15 OCTOBRE 2025

eFrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ

CONCEPTEUR DÉVELOPPEUR WEB
FULL STACK
N° 39608 – NIVEAU 6

TESTS AUTOMATISÉS & INTÉGRATION

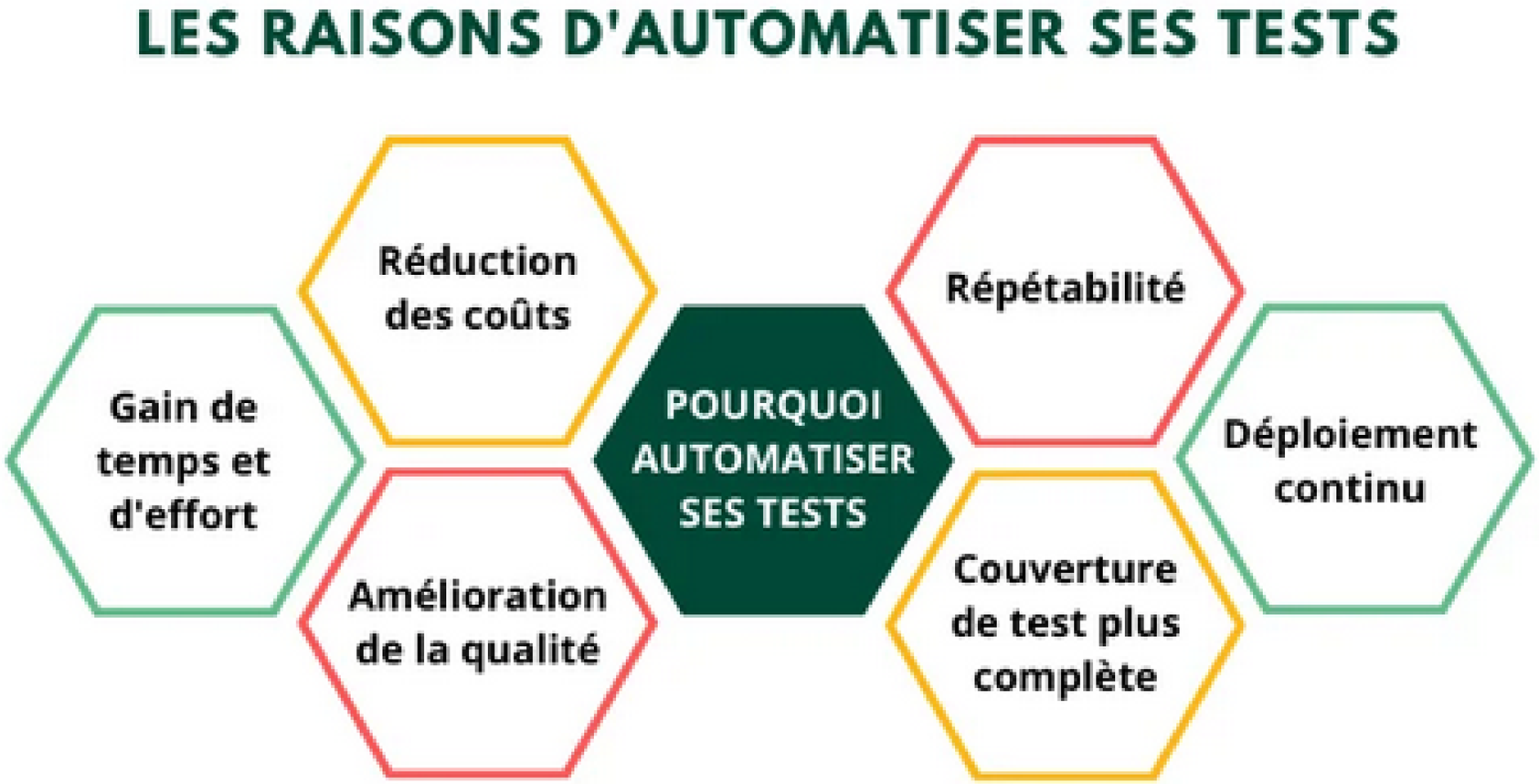
FORMATEUR : DIALLO ALIMOU

OBJECTIFS D’APPRENTISSAGE

- Appliquer les standards de qualité pour un code lisible et maintenable.
- Mener des revues de code efficaces pour détecter tôt les anomalies.
- Concevoir et automatiser les tests unitaires et d’intégration.
- Intégrer les tests dans la CI/CD pour sécuriser les déploiements.
- Rédiger et maintenir une documentation technique complète et utile.

La notion de test est souvent sous-estimée par les **développeurs**. Ils se lancent dans des projets qui ont pour vocation à grandir et négligent l'atout que représente **les tests** pour maintenir leur développement. Pour comprendre leur nécessité, il faut d'abord comprendre ce que sont ces **test**

Les tests automatisés sont une méthode de test logiciel permettant d'exécuter rapidement et de façon répétée des vérifications sans intervention humaine. Ils augmentent l'efficacité des tests en détectant promptement les erreurs ou bugs avant la mise en production.



TEST UNITAIRE

Le but d'un test unitaire est de vérier le bon fonctionnement d'une composante d'un programme, le plus souvent une fonction.

Le test unitaire essaiera d'être exhaustif en testant tous les types de scénarios possibles pour une fonction. Par exemple, si une fonction contient une condition, il faudra au moins deux tests : un dans lequel la condition est vraie et l'autre dans lequel la condition est fausse.

Une telle exhaustivité permet aussi de vérier que chaque partie du code est utile et atteignable dans des scénarios réalistes.

TEST FONCTIONNEL

Le but d’un test **fonctionnel** est de vérifier le bon fonctionnement d’une fonctionnalité complète d’un programme.

Ce type de test s’effectue en **boîte noire,** c’est-à-dire sans accéder au **code source**.

Contrairement au test unitaire, qui évalue une partie isolée du code, le test fonctionnel examine **le comportement global du programme**.

Il est donc moins exhaustif qu’un test unitaire, mais il permet de valider **l’intégration et le bon assemblage des différentes composantes du logiciel.**

LES FRAMEWORKS DE TEST

Il existe des frameworks de test qui encadrent et facilitent l'écriture et l'utilisation de tests. Ils complexient l'écriture des programmes, mais dans le cadre de projets professionnels, il est vivement recommandé d'utiliser ces frameworks.

- En Python, les deux frameworks principaux sont **unittest** et **pytest**. Le premier est le framework historique inclus par défaut à l'installation de Python. Le second est le plus récent et commence à faire l'unanimité dans la communauté Python.
- En JavaScript, il existe de nombreux frameworks mais les plus populaires sont : **Mocha** et **Jasmine**.

On appellera test unitaire une fonction qui appelle de multiples fois une autre fonction afin de tester sa validité selon différents paramètres.

OUVERTURE DE CODE

Une ligne de code est considérée comme couverte par un test si elle est exécutée durant le test.
Pour couvrir complètement le code, il faut tester la fonction avec différentes valeurs permettant de vérier toutes les conditions.
Les conditions doivent être couvertes car elles sont souvent à l'origine de bugs.
Les **frameworks** de test permettent de calculer facilement la couverture de son code.

EXEMPLE TESTER DES DES VALEURS

```python
def panier_prix(liste_prix_articles):
    """Calcule le total du panier (somme des prix)."""
    total = 0
    for prix in liste_prix_articles:
        total += prix
    return total

```

Voici un exemple de test sur une fonction simple. Si le résultat attendu n'est pas identique à celui retourné par la fonction, le test échoue et ache un message d'erreur.

UTILISER LA SYNTAXE ASSERT

Pour simplifier les tests, il est possible d'utiliser un élément de syntaxe : l'assertion. Une assertion ne fera rien de particulier si l'expression qui la suit est vraie mais une erreur sera émise si l'expression est fausse (AssertionError pour être précis).

```python
def test_panier_prix_normal():
    """Test : panier avec plusieurs articles."""
    assert panier_prix([2, 5, 29]) == 36, "Le total doit être 36 pour [2,5,29]"

def test_panier_vide():
    """Test : panier vide."""
    assert panier_prix([]) == 0, "Le total doit être 0 pour un panier vide"

def test_panier_avec_flottants():
    """Test : panier contenant des valeurs décimales."""
    assert panier_prix([1.5, 2.5, 3.0]) == 7.0, "Le total doit être 7.0 pour [1.5,2.5,3.0]"
```

RESULTAT DES EXEMPLES DE TEST

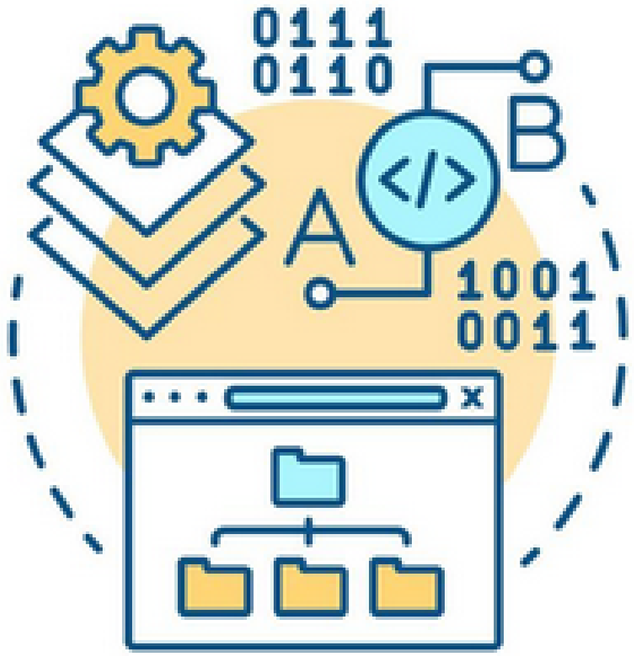
```
$ pytest -v
========================================= test session starts =========================================
platform win32 -- Python 3.11.9, pytest-8.4.2, pluggy-1.6.0 -- C:\Users\Utilisateur\AppData\Local\
\Python\Python311\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Utilisateur\Desktop\efrei\Projet\test
plugins: anyio-4.11.0
collected 5 items

fonction_test.py::test_panier_prix_normal PASSED                                             [ 20%]
fonction_test.py::test_panier_vide PASSED                                                    [ 40%]
fonction_test.py::test_panier_avec_flottants PASSED                                          [ 60%]
fonction_test.py::test_panier_negatif PASSED                                                 [ 80%]
fonction_test.py::test_panier_unique PASSED                                                  [100%]

========================================== 5 passed in 0.08s ==========================================
```

TEST D'INTÉGRATION

Un test d'intégration vérifie que la communication de plusieurs fonctions se fait comme prévue. **Il contrôle les assemblages de fonctions.**



Les principales technologies de test

Outil / Framework	Rôle	Points forts
Jest	Framework complet de tests (unitaires, intégration, mocks, couverture de code)	Simple, rapide, très utilisé (Facebook)
Mocha + Chai	Mocha = moteur de test ; Chai = assertions (vérifications)	Flexible, très populaire, syntaxe lisible
Supertest	Teste les routes d'une API Express (requêtes HTTP virtuelles)	Idéal pour tests d'intégration

EXEMPLE SIMPLE AVEC JEST

Initialiser le projet Node.js

`npm init -y` Cela crée un fichier package.json par défaut avec les informations de base.

Installer Jest

```
npm install --save-dev jest
```

Lancer les tests

```
npm test
```

UN FICHIER DE TEST

JS calcul.js > ...

```
1  function addition(a, b) {
2      return a + b;
3  }
4
5  function soustraction(a, b) {
6      return a - b;
7  }
8
9  module.exports = { addition, soustraction };
10
```

JS calcul.test.js > ...

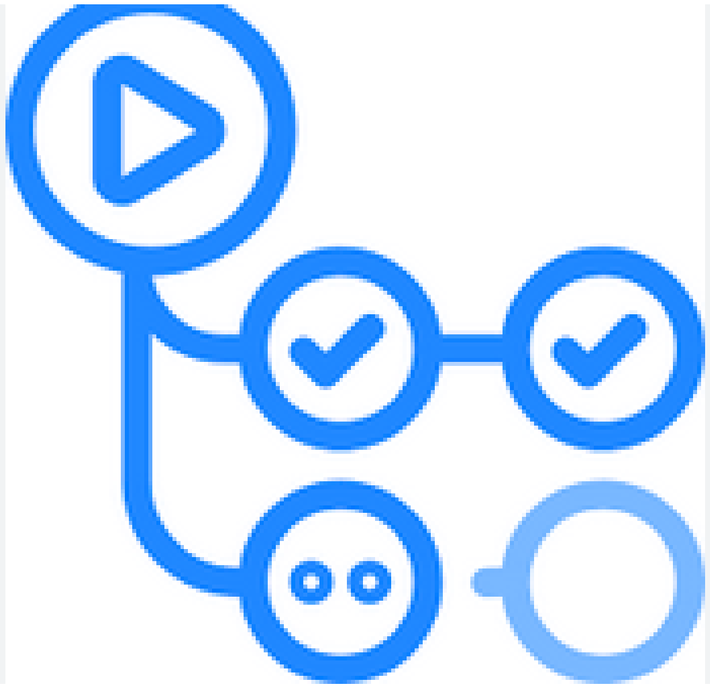
```
1   const { addition, soustraction } = require('./calcul');
2
3   test('addition(2, 3) doit retourner 5', () => {
4     expect(addition(2, 3)).toBe(5);
5   });
6
7   test('soustraction(5, 3) doit retourner 2', () => {
8     expect(soustraction(5, 3)).toBe(2);
9   });
10
```

RESULTAT

```
PASS  ./calcul.test.js
  ✓ addition(2, 3) doit retourner 5 (4 ms)
  ✓ soustraction(5, 3) doit retourner 2 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.652 s, estimated 1 s
Ran all test suites.
```

GITHUB ACTIONS



GITHUB ACTIONS

GitHub Actions est un outil d'intégration **continue (CI)** et de **déploiement continu (CD)** intégré à GitHub.

Il permet d'automatiser des tâches comme :

- Lancer les tests (unitaires, intégration, etc.)
- Construire et déployer une application
- Vérifier le code (lint, qualité)
- Publier des packages NPM ou Docker

Tout cela se fait à l'aide de fichiers **YAML** dans ton dépôt GitHub.

EXEMPLE DE WORKFLOW COMPLET POUR JEST

Fichier : .github/workflows/node-tests.yml

GITHUB/WORKFLOWS/NODE-TESTS.YML → TON WORKFLOW GITHUB ACTIONS

SRC/ECOMMERCE.JS → LE CODE SOURCE

TESTS/COMMERCE.TEST.JS → LES TESTS JEST

PACKAGE.JSON / PACKAGE-LOCK.JSON → LA CONFIGURATION NPM

.GITIGNORE → POUR IGNORER LES FICHIERS INUTILES

TP JOURNÉE