Plan de tests pour le microservice Triangulator

AMOROS GAËL

Contents

1. Portée des tests

Avant de détailler les différents types de tests, il est important de définir la portée des tests pour le microservice Triangulator.

Composants concernés par les tests :

- Triangulator (API)
- PointSetManager (API)
- Client (Utilisateur)

Fonctions couvertes

- Récupération d'un PointSet depuis PointSetManager ;
- Calcul de la triangulation à partir du PointSet ;
- Génération de la réponse binaire Triangles ;
- Gestion des erreurs (400, 404, 500, 503).

2. Catégories de tests

2.1 Tests unitaires

Objectif : vérifier le comportement isolé de chaque fonction.

a. Tests sur la conversion binaire

- Lecture/écriture d'un PointSet depuis/vers son format binaire :
- Cas nominal : 3 points -> N=3, coordonnées cohérentes.
- Cas limite : N=0.

- Cas invalide : taille binaire incohérente, données tronquées.
- Lecture/écriture d'un Triangles binaire :
- Cas nominal : 4 sommets / 2 triangles.
- Cas invalide : index hors bornes, structure malformée.

b. Tests sur la logique de triangulation

- Vérification que la triangulation produit le bon nombre de triangles.
- Vérification que les triangles couvrent bien le convex hull.
- Gestion des cas particuliers :
- Points colinéaires ;
- Points dupliqués ;
- Jeu de points vide ou avec un seul point.

c. Tests sur la gestion des erreurs internes

- Exceptions levées pour input invalide.
- Logs et messages d'erreur conformes au schéma Error.

2.2 Tests d'intégration

Objectif : vérifier l'interaction entre le Triangulator et le PointSetManager.

a. Appel réussi

- Mock du PointSetManager retournant un PointSet valide.
- Vérification :
- Requête HTTP correcte ;
- Réponse 200 avec contenu binaire Triangles valide.

b. Erreurs propagées depuis le PointSetManager

- Cas 404 -> le Triangulator renvoie 404.
- Cas 503 -> le Triangulator renvoie 503.
- Cas données corrompues -> le Triangulator renvoie 500.

c. Erreurs côté Triangulator

- Format d'ID invalide (bad_uuid) -> 400.
- Exception interne simulée -> 500.

2.3 Tests API (end-to-end)

Objectif : valider la conformité à la spécification OpenAPI (triangulator.yml).

Vérifications :

- Routes, paramètres et codes retour conformes.
- En-têtes et types MIME (application/octet-stream, application/json).
- Respect du format binaire attendu :
- Longueur cohérente ;
- Types (float, unsigned long) corrects ;
- Endianness cohérente (little endian par défaut).

Outils :

- flask.testing pour lancer le serveur en mode test.
- openapi-core ou schemathesis pour valider la conformité.

2.4 Tests de performance

Objectif : mesurer la scalabilité et le temps de réponse du service.

Les scénarios suivants seront testés :

- Nombre de points croissant : 10, 100, 1 000, 10 000.
- Mesure du temps de conversion binaire.
- Mesure du temps de triangulation.
- Test de charge : 100 requêtes simultanées via pytest-benchmark.

Les critères d'acceptation seront :

- Temps moyen par triangulation ;
- Temps max autorisé (à définir empiriquement) ;
- Aucune dégradation fonctionnelle (erreurs 500, timeouts).

Ces tests seront marqués avec @pytest.mark.performance et exclus du make unit_test.

2.5 Tests de robustesse et d'erreurs

Les cas d'erreurs suivants seront simulés :

- Requête sans PointSetID -> 400.
- PointSetID non-UUID -> 400.
- Serveur PointSetManager injoignable -> 503.
- PointSet vide -> 500 (ou erreur gérée explicitement).
- Corruption du flux binaire (par ex. taille incohérente) -> 400.

L'objectif est de garantir que les erreurs sont correctement traitées et que le service ne plante jamais sans renvoyer une erreur formelle (Error JSON).

3. Organisation des tests

Pour organiser les tests, la structure de répertoires suivante sera utilisée :

```
/tests

   unit/
     test_binary_encoding.py
     test_triangulation_logic.py
     test_error_handling.py

   integration/
     test_triangulator_pointsetmanager.py

   api/
     test_openapi_conformance.py
     test_end_to_end.py

   performance/
   test_performance_triangulation.py
```