# Bonnes pratiques- Datalake

## Arborescence :

- Process_zone : Zone relatives à Apache Airflow et Apache Spark.
- Service_zone : Zone contenant les applications frontend (GUI ReactJS) et backend (Python Flask).
- Raw-data-zone : Zone relative à Openstack Swift afin de gérer les données brutes du projet.

## Qualité de code

Le code doit nécessairement être clair afin que le travail collaboratif soit efficace et que le travail puisse être bien repris derrière.

Le fait, également, que le projet UPS soit un projet de mécénat signifie qu'il doit y avoir une certaine rigueur dans le travail et l'organisation.

### Les commentaires

Il doit y avoir des commentaires pour :

- chaque classe / fonction,
- chaque bout de code conséquent (comportant un certain nombre de lignes).

Parmi les choses les plus courantes dans les projets informatiques est le travail collaboratif, en codant, il faut toujours penser aux autres personnes qui liront votre code, et également penser au fait que le code soit réutilisable.

### Code clair

Les noms des fonctions et des variables doivent être clairs et concis. Egalement, par rapport à la clarté de code, il faut éviter la redondance de code, si vous pensez qu'il va y en avoir, placez des fonctions ou créez des classes si vous observez des comportements similaires ou que vous pensez qu'il y a une possibilité d'encapsulation (si le passage à la POO [1]est pertinente).

## Messages de commits / Merge Requests

Il faut créer une issue sur GitLab pour chaque branche / lot de fonctionnalités. Chaque issue créée est identifiée par un numéro de l'issue [2] qu'il faudra mettre dans chaque commit / nom de Merge Request afin de toujours respecter des règles de qualité de code et de gestion de version.

Egalement, les commits doivent être écrits de manière **concise** (brève afin que le message ne soit pas trop long) et **claire**.

---

[1] POO : Programmation Orientée Objet, un paradigme de programmation employé historiquement pour gérer le code par des objets, contrairement au procédural.

[2] Les issues peuvent être utilisés pour assurer le suivi des bogues, des améliorations ou d'autres demandes de projets.

# Bonnes pratiques- Datalake

## Tests unitaires

L'ensemble des tests unitaires sont effectués avec l'outil « pytest » automatiquement installé au moment de la compilation de l'image Docker car installé avec les dépendances Python.

Ils sont dans le répertoire « tests » dans le dossier « flask/neocampus » côté backend. Chaque fichier de tests correspond à un routeur et chaque méthode commence par « test_ », cela pemet à Pytest de détecter les méthodes de test.

La convention de nommage suivante est ensuite adoptée sur toutes les autres méthodes : « test_ »_ »nom_methode_à_tester »_ »nature_du_test ».

Exemple dans le fichier de test du routeur InfluxDB :

```
def test_get_all_buckets_if_key_is_present(client):
```

(la méthode testée est « get_all_buckets »permettant de récupérer l'ensemble des buckets de InfluxDB et la nature du test est de savoir si la clé «buckets » dans le résultat de la route est bien présente.

Pour exécuter les tests, vous pouvez exécuter l'une des 2 commandes suivantes :

Si vous restez sur la machine sans entrer dans le container :

```
docker exec -it flask pytest
```

Si vous entrez dans le container :

```
root@fabea364eb79:/app# pytest
```

Le résultat dans la console sera le même, dont celui actuel est ci-dessous.

```
======== 3 failed, 6 passed, 6 warnings in 19.40s ===
```

Le détail de chaque test est affiché au-dessus de ce résultat.