# Travail dirigé : Tests sur une Application déjà existante

© Pierre-Antoine Guillaume

2024-10-31

## TD : Conception et exécution de tests pour une application de gestion de bibliothèque

### Objectif

L'objectif de ce TD est de mettre en pratique plusieurs concepts de test :
— Black Box et White Box Testing
— Création de mocks
— Calcul de la couverture de tests
— Tests d'intégration et tests de bout en bout (End-to-End)
— Lecture et interprétation des rapports de tests

### Contexte

Vous devez réaliser les tests d'une application de gestion de stock pour une bibliothèque.
Cette application a trois grandes features :

1. Ajout de livres : Les utilisateurs peuvent ajouter des livres en fournissant son titre, son auteur, son année de publication et son stock.
2. Recherche de livres : Les utilisateurs ont la possibilité de rechercher des livres par titre ou auteur.
3. Emprunt de livres : Les utilisateurs peuvent emprunter un livre si celui-ci est disponible en stock.

Le code de l'application vous est fourni, avec les fonctions principales.

— D'un point de vue technique, il s'agit d'une API web, qui utilise Flask et sqlite.
— Flask est un framework web léger.
— sqlite est un SGBR *en mémoire*, qui est sauvegardée dans un fichier (En l'occurence session_2 /var/library.sqlite, c'est renseigné dans la variable d'environnement DATABASE_URL remplie par le fichier session_2/.env)
— Vous pouvez récupérer l'application dans le repository https://github.com/PierreAntoineGuillaume /methodologie−du−test, que vous pouvez cloner ou recloner.
— Pour lancer l'application, vous devez vous positionner dans le dossier session_2
— Puis soit lancer l'application sur votre poste (via le script start, décliné sous forme PowerShell également), soit le lancer dans un container docker via docker compose up
— Des commandes curl vous sont fournies dans le README du dossier session_2. cURL est un outil pour faire des appels HTTP (entre autres) depuis un terminal.

### Plan du TD

1. Mise en place et exploration (10-15 minutes)
2. Écrire des tests Black Box et White Box (20 minutes)
   — Black Box Testing : Créer des tests pour valider les fonctionnalités principales sans se baser sur le code interne (tests fonctionnels sur les entrées/sorties).
   — White Box Testing : Ajouter des tests pour couvrir des branches ou des conditions spécifiques, comme les vérifications sur les cas limites (ex. : emprunt d'un livre déjà emprunté).

   Pour obtenir un client qui permet d'appeler un client flask, vous devez définir une fixture pour ce client :

```python
from src.librairie import app, get_db

@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_my_url(client):
    response = client.post('/my/url')
    assert response.status_code == 200
    assert response.get_json() == {"key": "value"}
```

3. Créer des Mocks pour les dépendances (10-15 minutes)
   — Créer un mock de la base de données pour la fonction d'emprunt, en simulant l'état de disponibilité des livres sans modifier la base de données réelle.

   En installant la bibliothèque pytest−mock, vous pourrez utiliser mocker dans vos test-cases : def test_borrow_book_with_mock(mocker, client): Cette bibliothèque est déjà installée si vous avez installé les dépendances pip du dossier session_2.

```python
def test_borrow_book_with_mock(mocker, client) -> None:
    mock_db = mocker.patch('src.librairie.get_db')
    mock_cursor = mocker.MagicMock()
    mock_cursor.fetchone.return_value = (1,) # return value doit etre "subscriptable", par
        exemple un tuple
    mock_db.return_value.cursor.return_value = mock_cursor
```

4. Calcul de la couverture et amélioration des tests (10 minutes)
   — Utiliser pytest-cov pour génèrer un rapport de couverture.

```
(.venv) $ pytest --cov=src --cov-report=html
```

   — Lire le rapport dans htmlcov/index.html et ajouter des tests pour atteindre une couverture de 80 à 90
   Attention, une fonction mockée ne peut pas être couvertes. Vous pouvez ignorer les méthodes trop situationnelles (Elles ont été marquées par la directive #pragma: no cover, pour indiquer à pytest-cov de l'ignorer)

5. Exécution d'un test d'intégration ou de bout en bout (15 minutes)
   — Configurer un test d'intégration ou E2E pour vérifier l'enchaînement des fonctionnalités
   — Utiliser une approche systématique pour tester l'ensemble du flux utilisateur, en s'assurant que chaque fonctionnalité interagit correctement avec les autres.

6. Lecture et analyse d'un rapport de test (10 minutes)
   — Fournir un rapport de test (ou demander aux étudiants de lire celui qu'ils viennent de générer).
   — Analyser les résultats, identifier les échecs et discuter des actions correctives possibles.