Github offre la possibilité de pouvoir revenir à un état antérieur du projet dans le cas ou certaines modification apportés aux fichiers de notre projets ne nous conviennent pas, il est possible d'utiliser 2 commandes : **git checkout restore <nom du fichier>**
* **Contribution à un projet github (étapes) :**

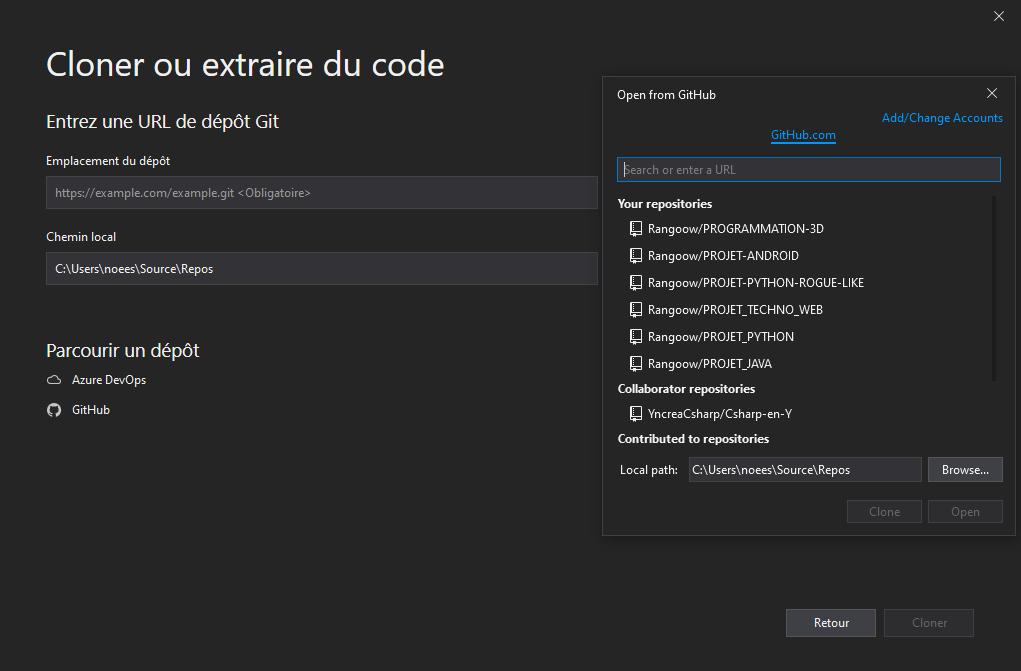
1. On copie un projet à l'aide de la commande **git clone / git fork** ;
2. On créer une branche dédiée à notre travail à partir de master ;
3. On effectue des modifications / améliorations sur le projet ;
4. On **commit** ensuite la branche locale vers le projet qu’on a copié sur GitHub et on lance un **pull request** depuis GitHub ;
5. Le propriétaire du projet choisit alors entre refuser nos modifications ou de les fusionner dans le projet d’origine ;
6. On met à jour notre version du projet en récupérant les dernières modifications à partir du projet d’origine.

Remarque : à l'aide d'un **git fork** il existe un lien entre notre copie et le projet d'origine ce qui permet de proposer plus facilement des **pull request** et ainsi participé au projet, tandis que le **git clone** ne fait que créé un copie locale du projet original.

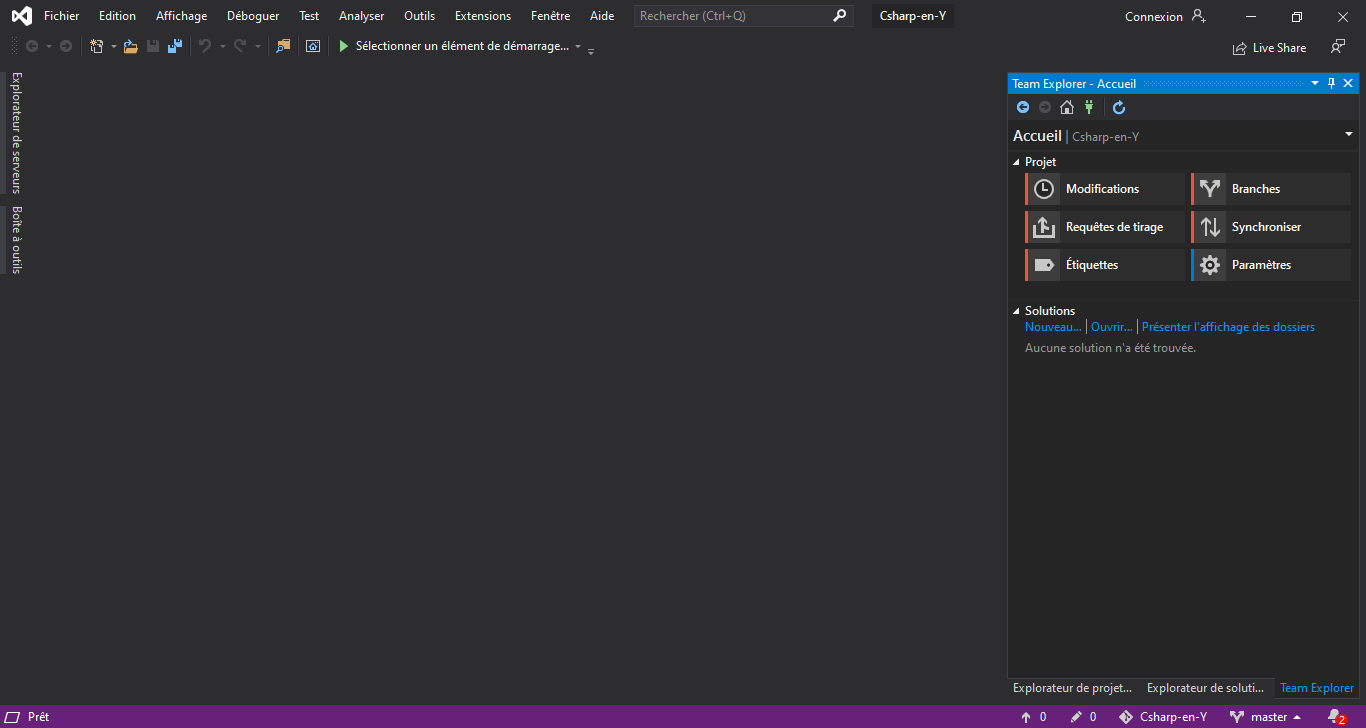
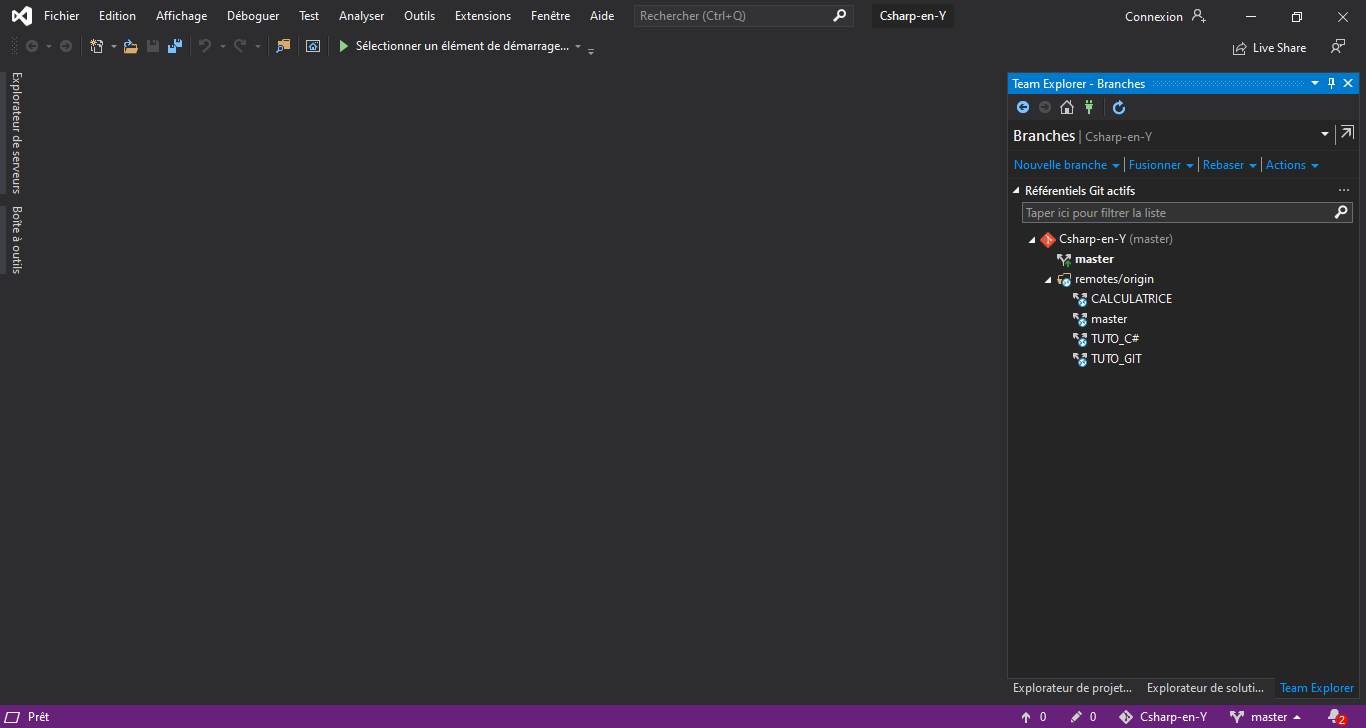
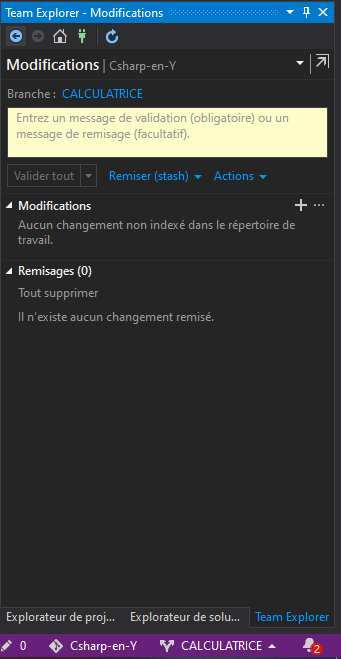
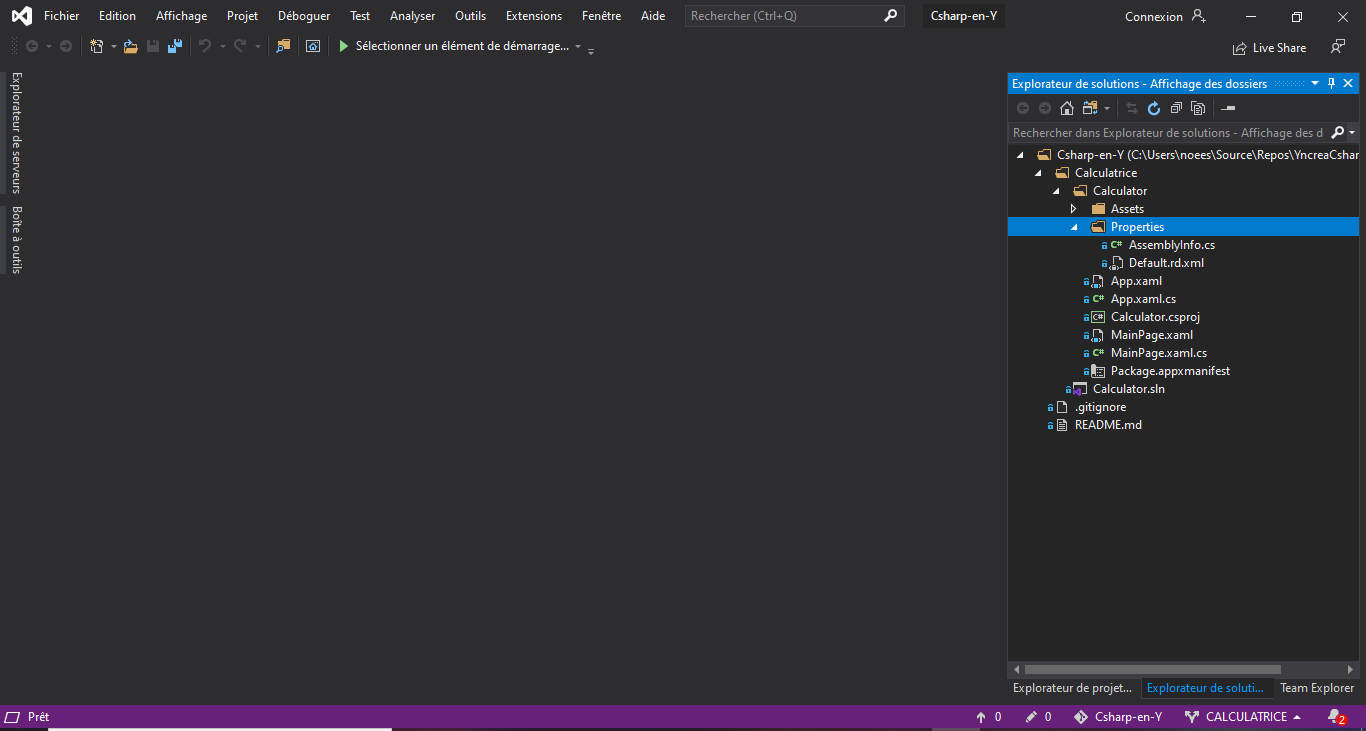
Git et son intégration dans visual studio :



A l’ouverture de Visual studio nous avons plusieurs possibilités pour créer un nouveau projet parmi celles-ci nous avons *« cloner ou extraire le code »*, qui nous permet d’accéder à un repositorie github.



Une fois dans le menu de cette option il est possible soit de connecter son compte github pour accéder et modifié directement ses propres repositories, ou bien de cloner en local un projet à l’aide de son URL. Nous devons ensuite spécifier le chemin dans lequel le projet sera cloné.

Une fois le projet ouvert nous avons sur la partie droite de l’écran ce menu permettant de gérer notre projet (Branche / Commit / Pull / Push / … ). En bas de cette fenêtre on peut voir plusieurs onglets :

* Team explorer : permet d’afficher les branches, les modifications, les pull requests etc…

Comme nous pouvons le voir sur les Screenshots ci-dessus, il y’a le menu principal de gestion qui nous redirige par exemple vers la visualisation de toutes les branches du projet. Ou encore vers les modifications éventuelles à commit pour le projet.

* Explorateur de solutions : qui permet de voir les fichiers présents dans notre dossier local en fonction de la branche dans laquelle nous nous situons. En cliquant sur l’un des fichiers que l’on souhaite modifier, celui-ci s’ouvre dans la partie gauche de notre fenêtre et nous sommes à présent en mesure de travailler dessus.

Pour finir on peut voire dans la barre violette en bas de notre écran dans quelle branche nous nous situons actuellment, le nom du repositorie, le nombre de modifications en attente de validation et le nombre de validation de faisant pas encore partie d’un push.