

Rapport de Mini Projet

Système de Surveillance Énergétique (IoT)

Réalisé par : BOUKHARI Redha
Module : Programmation (Doctorat)

16 décembre 2025

Table des matières

1	Introduction et Contexte	2
1.1	Objectifs du Projet	2
2	Architecture Technique	2
2.1	Structure du Projet	2
2.2	Implémentation des Modules	2
2.2.1	1. Simulation (sensor.py)	2
2.2.2	2. Analyse (analyzer.py)	3
2.2.3	3. Persistance (database.py)	3
3	Mise en Place de l'Environnement (Docker)	3
3.1	Création de l'image et lancement	3
4	Qualité Logicielle et Tests	4
4.1	Tests Unitaires (Unit Testing)	4
4.2	Linting (Flake8)	4
5	Exécution et Résultats	5
5.1	Simulation en temps réel	5
5.2	Validation de la persistance (MongoDB Compass)	5
6	Conclusion	6

1 Introduction et Contexte

Ce projet a été réalisé dans le cadre du module de programmation et gestion de données. Il vise à simuler une architecture IoT (Internet of Things) complète pour le Laboratoire de Recherche LARI.

1.1 Objectifs du Projet

L'objectif principal est de mettre en œuvre une chaîne de traitement de données (*Data Pipeline*) capable de :

- **Simuler** des données de consommation électrique via des capteurs virtuels.
- **Analyser** ces données en temps réel pour détecter des anomalies (seuil fixé à 12.0 kWh).
- **Stocker** l'historique des mesures dans une base de données NoSQL (**MongoDB**).
- **Assurer** la portabilité via **Docker** et la qualité du code via des tests unitaires et du linting.

2 Architecture Technique

Le projet repose sur une architecture modulaire respectant les principes de la Programmation Orientée Objet (POO). Le code est structuré en quatre fichiers principaux situés dans le dossier `src/`.

2.1 Structure du Projet

L'arborescence des fichiers est la suivante :

- `src/sensor.py` : Simulation de la génération de données.
- `src/analyzer.py` : Logique métier de détection d'anomalies.
- `src/database.py` : Gestion de la connexion MongoDB (Docker).
- `src/main.py` : Point d'entrée et orchestration.
- `tests/` : Dossier contenant les tests unitaires.
- `Dockerfile` : Configuration de l'image conteneurisée.

2.2 Implémentation des Modules

2.2.1 1. Simulation (`sensor.py`)

La classe `EnergySensor` génère des valeurs flottantes aléatoires pour simuler la variabilité d'un capteur physique.

```
1 import random
2
3 class EnergySensor:
4     def generate_data(self):
5         # Génération aléatoire entre 5.0 et 15.0 kWh
6         return round(random.uniform(5.0, 15.0), 2)
```

Listing 1 – Génération des données

2.2.2 2. Analyse (analyzer.py)

Ce module contient l'intelligence du système. Le seuil critique est paramétrable.

```
1 class EnergyAnalyzer:
2     def __init__(self, threshold=12.0):
3         self.threshold = threshold
4
5     def check_anomaly(self, value):
6         return value > self.threshold
```

Listing 2 – Détection d'anomalies

2.2.3 3. Persistance (database.py)

La connexion est configurée pour pointer vers le conteneur Docker écoutant sur le port **27018**.

```
1 from pymongo import MongoClient
2
3 class DatabaseHandler:
4     def __init__(self):
5         # Connexion au port exposé par Docker (27018)
6         self.uri = "mongodb://admin:pass@localhost:27018/..."
7         self.client = MongoClient(self.uri)
8         self.db = self.client["energy_db"]
```

Listing 3 – Connexion MongoDB

3 Mise en Place de l'Environnement (Docker)

Pour garantir l'isolation et la reproductibilité de la base de données, nous utilisons la technologie de conteneurisation Docker.

3.1 Crédit de l'image et lancement

Le fichier Dockerfile configure une instance MongoDB sécurisée. Le conteneur est lancé en mappant le port interne 27017 vers le port hôte **27018** pour éviter les conflits locaux.

```
1 docker build -t energy-mongo .
2 docker run -d -p 27018:27017 --name energy-db-container energy-mongo
```

La figure ci-dessous atteste du bon fonctionnement du conteneur.

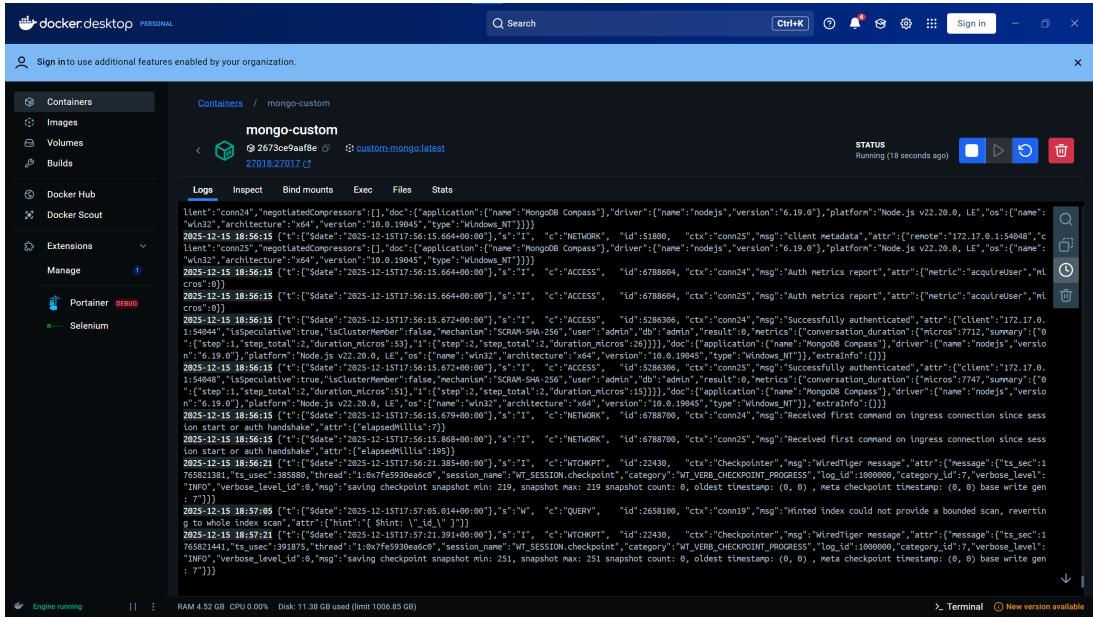


FIGURE 1 – Interface Docker Desktop montrant le conteneur MongoDB actif sur le port 27018.

4 Qualité Logicielle et Tests

Dans le cadre d'une démarche scientifique rigoureuse, la qualité du code a été validée selon deux axes : les tests fonctionnels et l'analyse statique.

4.1 Tests Unitaires (Unit Testing)

Nous utilisons le module `unittest` pour découvrir et lancer les tests définis dans le dossier `tests/`.

```
(venv) C:\Users\TRETEC\Documents\these\Cours Pedagogique\Programmation\Mini Projet>pytest
=====
platform win32 -- Python 3.11.7, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\TRETEC\Documents\these\Cours Pedagogique\Programmation\Mini Projet
collected 4 items

tests\test_project.py .... [100%]

=====
4 passed in 0.02s =====
```

FIGURE 2 – Exécution de la commande de découverte des tests unitaires.

4.2 Linting (Flake8)

Le respect des normes **PEP 8** a été vérifié à l'aide de l'outil `flake8`. L'absence de retour dans la console ci-dessous indique que le code dans le dossier `src/` est conforme aux standards de style.

```
(venv) C:\Users\TRETEC\Documents\these\Cours Pedagogique\Programmation\Mini Projet>flake8 src/  
(venv) C:\Users\TRETEC\Documents\these\Cours Pedagogique\Programmation\Mini Projet>
```

FIGURE 3 – Vérification du code avec Flake8 (aucune erreur détectée).

5 Exécution et Résultats

5.1 Simulation en temps réel

L'exécution du script principal `src/main.py` démarre la boucle de surveillance. Comme illustré dans la Figure 4, le terminal affiche les logs d'insertion. Le système distingue visuellement les consommations normales des alertes de surconsommation (lorsque la valeur dépasse 12.0 kWh).

The screenshot shows a Python development environment with the following details:

- File Explorer:** Shows the project structure with files like `database.py`, `analyzer.py`, `main.py`, `test_project.py`, `falke8`, `sensor.py`, `README.md`, `Dockerfile`, and `tests`.
- Code Editor:** The `database.py` file is open, containing code for interacting with a MongoDB database. It includes methods for inserting data and closing the connection.
- Terminal:** The terminal window shows the following log output:

```
[DB] Data inserted with ID: 69404bf585fcac68d292e0dd
[DB] [18:57:12] Consumption is normal.
[DB] Data inserted with ID: 69404bf585fcac68d292ede0
[DB] [18:57:14] Consumption is normal.
[DB] Data inserted with ID: 69404bf585fcac68d292edf
[DB] [18:57:16] Consumption is normal.
[DB] Data inserted with ID: 69404bf585fcac68d292e0d0
[DB] [18:57:18] Consumption is normal.
[DB] Data inserted with ID: 69404bf585fcac68d292ede1
[DB] [18:57:20] ALERT! High consumption detected. (12.24 kWh > 12.0 kWh)
[DB] Data inserted with ID: 69404bf585fcac68d292ede2
[DB] [18:57:22] Consumption is normal.
```
- Bottom Status Bar:** Shows the current file is `database.py`, and the status bar indicates: Ln 54, Col 54 | Spaces: 4 | UFT-8 | CRLF | Python | venv (3.11.7) | Q Mini Proj.

FIGURE 4 – Terminal VS Code montrant l'exécution du script principal. Les alertes de surconsommation sont mises en évidence.

5.2 Validation de la persistance (MongoDB Compass)

La vérification via l'interface graphique MongoDB Compass confirme que les données sont correctement historisées dans la collection **measures**.

Les figures 5 et 6 montrent les documents insérés. Chaque document contient l'horodatage, la valeur mesurée, le lieu, l'identifiant du capteur et surtout l'indicateur booléen `is_anomal` qui est positionné à `true` en cas de dépassement du seuil.

The screenshot shows the MongoDB Compass interface for a database named 'energy_db' and a collection named 'measures'. The table displays 168 documents. One document is highlighted in blue, representing an anomaly with a value of 12.56 kWh. The columns include: _id, ObjectId, sensor_id, location, value_kWh, timestamp, and is_anomaly. The 'is_anomaly' column shows 'No field' for most rows and 'true' for the highlighted row.

FIGURE 5 – Vue MongoDB Compass montrant une première série de mesures avec une anomalie détectée (12.56 kWh).

This screenshot shows a detailed view of historical measurement data from the 'energy_db' collection. It highlights several anomalies, including rows with values 14.21 kWh and 13.35 kWh. The table has 108 rows and includes columns for _id, ObjectId, sensor_id, location, value_kWh, timestamp, and is_anomaly. The 'is_anomaly' column identifies specific rows as anomalies.

FIGURE 6 – Vue détaillée de l'historique montrant d'autres anomalies capturées (ex : 14.21 kWh et 13.35 kWh).

6 Conclusion

Ce mini projet a permis de concevoir une architecture IoT fonctionnelle, intégrant les concepts avancés de la programmation orientée objet, de la conteneurisation et de l'assurance qualité logicielle.

L'approche modulaire adoptée facilite la maintenance et l'évolution future du système (par exemple, l'ajout de nouveaux types de capteurs). L'utilisation combinée de Docker

et de **MongoDB** s'est avérée particulièrement pertinente pour gérer efficacement des flux de données dans un environnement isolé et reproductible, répondant ainsi aux exigences techniques et scientifiques du laboratoire LARI.