Info - 2A - TP4

Utilisation de GIT et tests unitaires

Objectif du TP : Savoir utiliser Git pour partager le code source et écrire des premiers tests unitaires

1. Récupération du squelette et lancement

Le squelette utilise PyInquirer. Il s'agit d'une librairie python qui facilite le développement d'applications dans la console.
Vous pouvez retrouver la documentation ici : https://github.com/CITGuru/PyInquirer/

Vous avez des exemples ici : https://github.com/CITGuru/PyInquirer/tree/master/examples

Récupérer le squelette sur moodle (*squelette-tp4.zip*) et ouvrez le projet sous VS Code.

Sous Visual code, ajoutez un nouvelle extension qui s'appelle **GitLens**. Elle va vous permettre d'utiliser Git depuis votre éditeur

Il s'agit d'un programme dont vous pouvez vous inspirer pour construire votre projet (vous pouvez même le démarrer à partir de ce squelette et le customiser ensuite). Les dao (pour l'accès à la base de données), les services (pour la logique de votre application), les vues (pour l'affichage) et les objects métiers sont séparés par répertoire.

Un répertoire *assets* contient les contenus textes à afficher.

Le répertoire *test* contient les tests de l'application.

Une fois que vous êtes dans le répertoire de votre projet avec le terminal, lancez la commande suivante pour installer les dépendances

```
pip install -r requirements.txt --user --proxy http://pxcache-02.ensai.fr:3128
```

Dans le fichier dao/**connection.py**, modifier MY_ID pour mettre votre identifiant.

Dans phpPgAdmin, lancez le script de création des tables (sql/**create_tables.sql**)

Ouvrez le fichier *test_compte.py*, il contient 3 tests : un pour valider que la création de compte s'effectue correctement et 2 autres pour vérifier que la méthode qui vérifie si le pseudo existe déjà fonctionne correctement.

Lancez les tests et vérifiez qu'ils s'exécutent correctement

```
python -m pytest
```

Vous devez voir s'afficher un message qui vous indique que les 3 tests se sont exécutés avec succès.

Maintenant lancez l'application. Elle va vous proposer de vous connecter ou de vous créer un compte. Choisissez *Me créer un compte* et créez vous un compte.

Avec phPgAdmin, constatez que votre compte a bien été créé dans la table **account**.

Maintenant regardez le code des fichiers **main.py**, **accueil.py** et **compte_creation.py**

La méthode *display_info* permet d'afficher des informations (on pourrait utiliser tabulate pour afficher un tableau comme dans le tp précédent) et le méthode *make_choice* permet de récupérer la saisie utilisateur et de retourner le prochain écran à afficher.

2. Création du compte Gitlab

Nous allons maintenant déposer notre projet sur Gitlab pour pouvoir le partager. Chaque étudiant doit se créer un compte sur gitlab : https://about.gitlab.com/ > **Register**

Si vous êtes chef de projet de votre groupe, créez un groupe (*Create Group*)

Dans *path* et *group name* mettez le nom de votre groupe (n'oubliez pas d'indiquer le numéro de votre groupe dans le nom) et laissez **private** dans le champ *Visibility Level*

Cliquez sur le bouton *Create Group*

Vous arrivez sur la page de votre groupe. A gauche dans le menu cliquez sur *Members*

Remplacez Guest par Developer et ajoutez les membres de votre groupe et votre encadrant.

Maintenant que le groupe est créé. Chaque binôme va pouvoir créer son propre projet pour la suite du TP.

Allez sur votre TP (dans le barre de menu en haute *Groups* > *Your Groups*

Cliquez sur votre groupe puis sur le bouton vert ***New project***

Donnez un nom à votre projet (par exemple votre identifiant).

Vous avez quelques paragraphes intitulés *Commande Line Instruction* et notamment *Existing Folder*, c'est ce que nous allons faire dans la suite du TP.

3. Envoi du projet sur Gitlab

Lancez **Git bash** (Menu démarrer > tapez *git bash* dans la barre de recherche).

La première chose à faire est de configurer le proxy

```
git config --global http.proxy http://pxcache-02.ensai.fr:3128
```

Puis de configurer vos informations personnelles

```
git config --global user.email <VOTRE EMAIL>
git config --global user.email « PRENOM NOM »
```

En remplaçant <VOTRE EMAIL> par votre email et PRENOM NOM par vos prénoms et noms.

Cette config est conservée vous n'avez à la faire qu'une fois sur les machines de l'ENSAI

Puis de générer votre clé ssh

```
ssh-keygen -t rsa b 2048
```

Tapez entrée pour chaque question que l'on vous pose.

Votre clé SSH va vous permettre de vous identifier lors des échanges avec gitlab.

Sous Gitlab, allez dans vos réglages : https://gitlab.com/profile

Dans le menu à gauche allez dans *SSH Keys*

Vous avez un champ text sous **Key**, vous allez mettre le contenu de votre clé. Pour l'afficher, avec Visual Code, ouvrez le fichier qui est sous *c://Utilisateurs/votre identifiant/.ssh/id_rsa* (il y en a 2, il faut prendre celui qui est de type Document Microsoft …)

Une fois que vous l'avez copié, cliquez sur **Add Key**

La clé ssh est conservée. Vous n'aurez donc à la configurer qu'une fois à l'ensai. Par contre si vous travaillez avec un ordinateur perso, il faudra créer une clé ssh dessus (et l'ajouter sur gitlab)

Maintenant avec git bash, placez vous dans le répertoire du squelette que vous avez récupéré.

Tapez

```
git init
```

Vous avez initialisé un dépôt git sur votre machine.

Tapez

```
git status
```

Vous voyez l'ensemble des fichiers qui ont été modifié

Le souci est que vous retrouvé le répertoire *.pytest_cache* et les répertoires *__pycache__* qui ont été généré par python mais qui ne sont pas du code source à proprement parler.

Dans le répertoire, ajouter le fichier *.gitignore* qui va contenir les répertoires / fichiers à ignorer.

Ajoutez à l'intérieur :

.pytest_cache
__pycache___

Attention à bien passer la ligne pour chaque répertoire

Refaites à nouveau

```
git status
```

Les répertoires ont disparus

Nous allons maintenant créé notre premier commit. Pour le moment aucun fichier n'a été sélectionné (staged) pour en faire partie. Nous allons tous les ajouter.

```
git add -A
```

Puis refaites à nouveau

```
git status
```

Les fichiers qui apparaissaient auparavant en rouge sont maintenant affichés à vert. Ils sont candidats pour le prochain commit.

Créer votre commit

```
git commit -m 'mon premier commit'
```

Vous pouvez remplacer le message *mon premier commit* par celui que vous souhaitez. Le message doit aider à comprendre à quoi correspond les modifications qu'il embarque.

Si vous retournez sur votre projet sous gitlab et que vous rafraichissez la page, vos fichiers n'apparaissent toujours pas.

C'est normal, vous avez fait un commit sur votre machine mais vous n'avez pas synchronisé votre dépôt local avec le dépôt distant sous gitlab.

Nous allons maintenant synchroniser notre dépôt local avec le dépôt distant.

Dans gitlab, sur la page de votre projet, vous avez une url sous le nom de votre projet qui doit commencer par git@gitlab.com (si ce n'est pas le cas, remplacez https par ssh dans la sélection juste devant)

Puis dans le répertoire de votre projet, tapez

```
git remote add origin <MON_URL>
```

En remplaçant **<MON_URL>** par l'url que vous avez récupéré.

Nous venons d'ajouter un dépôt distant (appelé origin). Nous allons maintenant pousser notre commit local (qui est sur la branche master) sur le dépôt distant

```
git push -u origin master
```

Rafraichissez la page gitlab. Vos fichiers apparaissent maintenant.

Pour les commits suivants, il ne sera plus nécessaire de mettre l'option -u qui permet de lier la branche locale avec la branche distante

4. Clone du projet

Dans un autre répertoire nous allons faire un copie de notre projet. Créer un nouveau répertoire, placez vous à l'intérieur et tapez

```
git clone <MON_URL>
```

En remplaçant **<MON_URL>** par l'url que vous avez récupéré précédemment.

Maintenant dans une nouvelle fenêtre VS Code, ouvrez le projet qui a été récupéré dans votre nouveau répertoire.

Nous avons maintenant 2 fenêtre de VS Code : une avec le projet initial que nous avons poussé sur gitlab et une seconde avec le même projet dont nous avons fait un clone.

Ce que nous allons faire c'est modifier un projet, pousser les modifications dans gitlab et les récupérer ensuite dans le clone.

En vous inspirant du code permettant de créer son compte, développer l'authentification : modifier *compte_authentification.py* pour vérifier les identifiants (pseudonyme et mot de passe) saisis et vérifiez qu'ils correspondent bien à un utilisateur enregistré.

Vous devrez créer une nouvelle méthode dans *compte_service.py*. Ajoutez 2 tests : un pour le cas ou le compte est connu et un autre pour le cas ou le compte est inconnu dans *test_compte.py*

Vérifiez que votre test fonctionne

```
python –m pytest
```

Vérifier que votre code fonctionne en lançant le programme et choisissant *Se connecter*.

Quand tout fonctionne correctement, ajoutez vos fichiers, faites votre commit et pousser vos modifications sur *Gitlab*.

Maintenant que vous avez modifié le dépôt distant, nous allons récupéré les modification dans le second projet (le clone).

Faites

```
git fetch
```

Cette commande permet de consulter le dépôt distant et de savoir s'il y a des commits à récupérer

Faites ensuite

```
git status
```

Vous devez avoir un message qui vous indique qu'il y a un commit sur le dépôt distant que vous n'avez pas récupéré

Pour récupérer le commit, faites

```
git pull
```

5. Gestion d'un conflit

Nous allons maintenant voir ce qu'il se passe lorsque 2 utilisateurs modifient la même ligne de code. Il y a alors un conflit. Vous devez régler ce conflit.

Dans un des 2 dépôt, modifiez le fichier *accueil.py* : remplacez le message *Que souhaitez vous faire ?* par *Veuillez choisir ?*

Envoyez vos modification sur gitlab (en utiliser *git add*, *git commit* et *git push*)

Dans le second dépôt, modifiez aussi le fichier *accueil.py* : remplacez le message *Que souhaitez vous faire ?* par *Se connecter ou créer un compte ?*

Créez un commit (*git add* et *git commit*)

Essayez de pousser vos modification (*git push*)

Un message d'erreur vous indique que vous n'êtes pas à jour donc vous ne pouvez pas envoyer vos modifications.

Récupérez les modifications (*git fetch* puis *git pull*)

Un message vous indique que vous avez un conflit et qu'il va falloir le régler

Sous Visual studio, cliquez sur l'icône 

Vous voyez le fichier qui est en conflit

Cliquez dessus pour l'ouvrir, vous voyez le conflit qui s'affiche

<<<<< HEAD
Votre modification (current change)
==========
La modification du dépôt distant (incoming change)
>>>>>>>

Au dessus vous avez des liens :
- *Accept current change* pour conserver votre modification
- *Accept incoming change* pour abandonner votre modification et garder seulement
- *Accept both change* pour garder les 2 modifications
- *Compare change* pour voir cote à cote les 2 modifications (plus pratique pour faire le comparaison)

Nous allons conserver que notre dernière modification. Cliquez sur *current change*.

Puis commitez et poussez vos changements (*git add*, *commit* et *push*)

6. Initialiser votre projet

Mettez vous d'accord avec vos équipiers pour choisir un nom de projet et créez le sur gitlab pour pouvoir démarrer les développements de votre application. Vérifiez que tout le monde a bien accès au projet et peut le modifier.