TP : Gestion des données pharmaceutiques avec MongoDB et Neo4j

1 Introduction

Ce projet vise à organiser et modéliser des données pharmaceutiques à l'aide de deux bases de données NoSQL : **MongoDB** pour stocker les propriétés des médicaments et **Neo4j** pour représenter leurs relations avec les maladies, symptômes et effets secondaires.

2 Étape 1 : Schéma JSON dans MongoDB

Document Médicament

```
{
  "_id": "med1",
  "nom": "Paractamol",
  "forme": "Comprim",
  "laboratoire": "Sanofi",
  "composition": ["Paractamol"],
  "indications": ["Douleurs", "Fivre"],
  "contre_indications": ["Allergie au paractamol", "Insuffisance hpatique"],
  "effets_secondaires": ["Nauses", "ruptions cutanes", "Lsions du foie  forte dose"],
  "posologie": {
    "adulte": {
      "dose": "500mg",
      "frquence": "toutes les 6 heures",
      "voie": "orale"
    },
    "enfant": {
      "dose": "250mg",
      "frquence": "toutes les 8 heures",
      "ge_minimum": "6 ans"
    }
  },
  "stock_disponible": 320,
  "prix_unitaire": 1.2,
  "date_expiration": "2026-06-01"
}
```

Document Maladie

```
{
  "_id": "maladie1",
  "nom": "Diabte de type 2",
  "description": "Le diabte de type 2 est une maladie chronique...",
  "symptmes": ["Soif excessive", "Urination frquente", "Fatigue"],
  "facteurs_risque": ["Surpoids", "Sdentarit"],
  "complications": ["Maladies cardiovasculaires", "Pied diabtique"],
  "maladies_associees": ["Obsit", "Hypertension"]
}
```

3 Étape 2 :Connexion aux bases de donnees

Insertion dans MongoDB

```
from pymongo import MongoClient

client = MongoClient("mongodb+srv://admin:admin1234@cluster0.ilrrjdm.mongodb.net/?
    retryWrites=true&w=majority")
db = client["pharma"]

medicaments = db["medicaments"]
maladies = db["maladies"]

medicament_doc = { "_id": "med1", "nom": "Paractamol", ... }
maladie_doc = { "_id": "maladie1", "nom": "Diabte de type 2", ... }

medicaments.insert_one(medicament_doc)
maladies.insert_one(maladie_doc)
```

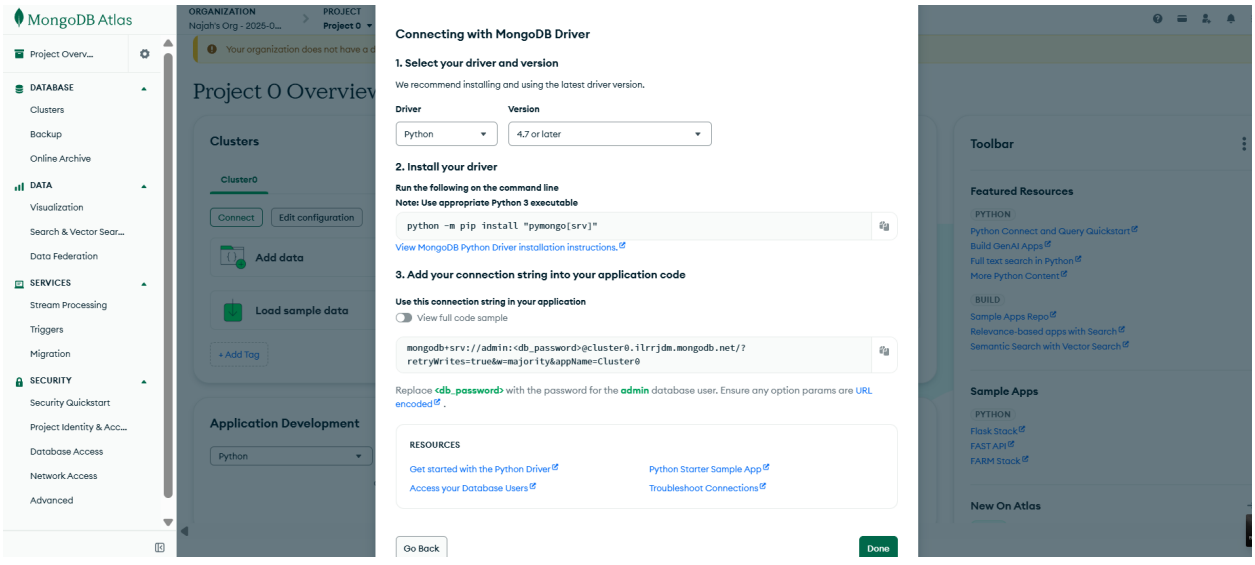

FIGURE 1 – MongoDB Parametres Connexion

Connexion à Neo4j et création des nœuds

```
from py2neo import Graph, Node, Relationship

graph = Graph("neo4j+s://e43ed943.databases.neo4j.io",
              auth=("neo4j", "X_x48MHBDBbdQVoxvBmG7LaPx2CMAbTw7qH3vJ0Xuz4"))

med_node = Node("Medicament", id="med1", nom="Paractamol")
mal_node = Node("Maladie", id="maladie1", nom="Diabte␣de␣type␣2")
graph.merge(med_node, "Medicament", "id")
graph.merge(mal_node, "Maladie", "id")
graph.create(Relationship(med_node, "TRAITE", mal_node))
```

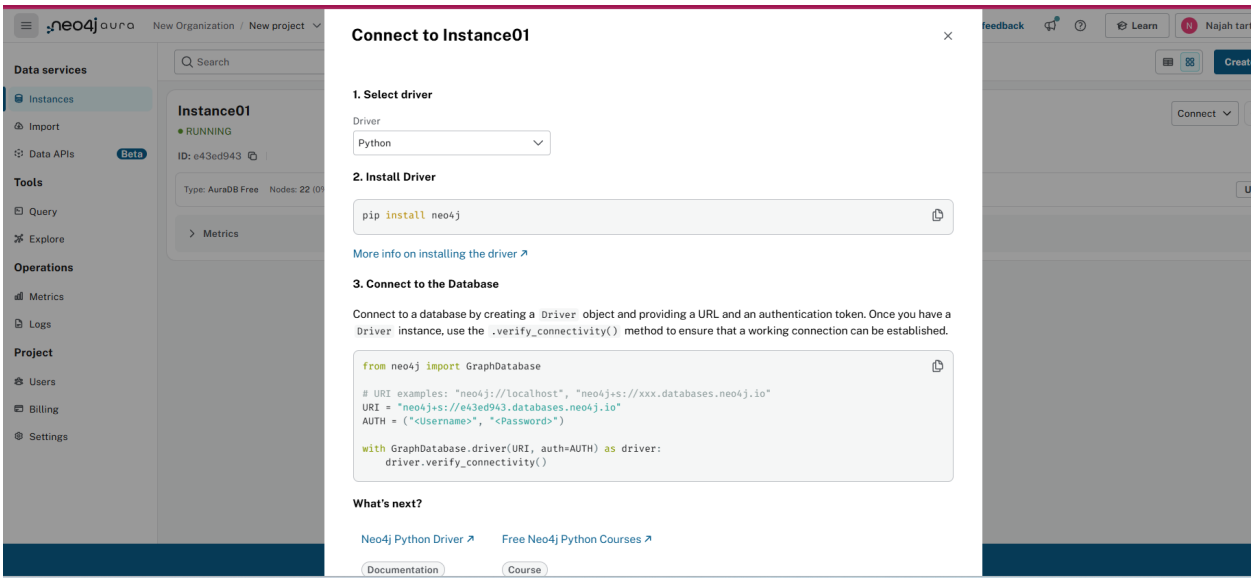

FIGURE 2 – Neo4j Parametres Connexion

Création des nœuds et relations

```
med_node = Node("Medicament", id="med1", nom="Paractamol")
mal_node = Node("Maladie", id="maladie1", nom="Diabte␣de␣type␣2")
graph.create(Relationship(med_node, "TRAITE", mal_node))
```

Ajout des propriétés

```
for ind in ["Douleurs", "Fivre"]:
    n = Node("Indication", nom=ind)
    graph.create(Relationship(med_node, "INDIQUE_POUR", n))
```

4 Requêtes Cypher en Python

Étape 3 : Requête - Médicament par maladie

```
MATCH (m:Medicament)-[:TRAITE]->(mal:Maladie {nom: "..."})
RETURN m.nom AS medicament
```

Étape 4 : Requête - Médicaments qui traitent un symptôme donné

```
MATCH (m:Medicament)-[:TRAITE]->(mal:Maladie)-[:PRESENTE_SYMPTOME]->(s:Symptome {nom:
    $nom_symptome})
RETURN DISTINCT m.nom AS medicament
```

Étape 5 :Maladies traitées par un médicament donné

```
MATCH (m:Medicament {nom: $nom_medicament})-[:TRAITE]->(mal:Maladie)
RETURN mal.nom AS maladie
```

Étape 6 : Requête - Effets secondaires

```
MATCH (m:Medicament {nom: $nom_medicament})-[:CAUSE]->(e:EffetSecondaire)
RETURN e.nom AS effet
```

Étape 7 : Requête - Identifier les interactions potentielles entre médicaments prescrits pour une même maladie

```
MATCH (m1:Medicament {nom: $nom_medicament})-[:TRAITE]->(mal:Maladie {nom: $nom_maladie})
    ,(m2:Medicament)-[:TRAITE]->(mal)
WHERE m1.nom <> m2.nom
RETURN DISTINCT m2.nom AS interaction
```

Étape 8 :Identifier les interactions potentielles pour un médicament prescrit à cause d'un symptôme donné

```
MATCH (m1:Medicament {nom: $nom_medicament})-[:TRAITE]->(mal:Maladie)-[:PRESENTE_SYMPTOME
    ]-(s:Symptome {nom: $nom_symptome}),
(m2:Medicament)-[:TRAITE]->(mal)
WHERE m1.nom <> m2.nom
RETURN DISTINCT m2.nom AS interaction
```

Étape 9 :Médicaments alternatifs pour une maladie

```
MATCH (m:Medicament)-[:TRAITE]->(mal:Maladie {nom: $nom_maladie})
""" + ("""
WHERE m.nom <> $exclure_medicament
""" if exclure_medicament else "") + """
RETURN m.nom AS alternatif
```

5 Étape 10 : Visualisation avec NetworkX

```
import networkx as nx
import matplotlib.pyplot as plt

def visualiser_maladie_et_medicaments(nom_maladie):
    query = """
    MATCH (m:Medicament)-[r:TRAITE]->(mal:Maladie {nom: $nom_maladie})
    OPTIONAL MATCH (mal)-[:PRESENTE_SYMPTOME]->(s:Symptome)
    OPTIONAL MATCH (mal)-[:A_COMME_FACTEUR_RISQUE]->(f:FacteurRisque)
    OPTIONAL MATCH (mal)-[:PEUT_CAUSER]->(c:Complication)
    RETURN m.nom AS medicament, s.nom AS symptome, f.nom AS facteur, c.nom AS
        complication
    """
    data = graph.run(query, nom_maladie=nom_maladie).data()

    G = nx.Graph()
    maladie_label = nom_maladie
    G.add_node(maladie_label, color="lightgreen", type="Maladie")

    for row in data:
        if row["medicament"]:
            G.add_node(row["medicament"], color="lightblue", type="Medicament")
            G.add_edge(row["medicament"], maladie_label, label="TRAITE")
        if row["symptome"]:
            G.add_node(row["symptome"], color="orange", type="Symptome")
            G.add_edge(maladie_label, row["symptome"], label="SYMPTOME")
        if row["facteur"]:
            G.add_node(row["facteur"], color="yellow", type="Facteur")
            G.add_edge(maladie_label, row["facteur"], label="FACTEUR")
        if row["complication"]:
            G.add_node(row["complication"], color="red", type="Complication")
            G.add_edge(maladie_label, row["complication"], label="COMPLICATION")

    # Extraire les attributs
    pos = nx.spring_layout(G)
    colors = [G.nodes[n].get("color", "gray") for n in G.nodes()]
    edge_labels = nx.get_edge_attributes(G, 'label')

    # Dessiner
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_color=colors, node_size=2000, font_size=10)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
```

```
    plt.title(f"Visualisation␣des␣relations␣pour␣:␣{nom_maladie}")
    plt.show()

# Exemple dappel
visualiser_maladie_et_medicaments("Diabte␣de␣type␣2")
```

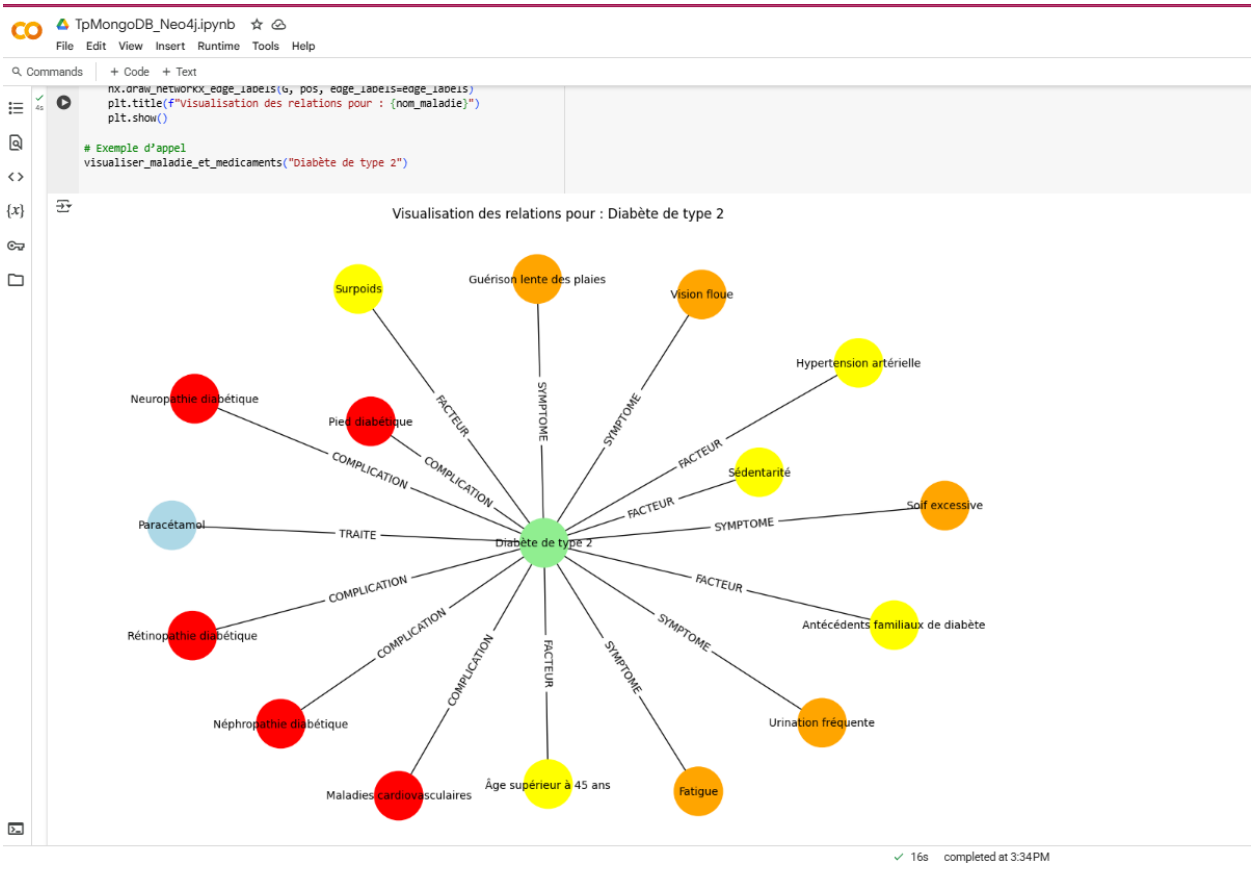

FIGURE 3 – Visualisation du graphe

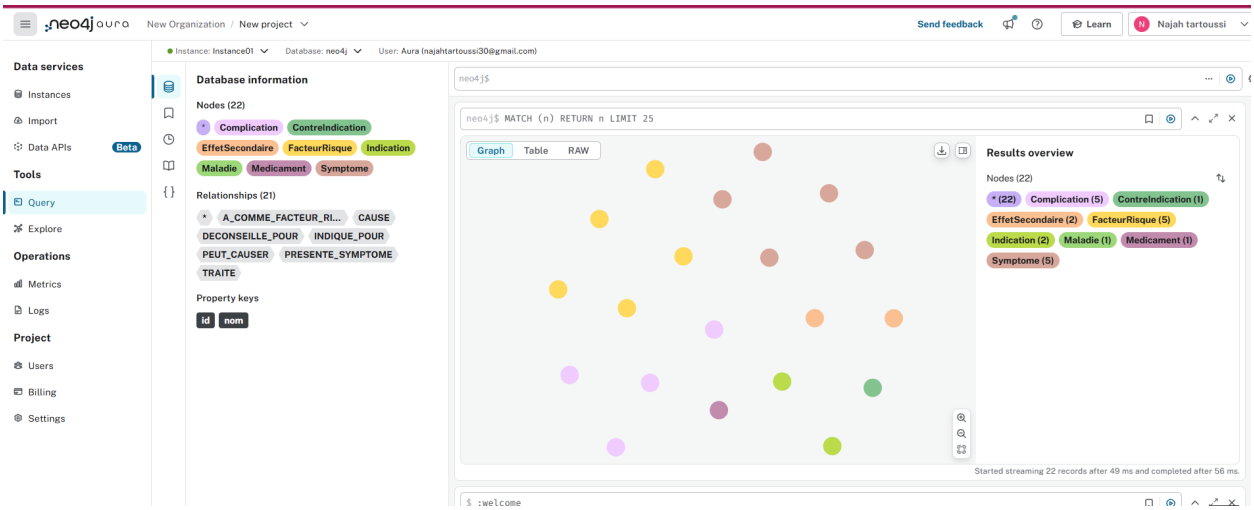

FIGURE 4 – Visualisation du graphe dans Ne04j aura

6 Conclusion

Cette application montre la complémentarité de MongoDB et Neo4j pour modéliser un système riche d'informations médicales. Nous avons pu gérer à la fois les propriétés internes (MongoDB) et les relations complexes (Neo4j) tout en assurant des requêtes efficaces et une visualisation pertinente.