diginamic FORMATION
« Votre passeport pour l'emploi numérique »

# IV. Tests automatisés avec Pytest

## 01. Pourquoi écrire des tests ?

- **Prévenir les bugs** avant qu'ils n'arrivent en production
- **Documenter le comportement attendu** d'une fonction ou d'une API
- **Gagner du temps** sur les tests manuels répétitifs
- Faciliter la **refactorisation** du code en évitant les régressions

## 02. Types de tests

- **Tests unitaires** Testent une **fonction isolée** sans dépendance (logique métier pure)
- **Tests d'intégration** Vérifient le bon fonctionnement entre **plusieurs composants** (ex. route FastAPI + accès base de données)
- **Tests end-to-end (E2E)** Simulent le parcours d'un utilisateur réel (navigateur, clics, etc.) ⚠ Non couvert dans ce module

## 03. Introduction à `pytest`

- Outil de test Python **rapide**, **lisible** et **extensible**
- Utilise les **assertions natives** de Python (`assert`)
- Convention :
  - Les **fichiers** de test commencent par `test_`
  - Les **fonctions** de test aussi : `def test_nom()`

## 04. Introduction à `httpx` pour tester une API

- Client HTTP **asynchrone** compatible avec FastAPI
- Permet de simuler des requêtes `GET`, `POST`, etc.
- Utilisable avec `TestClient` pour tester localement sans lancer de serveur

## 05. Fixtures utiles

- **Base de données temporaire** pour éviter d'impacter les vraies données
- **Client de test FastAPI** :

```
from fastapi.testclient import TestClient
```

- Utilisation de `@pytest.fixture` pour partager des ressources (ex : session DB, client API) entre plusieurs tests

## 06. Couverture de code et CI

- **Mesurer la couverture** des tests avec `pytest-cov` ou `coverage`
- Identifier les fichiers ou lignes **non testés**
- Possibilité d'intégrer les tests dans une **CI** (ex : GitHub Actions) pour lancer les tests à chaque push