Introduction à Pytest

Pytest est un framework de test moderne, simple et puissant pour Python. Contrairement à `unittest`, il ne nécessite pas de classes, fonctionne à base de fonctions et repose sur les **assertions natives de Python**.

Sommaire

Pourquoi choisir `pytest` ?

- 📦 **Léger et sans surcharge** : pas besoin d'hériter de classes.
- 🧼 **Lisibilité maximale** : les tests sont écrits comme des fonctions Python normales.
- 🚀 **Flexible et extensible** : fixtures, hooks, plugins, et paramétrage puissant.
- 🛠️ **Compatible avec unittest** : possibilité de migrer progressivement.

Structure d'un test avec `pytest`

Voici un test simple :

```
def test_addition():
    assert 1 + 1 == 2
```

🔧 Exécution

```
pytest
```

Par défaut, `pytest` recherche :

- Tous les fichiers nommés `test_*.py` ou `*_test.py`
- Toutes les fonctions dont le nom commence par `test_*`

Il affiche clairement les succès, les erreurs, et les messages d'échec.

Comparaison rapide avec `unittest`

Aspect	`unittest`	`pytest`
Syntaxe	Basée sur les classes	Fonctions simples
Framework	Bibliothèque standard	Installation externe
Assertions	`self.assertEqual(a, b)`	`assert a == b`
Verbosité	Plus verbeux	Plus concis
Fixtures	setUp/tearDown	`@pytest.fixture`, très puissantes
Plugins	Limité	Très riche écosystème

Bonnes pratiques avec `pytest`

- Nommer les tests avec **des fonctions explicites** (`test_xxx`)
- Ajouter des **messages d'erreur personnalisés** avec `assert`
- Séparer les **tests unitaires** et les **tests d'intégration**
- Grouper les tests liés dans des **fichiers thématiques** (`test_utils.py`, `test_users.py`, etc.)
- Utiliser des **fixtures** pour isoler les dépendances (ex : base de données temporaire)

Philosophie : Moins de code, plus de clarté

Exemple `unittest` :

```
import unittest

class TestAddition(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(1 + 1, 2)
```

Même test avec `pytest` :

```
def test_addition():
    assert 1 + 1 == 2
```

👉 Même efficacité, mais beaucoup plus concis.

Organisation d'un projet de test avec `pytest`

```
mon_projet/
├── app/
│   └── ... (code source)
├── tests/
│   ├── __init__.py
│   ├── test_routes.py
│   ├── test_utils.py
│   └── ...
├── requirements.txt
└── requirements-dev.txt
```

Fichier `requirements-dev.txt` :

```
pytest
pytest-cov
httpx
```

À retenir

- `pytest` est **simple mais puissant** : il permet de couvrir tous les cas de test, des plus simples aux plus complexes.
- Il est **parfaitement adapté à FastAPI**, grâce à sa compatibilité avec `httpx`, `TestClient`, et ses fixtures élégantes.
- Il sera le **cœur de votre stratégie de test** dans ce cours.