TDD avec PyTest et FastAPI

Objectifs pédagogiques

À l'issue de ce support, l'apprenant sera capable de :

- Comprendre le **principe du TDD** (Test Driven Development)
- Écrire des **tests PyTest** pour une API FastAPI
- Tester une API **sans lancer de serveur**
- Séparer **logique métier** et **couche HTTP**
- Mettre en place une **base saine de tests automatisés**

Rappels : qu'est-ce que le TDD ?

Le TDD repose sur un cycle court et itératif :

> 🔴 **Red** → 🟢 **Green** → 🔵 **Refactor**

1. **Red** : écrire un test qui échoue
2. **Green** : écrire le minimum de code pour faire passer le test
3. **Refactor** : améliorer le code sans casser les tests

👉 Le test **définit le besoin**, le code **répond au besoin**.

Stack technique utilisée

- **Python 3.10+**
- **FastAPI**
- **PyTest**
- **httpx / TestClient**

Installation :

```
pip install fastapi pytest httpx uvicorn
```

Structure minimale du projet

```
app/
 ├── main.py
 ├── services.py
tests/
 ├── test_main.py
 ├── test_services.py
```

👉 **Bonne pratique TDD** : tester d'abord la logique métier (`services.py`) avant l'API.

Cas d'étude : calcul de bonus salarial

Règle métier

- Ventes < 50 000 € → bonus de 5 %
- Ventes ≥ 50 000 € → bonus de 10 %
- Vente négatif → erreur

Étape 1 — TDD sur la logique métier (sans FastAPI)

🔴 Test d'abord (`tests/test_services.py`)

```
import pytest
from app.services import compute_bonus

def test_bonus_low_sales():
    assert compute_bonus(40000) == 2000

def test_bonus_high_sales():
    assert compute_bonus(60000) == 6000

def test_negative_sales():
    with pytest.raises(ValueError):
        compute_bonus(-1000)
```

👉 À ce stade : **les tests échouent**, la fonction n'existe pas encore.

Implémentation minimale (`app/services.py`)

```
def compute_bonus(sales: float) -> float:
    if sales < 0:
        raise ValueError("Sales must be positive")

    if sales < 50000:
        return sales * 0.05
    return sales * 0.10
```

```
bash
        pytest
```

✅ Les tests passent.

Étape 2 — Exposer la logique via FastAPI

API (`app/main.py`)

```
from fastapi import FastAPI, HTTPException
from app.services import compute_bonus

app = FastAPI()

@app.get("/bonus")
def get_bonus(sales: float):
    try:
        return {
            "sales": sales,
            "bonus": compute_bonus(sales)
        }
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
```

Étape 3 — Tests de l’API FastAPI

Test API (`tests/test_main.py`)

```
from fastapi.testclient import TestClient
import os
import sys

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))
from app.main import app
```

```
client = TestClient(app)

def test_bonus_endpoint_low_sales():
    response = client.get("/bonus?sales=40000")
    assert response.status_code == 200
    assert response.json()["bonus"] == 2000

def test_bonus_endpoint_negative_sales():
    response = client.get("/bonus?sales=-1")
    assert response.status_code == 400
```

👉 Ici :

- Pas de serveur lancé
- FastAPI est testé **en mémoire**
- Les tests sont **rapides et reproductibles**

Sources et références (fiables)

Officielles

- FastAPI – Testing
  https://fastapi.tiangolo.com/tutorial/testing/
- PyTest – Documentation
  https://docs.pytest.org/
- TestClient (Starlette)
  https://www.starlette.io/testclient/

TDD

- Martin Fowler – TDD
  https://martinfowler.com/bliki/TestDrivenDevelopment.html
- Kent Beck – *Test Driven Development: By Example*

Bonnes pratiques API

- FastAPI Best Practices
  https://fastapi.tiangolo.com/advanced/
