Couverture de code et intégration continue

Les tests sont utiles, mais **encore faut-il qu'ils couvrent l'essentiel de votre application**. La **couverture de code** permet de savoir quelles parties du code ont été exécutées lors des tests automatisés.

Sommaire

Pourquoi mesurer la couverture ?

La couverture de test vous aide à :

- **Identifier les parties du code non testées**
- **Renforcer la qualité des tests** (éviter les faux positifs)
- **Prioriser vos efforts** de test en cas de bugs ou de réécriture

> ⚠ Une couverture élevée n'est pas synonyme de qualité : 100% de couverture ne garantit pas qu'un test est pertinent. Il s'agit simplement d'un **indicateur utile**, pas d'un objectif en soi.

Installation et utilisation de pytest-cov

Installez le module `pytest-cov`, puis utilisez-le pour exécuter vos tests :

```
pip install pytest-cov
pytest --cov=app
```

Explication :

- `--cov=app` : indique que l'on veut mesurer la couverture du dossier `app/` (remplace par le nom de ton dossier)
- Le résultat indique le pourcentage de lignes de code testées

Interpréter les résultats

Un exemple de sortie :

```
---------- coverage: platform linux, python 3.11 ----------
Name               Stmts   Miss  Cover
------------------------------------
app/main.py           30      4    86%
app/routes.py         50      0   100%
app/models.py         20     10    50%
------------------------------------
TOTAL                100     14    86%
```

- `Stmts` : nombre total de lignes exécutables
- `Miss` : lignes non couvertes
- `Cover` : pourcentage de lignes testées

💡 Objectif réaliste en formation : atteindre **70–90%** de couverture, avec des tests utiles.

Option : générer un rapport HTML

Pour visualiser les lignes non couvertes directement dans le code, ajoute l'option `--cov-report html` :

```
pytest --cov=app --cov-report=html
```

Un dossier `htmlcov/` est généré. Ouvre `htmlcov/index.html` dans ton navigateur pour explorer les fichiers ligne par ligne.

Bonus : automatiser les tests avec GitHub Actions

GitHub Actions permet de **lancer automatiquement les tests** à chaque `push` ou `pull request`. Voici un exemple de configuration simple (`.github/workflows/tests.yml`) :

```yaml
name: Run tests

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest # Choisis l'environnement d'exécution
    steps:
    - uses: actions/checkout@v3 # Télécharge le code
    - name: Set up Python
      uses: actions/setup-python@v4 # Set up Python
      with:
        python-version: '3.11'
    - name: Install dependencies
      run: pip install -r requirements.txt pytest pytest-cov # Installe les
dépendances
    - name: Run tests
      run: pytest --cov=app # Exécute les tests avec couverture
```

📌 Ce workflow :

- installe Python,
- télécharge le code,
- installe les dépendances,
- exécute les tests avec couverture.

> 🔎 Ce type d'automatisation est **optionnel** mais devient essentiel dans des projets collaboratifs ou professionnels.