# Processus

Comme graphhopper est un logiciel extensif, c’était difficile d’identifier les tests unitaires à ajouter au logiciel en manuellement fouillent dans le code. En apprenant à propos des tests par mutation pour nos présentations, on a remarqué qu’il était une façon plus efficace d’identifier les bouts de code non testé où pas testé assez robustement. Donc on a exécuté des tests par mutation et en utilisant les résultats, a écrit des tests unitaires qui amélioreraient la robustesse de la suite des tests.

# Justification des tests

La classe CustomModelAreasDeserializer joue un rôle crucial dans le projet GraphHopper, assurant plusieurs fonctions importantes telles que la désérialisation personnalisée, la gestion des formats GeoJSON, et la validation des données. Ces fonctionnalités sont essentielles pour le traitement efficace et fiable des informations géographiques dans le système de routage.

Lors de notre analyse, nous avons identifié deux mutants survivants, révélant une lacune dans notre couverture de tests. En effet, il n'existait pas de classe de test spécifique pour CustomModelAreasDeserializer. Pour remédier à cette situation et renforcer la robustesse de notre code, nous avons élaboré une série de tests couvrant divers scénarios :

### Test 1 : testValidFeatureCollection()

Justification:

Ce test est fondamental car Il assure que le processus de désérialisation fonctionne correctement dans des conditions normales. Sans ce test, nous ne pourrions pas garantir que le désérialiseur traite correctement même les cas les plus basiques.

### Test 2 : testInvalidType()

Justification:

Crucial pour l'intégrité des données car il vérifie que le désérialiseur rejette activement les données mal formatées ou incorrectes. Sans cette vérification, le système pourrait accepter et traiter des données invalides.

### Test 3 : testMissingType()

Justification:

Ce test renforce la robustesse en s’assurant que le programmme reagit correctement à du l'absence du champ "type”

### Test 4 : testNullNode()

Justification:

Bien que non directement lié au mutant ci-haut, ce test de robustesse du système est important pour prévenir les crashs potentiels

### Test 5 : testEmptyNode()

Justification:

Ce test est nécessaire pour renforcer la robustesse du système et prévient les comportements inattendus qui pourraient survenir lors du traitement de données réelles et variées.

### Test 6 : testFeatureWithoutId()

Justification:

Ce test est crucial pour la vérification de l'intégrité des données dans le processus de désérialisation, en particulier en ce qui concerne le traitement des identifiants des fonctionnalités géographiques.

### Test 7 : testDuplicateFeatureIds()

Justification:

Ce test est crucial pour la vérification de l'unicité des données dans le processus de désérialisation, en particulier en ce qui concerne le traitement des identifiants des fonctionnalités géographiques.

### Test 8 : void testCalcBBox2D()

Justification:

Ce test est fondamental pour prevenir les erreurs potentiellement graves dans les calculs de distance et garantit que la méthode n'omet aucun point extrême.

### Test 9 : testAddPathDetails()

Justification:

Ce test est essentiel pour garantir l'intégrité et la cohérence des détails de chemin (path details) dans GraphHopper.

### Test 10 : testAddAreasWithInvalidId()

Justification :

Ce test est crucial pour garantir l'intégrité et la validité des identifiants des zones (areas) dans le modèle personnalisé de GraphHopper. Il joue un rôle fondamental dans la prévention d'erreurs potentiellement graves liées à l'utilisation d'identifiants invalides.

### Test 11 : testEmptyCamelCaseToEmptyUnderscore()

Justification :

Ce test est fondamental face à une entrée vide, ce qui constitue un cas limite important.

### Test 12 : testEmptyUnderscoreToEmptyCamelCase()

Justification :

En tandem avec le test précédent, il assure une symétrie dans le traitement des chaînes vides, ce qui est crucial pour maintenir la cohérence dans les opérations de conversion bidirectionnelles.

Svp noter que les intentions de chaque test sont incluses dans les commentaires.

# Où peut se trouver les tests

Les tests 1 à 7 peuvent se trouver à partir du chemin suivant :

A black screen with white text

Description automatically generated

Le lien vers la classe CustomModelAreasDeserializerTest.java sur notre repo GitHub peut se trouver [ici](https://github.com/h-mbl/graphhopper/blob/4c82170dfe7923a6b040e7d4369752f647482a5c/web-api/src/test/java/com/graphhopper/jackson/CustomModelAreasDeserializerTest.java).

Les tests 8 et 9 peuvent se trouver à partir du chemin suivant :

A black background with white text

Description automatically generatedLe lien vers la classe ResponsePathTest.java sur notre repo GitHub peut se trouver [ici](https://github.com/h-mbl/graphhopper/blob/af6b0895d6cb17e893326dd6614b5bf547b7325d/web-api/src/test/java/com/graphhopper/ResponsePathTest.java).

Le test 10 peut se trouver à partir du chemin suivant :

A black screen with white text

Description automatically generated

Le lien vers la classe CustomModelTest.java sur notre repo GitHub peut se trouver [ici](https://github.com/h-mbl/graphhopper/blob/af6b0895d6cb17e893326dd6614b5bf547b7325d/web-api/src/test/java/com/graphhopper/util/CustomModelTest.java).

Les tests 11 et 12 peuvent se trouver à partir du chemin suivant :

A black screen with white text

Description automatically generated

Le lien vers la classe sur notre repo GitHub peut se trouver [ici](https://github.com/h-mbl/graphhopper/blob/3470b53653855a49cdf097b9e93b6d201cc0ee72/web-api/src/test/java/com/graphhopper/util/HelperTest.java).