# Travail Pratique : Tests sur une Application déjà existante

© Pierre-Antoine Guillaume

2024-10-31

## 1 Introduction

Ce travail pratique est conçu pour évaluer vos compétences en tests logiciels sur une application Flask de gestion de tâches.

Vous y appliquerez des concepts de couverture, de mock, de tests boîte noire et boîte blanche, ainsi que des tests de bout-en-bout et d'intégration.

## 2 Rendu

Le format attendu du rendu est un fichier ainsi nommé :
`mdt-<m2|m1>-data<sci|eng>-<nom>-<prenom>.<pdf|ipynb>`
Il sera rendu en fonction des possibilités :

1. soit sur moodle
2. soit via mail à pierreantoine.guillaume@ynov.com en **pièce jointe**
3. soit via teams

À la fin de ce TP, vous devez fournir :
— Le code de tous les tests écrits.
— Une explication précisant votre stratégie de tests et vos observations.

### 2.1 Recommandations

— créez votre document de rendu (correctement nommé) dès le début
— nommez correctement vos tests : i.e. def test_score_spare_adds_score_once_more()
— si le nom du test ne se suffit pas, ajoutez un commentaire pour mieux exprimer votre intention
— composez le document au fur et à mesure que vous faites votre travail

### 2.2 Retards

J'attends de vous du professionnalisme : les retards ne sont pas éliminatoires, je préfère un travail rendu tardivement plutôt que pas du tout. Un rendu en retard entraînera cependant une pénalité, mais il est crucial de me prévenir avant l'échéance. Une bonne communication en temps et en heures pourra réduire la pénalité.

## 3 Objectifs

Les objectifs de ce TP sont :
— Comprendre et tester les fonctionnalités d'une application Flask avec base de données SQLite.
— Mettre en pratique les techniques de tests boîte noire et boîte blanche.
— Utiliser les mocks pour isoler des dépendances.
— Analyser la couverture de code et identifier les segments non couverts.
— Écrire des tests d'intégration et de bout-en-bout (E2E) pour valider l'ensemble du flux utilisateur.

# 4 L'application : Code et Fonctionnalités

L'application fournie permet d'ajouter, compléter, nettoyer et consulter des tâches, tout en calculant un score pour chaque tâche complétée.

Vous trouverez, dans le repository git du cours, le code de l'application à tester :

Listing 1 – Cloner le dépot

```
git clone https://github.com/PierreAntoineGuillaume/methodologie-du-test
```

# 5 Consignes de Travail

Les consignes du TP sont divisées en étapes pour vous guider dans les différents aspects des tests.

Tout le long du TP, n'oubliez pas les enjeux du tests, et pensez à faire des tests aux valeurs limites.

## 5.1 Étape 1 : Tests Fonctionnels et Boîte Noire

**Objectif** : Vérifier le bon fonctionnement des routes principales (ajout, complétion, consultation, nettoyage) sans regarder l'implémentation interne.

- Écrivez des tests pour vérifier qu'une tâche peut être ajoutée avec les champs requis (titre, priorité, difficulté, date d'échéance).
- Testez la complétion d'une tâche et assurez-vous que le score soit correctement calculé et mis à jour.
- Écrivez des cas de tests pour la route de nettoyage afin de vérifier que les tâches obsolètes ou complétées sont bien supprimées.

## 5.2 Étape 2 : Tests Boîte Blanche et Couverture de Code

**Objectif** : Analyser et augmenter la couverture de code avec des tests boîte blanche pour vérifier l'exactitude de chaque fonction.

- Exécutez les tests en générant un rapport de couverture avec `pytest-cov`.
- Identifiez les lignes non couvertes et écrivez des tests spécifiques pour couvrir les cas d'usage complexes (par exemple, la fonction `worth` et la vérification des dates dans le passé).
- Commentez les résultats du rapport de couverture et expliquez comment vous avez couvert les segments de code manquants.

## 5.3 Étape 3 : Utilisation de Mocks

**Objectif** : Isoler les dépendances en utilisant des mocks pour les interactions avec la base de données dans le test de complétion de tâche.

- Utilisez `pytest-mock` pour simuler les interactions avec la base de données dans la route de complétion.
- Testez que la fonction ajoute bien un score pour la tâche complétée sans toucher la base de données réelle.
- Discutez des avantages d'utiliser des mocks pour ces tests.

## 5.4 Étape 4 : Tests Bout-en-Bout (E2E)

**Objectif** : Tester le flux complet de création, complétion et nettoyage d'une tâche.

- Écrivez des tests E2E couvrant le cycle de vie d'une tâche : ajout, complétion, et nettoyage.
- Vérifiez qu'après le nettoyage, la tâche complétée soit bien supprimée de la base.

## 5.5 Étape 5 : Tests d'Intégration et Validation de l'État de la Base de Données

**Objectif** : Valider l'intégration entre la table des tâches et celle des scores.

- Écrivez des tests pour vérifier que la feature *compléter une tâche* met bien à jour le score total dans la base de données.

— Confirmez que la feature *suppression des tâches obsolètes* ne modifie pas les scores existants.
— Testez que les fonctions de nettoyage et de calcul de score interagissent bien avec les tables respectives.

### 5.6 Étape 6 : Génération de Rapport de Tests

**Objectif** : Fournir un rapport de tests structuré en HTML pour examiner les résultats et la couverture.

— Générez un rapport de tests avec `pytest-cov` en HTML.
— Analysez le rapport HTML pour identifier toute amélioration de couverture possible.
— Commentez les résultats et identifiez les points forts et faibles des tests.

## 6 Évaluation

L'évaluation portera sur :
— La complétude des tests écrits (fonctionnels, boîte noire/blanche, E2E, intégration).
— La bonne utilisation d'un mock.
— La couverture de code et la qualité du rapport généré.
— La capacité à identifier et documenter les cas non couverts ou les potentiels d'amélioration.

# 7 Liste des features

Ce document décrit les requêtes `curl` pour utiliser l'API de gestion des tâches, cependant, vous ne les utiliserez que pour activer l'API à la main.

Lors de vos tests, vous devrez utiliser un client de test de Flask :

Listing 2 – Utiliser un client de test flask

```
@pytest.fixture
def test_app():
    app.config.update(
        {
            "TESTING": True,
        }
    )
    return app


@pytest.fixture
def client(test_app):
    return test_app.test_client()


def test_url(client):
    response = client.post("/URL", json={"key": "payload"})

    assert response.status_code == 201
    assert response.get_json() == {"message": "Tout␣fonctionne"}
```

## 7.1 Ajout d'une tâche

Pour ajouter une tâche, utilisez la route `POST /tasks`.

La requête suivante ajoute une tâche avec un titre, une description, une priorité et une difficulté.

Listing 3 – Ajout d'une tâche

```
curl -X POST http://localhost:5000/tasks -H "Content-Type:␣application/json" -d '{
␣␣"title":␣"Preparer␣le␣rapport",
␣␣"description":␣"Finaliser␣le␣rapport␣pour␣la␣reunion",
␣␣"due_date":␣"2023-12-15",
␣␣"priority":␣3,
␣␣"difficulty":␣2
}'
```

## 7.2 Complétion d'une tâche

Pour marquer une tâche comme terminée, utilisez la route `POST /tasks/{task_id}/complete`, où `{task_id}` est l'identifiant de la tâche.

Listing 4 – Complétion d'une tâche

```
curl -X POST http://localhost:5000/tasks/1/complete
```

Cette requête marque la tâche avec l'identifiant `1` comme terminée et ajoute le score calculé à la table des scores.

## 7.3 Consultation des tâches actives

Pour consulter toutes les tâches actives, utilisez la route `GET /tasks/active`.

Cette requête renvoie une liste des tâches non terminées avec une date d'échéance dans le futur ou sans date d'échéance.

Listing 5 – Consultation des tâches actives

```
curl -X GET http://localhost:5000/tasks/active
```

### 7.4 Nettoyage des tâches obsolètes ou complétées

Pour supprimer les tâches terminées ou avec une date d'échéance passée, utilisez la route `DELETE /tasks/cleanup`.

Listing 6 – Nettoyage des tâches obsolètes ou complétées

```
curl -X DELETE http://localhost:5000/tasks/cleanup
```

Cette requête supprime toutes les tâches obsolètes ou complétées de la base de données et renvoie le nombre de tâches supprimées.

### 7.5 Consultation du score total

Pour consulter le score total des tâches complétées, utilisez la route `GET /scores/total`.

Listing 7 – Consultation du score total

```
curl -X GET http://localhost:5000/scores/total
```

Cette requête retourne le score total accumulé pour toutes les tâches complétées.