Rapport - Projet Triangulator

Master 1 - Technique de Test

Caroline HULOT

17 décembre 2025

1 Introduction

Le projet **Triangulator** consiste en la réalisation d'un micro-service de calcul géométrique. Il y a eu la mise en place d'une approche *Test-First*.

Fonctionnement du Triangulator

1. **Réception :** Récupère un ensemble de points (`PointSet`) identifié par un UUID.
2. **Traitement :** Effectue une triangulation.
3. **Réponse :** Retourne le résultat (`Triangles`) dans un format binaire.

2 Architecture du projet

Structure du répertoire

```
TP/
|-- Triangulator/
|    |-- app.py              # API Flask
|    |-- triangulator.py     # Algorithme de triangultaion
|    |-- binary_format.py    # serialisation
|
|-- tests/
|    |-- TestsUnitaires.py
|    |-- TestsRoutes.py
|    |-- TestsPerformance.py
|    |-- TestsRobustesse.py
|    |-- TestsSecurite.py
|    |-- TestsQualite.py
|    |-- mocks.py
```

3 Réalisation

3.1 Implémentation du Triangulator (`triangulator.py`)

L'algorithme de triangulation vient d'internet car le but premier du projet étais les tests. Celui que j'ai choisis est performant et marche parfaitement.

4 Les Test

Au total, **78 tests** verifie le fonctionnement du Triangulator.

4.1 1. Tests Unitaires (24 tests)

Validation de la logique pure, répartie entre `TestsUnitaires.py` (22 tests) et `TestsAutres.py` (2 tests).

— **Algorithmique :** Vérification des cas standards, (1 triangle, carré,...) et des cas dégénérés (points alignés → 0 triangle, points dupliqués,...)

— **Binaire :** V.érification de la conversion (`PointSet` ↔ Binaire).
— **Limites :** Gestion des valeurs `NaN`, `Inf` et précision des coordonnées.

4.2 2. Tests API et Routes (11 tests)

Validation des routes HTTP dans `TestsRoutes.py`.
— Vérification des codes de retour (200, 201, 400, 404).
— Validation des types de contenu et des structures de réponse.

4.3 3. Robustesse et Sécurité (19 tests)

Tests offensifs pour verifier la résilience du système.
— **Robustesse (14 tests) :** Résistance aux données corrompues, timeouts et pannes simulées.
— **Sécurité (5 tests) :** Protection contre les injections (SQL, Shell), DoS et ID malveillants.

4.4 4. Intégration et Système (16 tests)

Validation des interactions et du flux complet (`TestsIntegrations.py`, `TestsSysteme.py`, `test-triangultor-interactive.py`).

4.5 5. Performance (4 tests)

Validation des temps de réponse et benchmarks.

NB Points	Temps Moyen	NB Triangles
100	$\sim$ 0.0007 s	98
1 000	$\sim$ 0.0613 s	998
5 000	$\sim$ 1.8589 s	4 998
10 000	$\sim$ 7.2022 s	9 998
50 000	$\sim$ 218.6335 s	49 998
100 000	$\sim$ 957.3221 s	99 998

TABLE 1 – Benchmark de la triangulation

5 Résultats

5.1 Tableau de Synthèse

Composant	Statut	Détails
Tests	100% Passés	**78 tests** exécutés avec succès.
Qualité	Conforme	`ruff` (linting) et `coverage` (93%).
Documentation	Générée	HTML disponible via `make doc`.

5.2 Analyse des Écarts (Implémentation Test vs Réalité)

Il y a une légère différence entre les tests que je voulais faire et les tests finaux.

Section	Voulu	Exécuté	Justification de l'écart
Tests Unitaires	31	24	Regroupement de cas.
Tests API	16	11	Consolidation des tests de validation de payload.
Tests Robustesse	15	14	Fusion des tests sur les exceptions internes.
Tests Sécurité	5	5	Pareil.
Tests Intégration	9	9	Pareil.
Tests Système	7	7	Pareil.
Performance	4	4	Pareil.
Qualité	4	4	Pareil.
TOTAL	**91 (théorique)**	**78 (réel)**	Optimisation.

TABLE 2 – Comparatif Planification vs Exécution

6 Comparaison : Plan Initial vs l'Implémentation

Le tableau montre les zones où l'implémentation a changé par rapport au plan de depart.

Axe	Plan Initial (Théorique)	Rapport d'implémentaion	Réalité (Implémenté)
Nombres	50 tests estimés	81 tests annoncés	**78 tests exécutés**
Unitaires	Algo de base	31 tests (avec doublons logiques)	**24 tests optimisés** (Regroupement des cas limites)
Performance	"Mesurer temps"	4 tests	**4 tests + Benchmark** séparé
Sécurité	"ID trafiqué"	5 tests (SQL, DoS)	**5 tests**
API / Routes	Codes HTTP classiques	16 tests	**16 tests**
Couverture	Obj. > 90%	100%	**93%** de couverture de code réelle

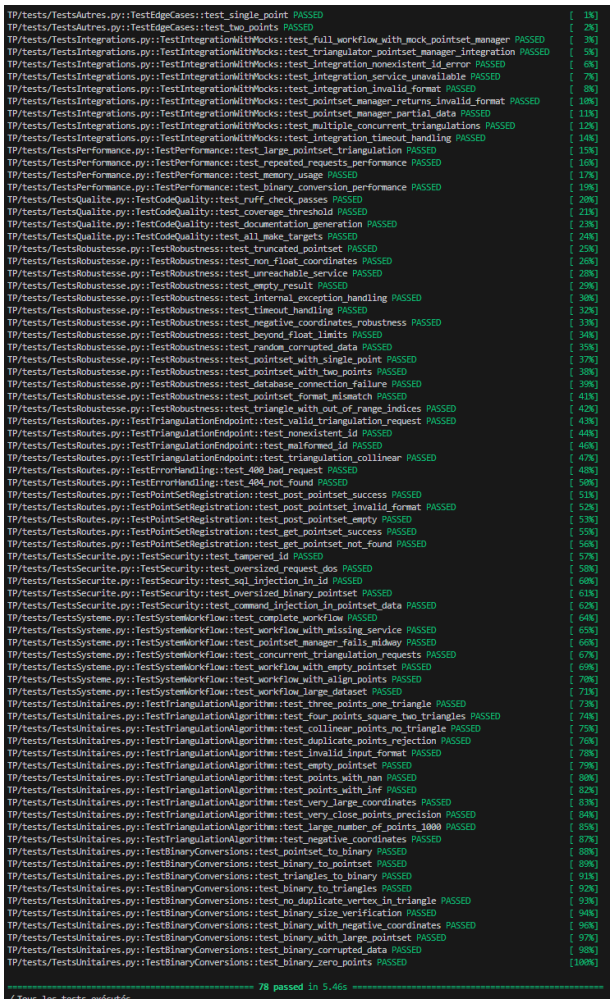
TABLE 3 – Évolution des test

7 Automatisation et Outillage (Makefile)

On peut tout faire avec le`Makefile`.

7.1 Commandes Principales

— `make test` : Lance tout les tests (Unitaires, Intégration, Performance,...).



— `make unit_test` : Exécute uniquement les tests funitaires.
— `make perf_test` : Isole les tests de performance et de charge.
— `make benchmark` : Lance le script de benchmark dédié (`benchmark_triangulation.py`) pour mesurer les temps d'exécution.
— `make lint` applique les règles de style via `ruff` (avec auto-fix).
— `make coverage` génère le rapport HTML complet dans : `TP/htmlcov/index.html`.

```
============================================================ 78 passed in 3.49s ============================================
Name                                Stmts   Miss  Cover   Missing
-------------------------------------------------------------------
TP/Triangulator/app.py                 72     15    79%   56-58, 80, 82, 111, 149-150, 185, 203-204, 231, 243-244, 250
TP/Triangulator/binary_format.py       58      7    88%   97, 153, 195, 201, 209, 219, 222
TP/Triangulator/triangulator.py        70      9    87%   84, 136, 198-213
TP/__init__.py                          0      0   100%
TP/tests/TestsAutres.py                12      0   100%
TP/tests/TestsIntegrations.py          76      0   100%
TP/tests/TestsPerformance.py           47      0   100%
TP/tests/TestsQualite.py               28      0   100%
TP/tests/TestsRobustesse.py            84      1    99%   98
TP/tests/TestsRoutes.py                56      0   100%
TP/tests/TestsSecurite.py              25      0   100%
TP/tests/TestsSysteme.py               66      0   100%
TP/tests/TestsUnitaires.py            168      6    96%   53-56, 137-139
TP/tests/__init__.py                    0      0   100%
TP/tests/conftest.py                   11      0   100%
TP/tests/mocks.py                      82     21    74%   61, 79, 88, 99-100, 113-119, 124-129, 134-136, 141-143, 148, 153, 158
-------------------------------------------------------------------
TOTAL                                 855     59    93%
Wrote HTML report to TP/htmlcov/index.html
✓ Rapport de coverage généré dans TP/htmlcov/index.html
```

FIGURE 1 – make coverage
labelfig :placeholder

— `make doc` génère la documentation API dans :
`TP/docs`.
— `make clean` : Supprime tous les artefacts de build, caches python, et rapports générés pour repartir sur une base saine.