Technique de test

1-hiérarchie du projet

A- Dossier triangulator

Ce dossier contient tout le code du micro-service de triangulation. Il regroupe l'API HTTP, la logique métier de calcul des triangles et la gestion des formats binaires. Il correspond au composant fonctionnel principal du projet.

Triangulator/App.py

Ce fichier définit l'API HTTP du Triangulator. Il expose l'endpoint `/triangulation/<pointSetId>` et orchestre tout le workflow : validation de l'ID, récupération du PointSet, décodage, triangulation, encodage du résultat et gestion des erreurs HTTP.

triangulator/tringulator.py

⇒Ce fichier contient la logique métier du Triangulator. Il gère l'encodage et le décodage des formats binaires (PointSet et Triangles) et implémente une triangulation simple à partir d'une liste de points, en traitant les cas limites.

triangulator/clients.py

Ce fichier définit l'interface de communication avec le PointSetManager. La classe `PointSetManagerClient` permet de récupérer un ensemble de points à partir d'un identifiant, et est volontairement mockée dans les tests afin de tester le Triangulator de manière isolée.

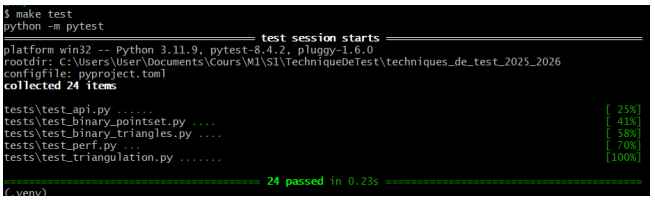
B-Dossier test

Le dossier `tests` sert à vérifier que le micro-service Triangulator se comporte correctement dans toutes les situations prévues. Les tests réalisés sont :

- **Tests d'API** : vérification du comportement de l'endpoint `/triangulation/<id>` (UUID invalide, PointSet introuvable, PointSetManager indisponible, format binaire invalide, échec de triangulation, succès avec réponse binaire).
- **Tests d'encodage/décodage des PointSet** : validation du format binaire des points (cas normal, ensemble vide, valeurs extrêmes, buffer invalide).
- **Tests d'encodage/décodage des Triangles** : validation du format binaire des triangles, gestion des indices hors limites et des buffers incohérents.
- **Tests de triangulation** : calcul des triangles pour des cas simples et limites (0, 1, 2, 3 points, points colinéaires, carré, points dupliqués).
- **Tests de performance** : mesure du temps de triangulation pour des ensembles de points de tailles croissantes (100, 1000, 5000 points).

2-Résultats des tests

- **make test**



⇒ L'ensemble des tests du projet a été exécuté à l'aide de la commande `make test`. Les 24 tests définis ont tous passé avec succès, ce qui valide le bon fonctionnement du micro-service Triangulator, aussi bien au niveau de la logique interne que de l'API HTTP. Les tests couvrent les formats binaires, la triangulation, la gestion des erreurs et les performances, conformément aux objectifs du TP.

- **make unit_test: tester les tests unitaires**

```
$ make unit_test
python -m pytest -m "not perf"
============================= test session starts =============================
platform win32 -- Python 3.11.9, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\User\Documents\Cours\M1\S1\TechniqueDeTest\techniques_de_test_2025_2026
configfile: pyproject.toml
collected 24 items / 3 deselected / 21 selected

tests\test_api.py ......                                                 [ 28%]
tests\test_binary_pointset.py ....                                       [ 47%]
tests\test_binary_triangles.py ....                                      [ 66%]
tests\test_triangulation.py .......                                      [100%]

================================ 21 passed, 3 deselected in 0.21s =================================
(.venv)
User@DESKTOP-HIC05TP MINGW64 ~/Documents/Cours/M1/S1/TechniqueDeTest/techniques_de_test_2025_2026/TP (m
ain)
```

⇒Sur les 24 tests définis, 21 ont été exécutés et tous ont passé avec succès, confirmant le bon fonctionnement du Triangulator hors aspects de performance.

- `make perf_test` tester les tests performances

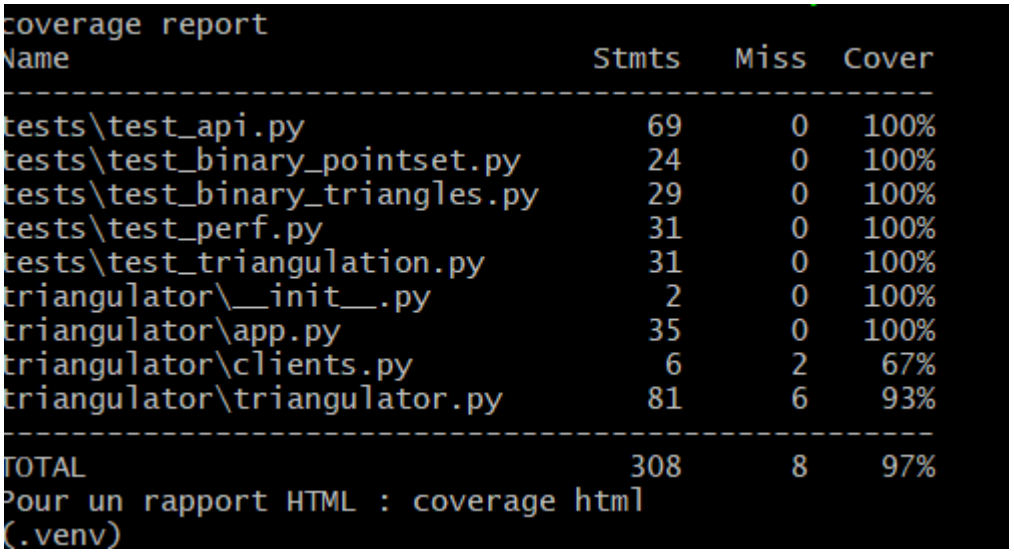
```
$ make perf_test
python -m pytest -m perf
============================= test session starts =============================
platform win32 -- Python 3.11.9, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\User\Documents\Cours\M1\S1\TechniqueDeTest\techniques_de_test_2025_2026
configfile: pyproject.toml
collected 24 items / 21 deselected / 3 selected

tests\test_perf.py ...                                                   [100%]

================================ 3 passed, 21 deselected in 0.20s =================================
(.venv)
User@DESKTOP-HIC05TP MINGW64 ~/Documents/Cours/M1/S1/TechniqueDeTest/techniques_de_test_2025_2026/TP (m
ain)
```

⇒ Les tests de performance ont été exécutés séparément à l'aide de la commande `make perf_test`. Trois tests mesurant le temps de triangulation sur des ensembles de points de tailles croissantes (100,1000,5000)ont été lancés et ont tous passé avec succès, confirmant que les performances du Triangulator sont acceptables dans les conditions testées.

- **make coverage couverture du code**



⇒ La couverture de code a été mesurée à l'aide de l'outil `coverage`. Le projet atteint une couverture globale de 97 %, ce qui montre que la quasi-totalité du code applicatif est exercée par les tests. Les parties non couvertes correspondent principalement à du code volontairement non implémenté (client PointSetManager mocké) ou à des chemins d'erreur difficiles à déclencher sans tests artificiels.

- **make lint la qualité du code**

La documentation a été régénérée après correction des docstrings en supprimant l'ancienne version puis en exécutant `make doc,` un dossier doc s'est crée en se basant sur la docstring générée.

- **make lint**

Les erreurs restantes signalées par Ruff concernent exclusivement des règles de style (docstrings, longueur de ligne, précision des exceptions levées). Elles n'affectent ni la logique métier ni le comportement des tests, qui passent tous avec succès. Ces règles ont été progressivement corrigées afin de garantir un code conforme aux standards de qualité imposés.(j'ai pas eu le temps de tout finir)