Rapport SAE 5.02 NOSQL Mongo DB

Professeur encadrant :

Mouna Kamel - Institut Universitaire de Technologie de Perpignan à Carcassonne

Etudiant :

Mouhcine Maimouni et Thomas Almodovar : SD3ème année - Institut Universitaire de Technologie de Perpignan à Carcassonne

Année scolaire 2023/ 2024 à Carcassonne

SOMMAIRE

Table des matières

Présentation du projet

Source des données : Nous imaginons que les données nous sont envoyées chaque mois au même format.

Voici les tables :

- Actor, qui liste tous les acteurs
- Film, qui liste tous les films
- Film_actor, qui liste quels acteurs jouent dans quel film

FILM(film_id, title, description, language_id, original_language_id)
ACTOR(actor_id, first_name, last_name)
FILM_ACTOR(actor_id, film_id)

Mission 1. Créer 5 bases de données postgres (pagila1, pagila2, ..., pagila5) selon ce même schéma. Vous avez à disposition :
- Le script sql pour créer les bases de données
- Les fichiers csv permettant d'alimenter les relations de ces bases de données

Mission 2. À partir d'une de ces bases de données, proposer les différentes modélisations orientées document que vous identifiez.

Mission 3. Convertir la base de données relationnelles pagila1 en une base de données au format JSON. Créer une base de données MongoDB et importer ces données :
a. Sous Postgres
b. En python

Mission 4. Ecrire un script python qui permette d'insérer automatiquement dans cette base de données MongoDB, les données issues des bases de données relationnelles pagila2, ..., pagila5, ainsi que les bases de données de même schéma à venir.

Mission 5. Développer une interface graphique en python permettant à un utilisateur de saisir les informations constituant un document (selon le modèle que vous aurez choisi). Les informations saisies seront automatiquement stockées dans un format JSON puis insérées dans la base de données MongoDB.

Mission 1

Insertion des différents pagila de manière semi-automatique, de façon à ce que l'utilisateur puisse le faire tous les mois.

Le programme demande à l'utilisateur quel pagila il veut traiter, si le pagila n'existe pas la base de données est créer, sinon il demande à l'utilisateur s'il veut écraser la base de données.

Si l'utilisateur écrase la base de données les données du pagila choisi sont supprimer de la BDD Postgres et sont réinsérées.

```
import psycopg2
from psycopg2 import sql
import pandas as pd
import time



i = input("Quel pagila voulez-vous modifier/créer ? \n")

try:
    host = "10.11.159.10"
    database = "2023_Maimouni_Almodovar_pagila" + i
    user = "admindbetu"
    password = "admindbetu"

    # Connexion à la base de données
    conn = psycopg2.connect(host=host, database=database, user=user,
password=password)
    cursor = conn.cursor()
except psycopg2.Error as e:
    print(f"Erreur lors de la connexion à la base de données : {e}")
    a = input(f"Pagila{i} n'existe pas, voulez-vous créer la base de données ? \n
Répondez O / N \n")

    if a == "N":
        print("Annulation du programme...")
    elif a != 'N' and a != "O":
        print("Réponse non comprise \n Répondez O / N")
    else:
        # Connexion au serveur postgres
        conn = psycopg2.connect(dbname="postgres", host=host, user=user,
password=password)
        conn.autocommit = True
```

```
        cursor = conn.cursor()

        # Remplacez "nouvelle_base_de_donnees" par le nom que vous souhaitez donner
à votre base de données
        nom_nouvelle_base_de_donnees = "2023_Maimouni_Almodovar_pagila" + i

        # Utiliser l'objet sql.Identifier pour s'assurer que le nom de la base de
données est correctement formaté
        nom_nouvelle_base_de_donnees_identifie =
sql.Identifier(nom_nouvelle_base_de_donnees)

        # Exécuter la requête SQL CREATE DATABASE
        requete = sql.SQL("CREATE DATABASE
{}").format(nom_nouvelle_base_de_donnees_identifie)
        cursor.execute(requete)

        # Fermer le curseur et la connexion
        cursor.close()
        conn.close()
        time.sleep(5)
        print(f"Base de données {nom_nouvelle_base_de_donnees} créée avec succès")

        # Reconnecter à la nouvelle base de données
        database = "2023_Maimouni_Almodovar_pagila" + i
        conn = psycopg2.connect(host=host, database=database, user=user,
password=password)
        cursor = conn.cursor()

        # Création des tables et insertion de données
        cursor.execute("""
            CREATE TABLE public.film (
                film_id integer NOT NULL,
                title text NOT NULL,
                description text,
                language_id integer NOT NULL,
                original_language_id integer
            );
            ALTER TABLE public.film OWNER TO postgres;

            CREATE TABLE public.actor (
                actor_id integer NOT NULL,
                first_name text NOT NULL,
                last_name text NOT NULL
            );
            ALTER TABLE public.actor OWNER TO postgres;

            CREATE TABLE public.film_actor (
                actor_id integer NOT NULL,
```

```
                film_id integer NOT NULL
            );
            ALTER TABLE public.film_actor OWNER TO postgres;
        """)

        # Valider les changements
        conn.commit()

# Supprimer les données existantes si l'utilisateur le souhaite
r = input(f"Voulez-vous écraser la base de données pagila{i} ? Répondez O / N \n")
if r == "N":
    print("Annulation du programme...")
elif r != 'N' and r != "O":
    print("Réponse non comprise \n Répondez O / N")
else:
    cursor.execute("DELETE FROM actor")
    cursor.execute("DELETE FROM film")
    cursor.execute("DELETE FROM film_actor")

    # Valider les changements
    conn.commit()

    # Chargement des données depuis les fichiers CSV
    actor = pd.read_csv('actor'+i+'.csv', sep=',')
    film_actor = pd.read_csv('film_actor'+i+'.csv', sep=',')
    film = pd.read_csv('film'+i+'.csv', sep=',')

    # Insertion des données dans les tables
    for ligne in range(0, len(actor)):
        actor_id = actor["actor_id"][ligne]
        first_name = actor["first_name"][ligne]
        last_name = actor["last_name"][ligne]
        requete = f"INSERT INTO actor VALUES ({actor_id}, '{first_name}',
'{last_name}')"
        cursor.execute(requete)
        conn.commit()

    for ligne in range(0, len(film_actor)):
        actor_id = film_actor["actor_id"][ligne]
        film_id = film_actor["film_id"][ligne]
        requete = f"INSERT INTO film_actor VALUES ({actor_id}, {film_id})"
        cursor.execute(requete)
        conn.commit()

    for ligne in range(0, len(film)):
        film_id = film["film_id"][ligne]
        title = film["title"][ligne]
        description = film["description"][ligne]
```

```
        language_id = film["language_id"][ligne]
        original_language_id = film["original_language_id"][ligne]

        if pd.isna(original_language_id):
            original_language_id = 'NULL'

        requete = f"INSERT INTO film VALUES ({film_id}, '{title}', '{description}',
{language_id}, {original_language_id})"
        cursor.execute(requete)
        conn.commit()

    print(f"Données insérées dans {database} avec succès.")

# Fermer la connexion à la fin du programme
conn.close()
```

Mission 2

Les deux modélisations auxquelles nous avons pensée sont :

Modélisation 1 :

```
film_data = {
    "_id": film_id,
    "title": title,
    "description": description,
    "language_id": language_id,
    "original_language_id": original_language_id,
    "actors": {
        "actor_id": actor_id,
        "first_name": first_name,
        "last_name": last_name
    }, {
        "actor_id": actor_id,
        "first_name": first_name,
        "last_name": last_name
    }, …

}
```

Modélisation 2 :

```
actor_and_film_data = {
    "_id": actor_id,
    "first_name": first_name,
    "last_name": last_name,
    "films": {
        "film_id": film_id,
        "title": title,
        "description": description,
        "language_id": language_id,
        "original_language_id": original_language_id
    }, {
        "film_id": film_id,
        "title": title,
        "description": description,
        "language_id": language_id,
        "original_language_id": original_language_id
    }, …

}
```

Mission 3 et Mission 4

Insertion des différents pagila de manière semi-automatique, de Postgres à Mongo DB.
De façon à ce que l'utilisateur puisse exécuter le programme chaque mois.

Le programme demande à l'utilisateur quel pagila il veut traiter (dans Postgres), pour mettre à jour les bases de données Mongo DB (une avec la modélisation 1 et une deuxième avec la modélisation 2)

La mise à jour ce base sur les _ID des objets MongoDB (film ou actor).
De plus si un ID existe déjà, le programme ajoute les données manquantes à l'objet, exemple : si un acteur joue dans un film et qu'il est déjà dans la base de données, le film va être ajouté à la liste des films dans lesquels il a joué.

Sous Postgresql :

Nous permet d'avoir un fichier Json directement créer par Postgresql

```
CREATE OR REPLACE FUNCTION export_to_json()
RETURNS VOID AS $$
DECLARE
    output_file TEXT := '. /resultats.json';
BEGIN
    EXECUTE 'COPY (
        SELECT
            jsonb_pretty(jsonb_build_object(
                "film_id", film.film_id,
                "title", film.title,
                "description", film.description,
                "language_id", film.language_id,
                "original_language_id", film.original_language_id,
                "actors", jsonb_agg(jsonb_build_object(
                    "actor_id", actor.actor_id,
                    "first_name", actor.first_name,
                    "last_name", actor.last_name
                ))
            ))
        FROM public.film
        LEFT JOIN public.film_actor ON film.film_id = film_actor.film_id
        LEFT JOIN public.actor ON film_actor.actor_id = actor.actor_id
        GROUP BY film.film_id
    ) TO ''' || output_file || ''';
END;
$$ LANGUAGE plpgsql;

-- Appelez la fonction pour exécuter la requête et écrire dans un fichier
SELECT export_to_json();
```

Sous Python

```python
import psycopg2
from psycopg2 import sql
import pandas as pd
import json
from pymongo import MongoClient


i =input("Quel pagila faut-il mettre à jour :\n")


# Informations de connexion à la base de données
db_config = {
    'host': '10.11.159.10',
    'database': '2023_Maimouni_Almodovar_pagila'+str(i),
    'user': 'admindbetu',
```

```
    'password': 'admindbetu',
    'port': '5432',  # Port par défaut pour PostgreSQL
}

# Connexion à la base de données
connection = psycopg2.connect(**db_config)
cursor = connection.cursor()

# Exécutez la requête SQL pour obtenir les résultats
cursor.execute("SELECT film_id,title,description,language_id,original_language_id
FROM film;")
films_data = cursor.fetchall()

# Créez un DataFrame pandas à partir des résultats de la requête
columns = ['film_id','title','description','language_id','original_language_id']
df_film = pd.DataFrame(films_data, columns=columns)

# Affichez le DataFrame
print(df_film)

#_____________________________________________________________________________
_________________________________

# Exécutez la requête SQL pour obtenir les résultats
cursor.execute("SELECT actor_id,film_id FROM film_actor;")
films_data = cursor.fetchall()

# Créez un DataFrame pandas à partir des résultats de la requête
columns = ['actor_id','film_id']
df_film_actor = pd.DataFrame(films_data, columns=columns)

# Affichez le DataFrame
print(df_film_actor)


#_____________________________________________________________________________
_________________________________

# Exécutez la requête SQL pour obtenir les résultats
cursor.execute("SELECT actor_id,first_name, last_name FROM actor;")
films_data = cursor.fetchall()

# Créez un DataFrame pandas à partir des résultats de la requête
columns = ['actor_id','first_name', 'last_name']
df_actor = pd.DataFrame(films_data, columns=columns)

# Affichez le DataFrame
print(df_actor)
```

```
#_____
_____


# Suppose you have three dataframes named df_film, df_actor, and df_film_actor

# Remplacer les valeurs NaN par des chaînes vides dans les DataFrames
df_film = df_film.fillna('')
df_actor = df_actor.fillna('')
df_film_actor = df_film_actor.fillna('')

# Create JSON for actors
json_actors = df_actor.to_json(orient='records')

# Manipulation du JSON pour correspondre au schéma spécifié
films_data = []
for film_data in
pd.read_json(df_film.to_json(orient='records')).to_dict(orient='records'):
    # Get actors associated with the film
    actors_in_film = df_film_actor[df_film_actor['film_id'] ==
film_data['film_id']].to_dict(orient='records')
    actors_info = []
    for actor_in_film in actors_in_film:
        actor_info = df_actor[df_actor['actor_id'] ==
actor_in_film['actor_id']].to_dict(orient='records')[0]
        actors_info.append({
            "actor_id": actor_info["actor_id"],
            "first_name": actor_info["first_name"],
            "last_name": actor_info["last_name"]
        })

    films_data.append({
        "_id": film_data["film_id"],
        "title": film_data["title"],
        "description": film_data["description"],
        "language_id": film_data["language_id"],
        "original_language_id": film_data["original_language_id"],
        "actors": actors_info
    })

# Affichage du JSON final pour les films
#print(films_data)
json_film = films_data

# Exporter les données au format JSON dans un fichier
```

```
with open("film_data.json", 'w') as fichier_json:
    json.dump(films_data, fichier_json, indent=2)

#_____________________________________________________________________________
_____________________________
json_actors = df_actor.to_json(orient='records')

# Manipulation du JSON pour correspondre au schéma spécifié
actors_data = []
for actor_data in pd.read_json(json_actors).to_dict(orient='records'):
    films_data = []
    for film_id in df_film_actor[df_film_actor['actor_id'] ==
actor_data['actor_id']]['film_id']:
        film_info = df_film[df_film['film_id'] ==
film_id].to_dict(orient='records')[0]
        films_data.append({
            "film_id": film_info["film_id"],
            "title": film_info["title"],
            "description": film_info["description"],
            "language_id": film_info["language_id"],
            "original_language_id": film_info["original_language_id"]
        })

    actors_data.append({
        "_id": actor_data["actor_id"],
        "first_name": actor_data["first_name"],
        "last_name": actor_data["last_name"],
        "films": films_data
    })

# Affichage du JSON final pour les acteurs
#print(actors_data)

# Exporter les données au format JSON dans un fichier
with open("actor_data.json", 'w') as fichier_json:
    json.dump(actors_data, fichier_json)



# Fermez la connexion à la base de données
cursor.close()
connection.close()
print("Connexion à la base de données fermée.")

#_____________________________________________________________________________
_____________________________
```

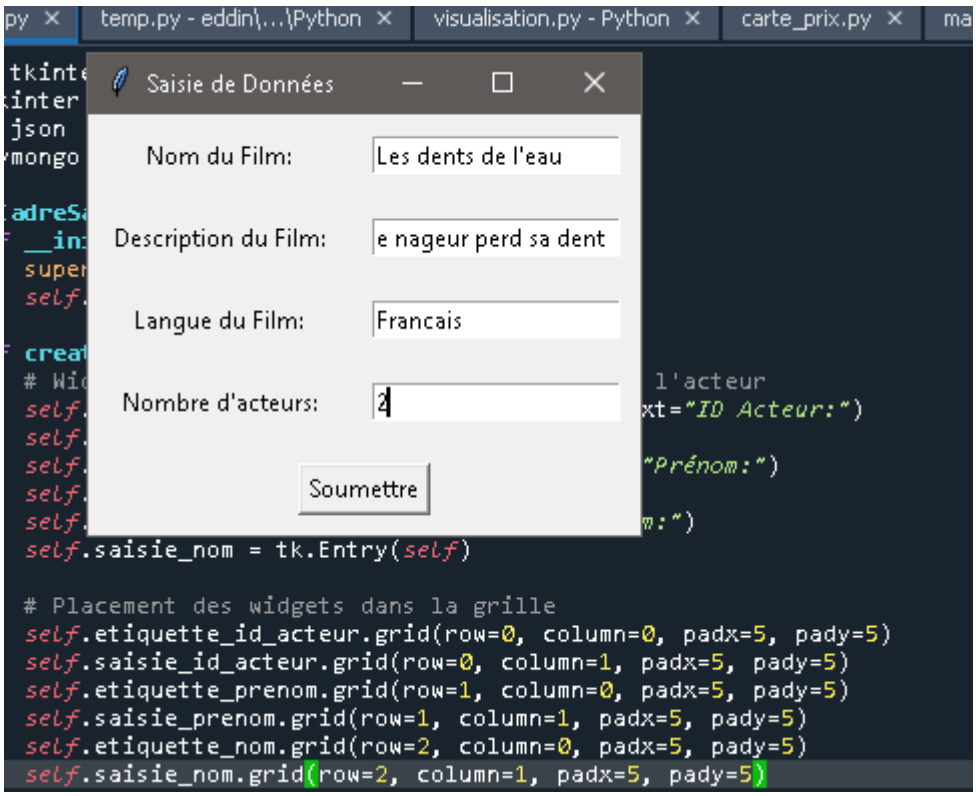
```
# Connexion à la base de données MongoDB
client = MongoClient('localhost', 27017)  # Assurez-vous de spécifier la bonne
adresse et le bon port
database = client['film_actor']
collection = database['film']

# Récupérer les _id des documents dans la collection
liste_ids = collection.find({}, {"id": 1})

#Liste des _id présent dans la base mongoDB
liste_mongo_id = []
for i in liste_ids:
    liste_mongo_id.append(i["_id"])


#Création d'une liste, on stocke toutes les données qui ne sont pas dans la base
mongodb
#On compare les id
#La liste créer sera utilisée pour insérer les données
id_a_inserer_film = []

for i in range(len(json_film)):
    if json_film[i]['_id']  not in liste_mongo_id:
        id_a_inserer_film.append(json_film[i])


#Si la liste des données à insérer est vide, on affiche un message adapté, sinon on
insère les données
if len(id_a_inserer_film) != 0:
    print("Nombre éléments json film: "+str(len(json_film)))
    # Insérer les données dans la collection
    collection.insert_many(id_a_inserer_film)
else:
    print("Pas de données insérer (Les données dont déjà présentes dans la table)")



#_____________________________________________________________________________
_____________________________




collection = database['actor']

# Récupérer les _id des documents dans la collection
liste_ids = collection.find({}, {"id": 1})

#Liste des _id présent dans la base mongoDB
```

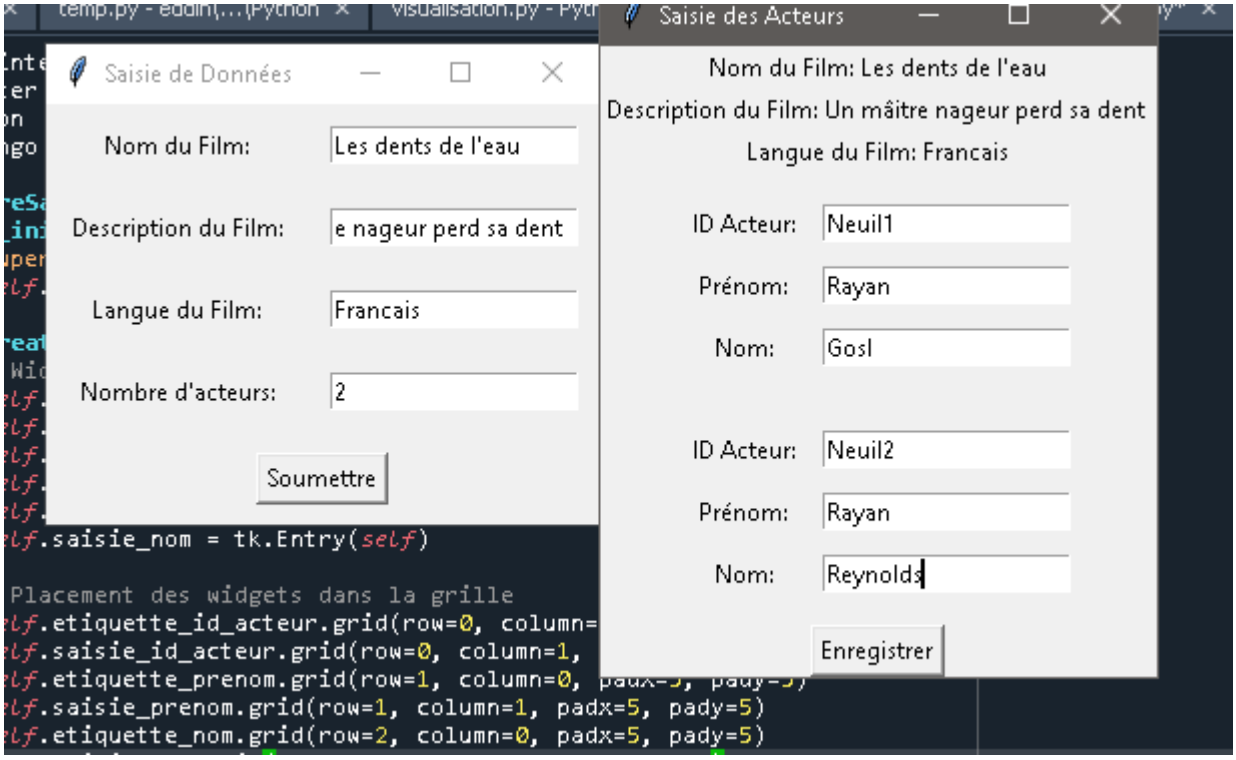
```
liste_mongo_id = []
for i in liste_ids:
    liste_mongo_id.append(i["_id"])


#Création d'une liste, on stocke toutes les données qui ne sont pas dans la base
mongodb
#On compare les id
#La liste créer sera utilisée pour insérer les données
id_a_inserer_actors = []

for i in range(len(actors_data)):
    if actors_data[i]['_id']  not in liste_mongo_id:
        id_a_inserer_actors.append(actors_data[i])
    else:
        #Créer une liste qui renseigne les films où l'acteur a joué qui sont stocké
dans la base Mongo
        actor_id = collection.find_one({'_id': actors_data[i]['_id']})
        print(actor_id)
        nb_film = len(actor_id["films"])
        id_film_joue_mongo =  []
        for id_film in range(nb_film):
            id_film_joue_mongo = actor_id["films"]




        #Créer la liste de ses nouveaux films
        nb_film_json = len(actors_data[i]["films"])
        id_film_joue_json =  []
        for id_film in range(nb_film_json):
            film_joue_json = actors_data[i]["films"][id_film]
            id_film_joue_json.append(film_joue_json)



        test = []
        test = id_film_joue_json
        test.extend(id_film_joue_mongo)


        # Supprimer les doublons en utilisant un ensemble (set)
        unique_films = []
        seen_ids = set()

        for film in test:
            film_id = film['film_id']
            if film_id not in seen_ids:
```

```
                unique_films.append(film)
                seen_ids.add(film_id)

        # Maintenant, unique_films contient la liste sans doublons
        print(unique_films)




        actors_data[i]["films"] = unique_films

        collection.delete_one({'_id':  actors_data[i]["_id"]})

        id_a_inserer_actors.append(actors_data[i])


#Si la liste des données à insérer est vide, on affiche un message adapté, sinon on
insère les données
if len(id_a_inserer_actors) != 0:
    print("Nombre éléments json actors: "+str(len(actors_data)))
    # Insérer les données dans la collection
    collection.insert_many(id_a_inserer_actors)
else:
    print("Pas de données insérer (Les données dont déjà présentes dans la table)")


# Fermer la connexion à MongoDB
client.close()
```

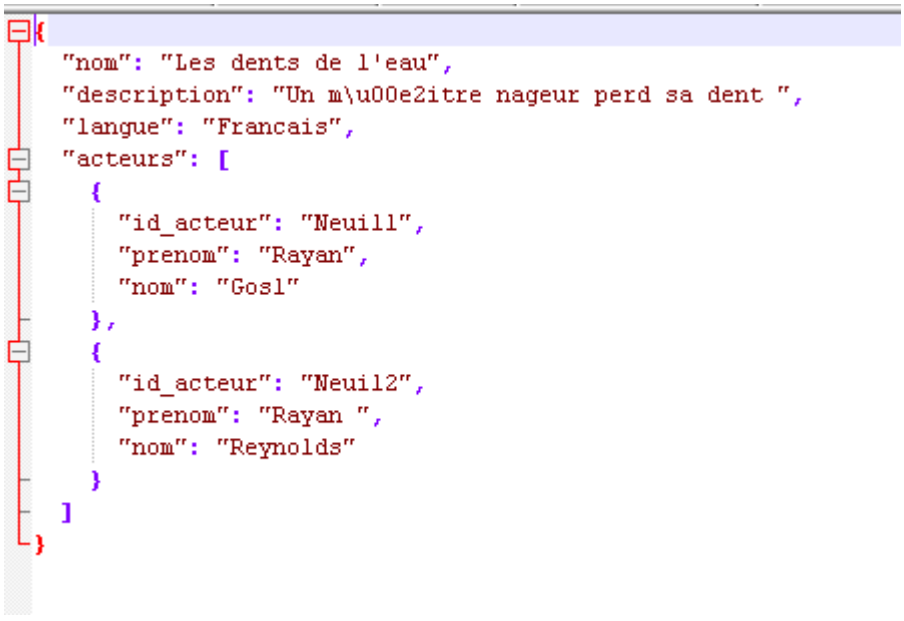
Mission 5

Pour la mission 5, nous avons choisi la bibliothèque ThinkerView en Python afin de créer une interface graphique plus esthétique. L'objectif de cette interface était de permettre à l'utilisateur d'entrer le nom du film, la description et la langue, et de spécifier le nombre d'acteurs qu'il souhaite associer à ce film. Une fois ces informations saisies, un fichier JSON est automatiquement généré avec ces données et intégré dans la base de données MongoDB.





Rendu :



Voici le code Python correspondant :

```
import tkinter as tk
from tkinter import messagebox
import json
from pymongo import MongoClient

class CadreSaisieActeur(tk.Frame):
    def __init__(self, master=None):
```

```
        super().__init__(master)
        self.create_widgets()

    def create_widgets(self):
        # Widgets pour la saisie des informations de l'acteur
        self.etiquette_id_acteur = tk.Label(self, text="ID Acteur:")
        self.saisie_id_acteur = tk.Entry(self)
        self.etiquette_prenom = tk.Label(self, text="Prénom:")
        self.saisie_prenom = tk.Entry(self)
        self.etiquette_nom = tk.Label(self, text="Nom:")
        self.saisie_nom = tk.Entry(self)

        # Placement des widgets dans la grille
        self.etiquette_id_acteur.grid(row=0, column=0, padx=5, pady=5)
        self.saisie_id_acteur.grid(row=0, column=1, padx=5, pady=5)
        self.etiquette_prenom.grid(row=1, column=0, padx=5, pady=5)
        self.saisie_prenom.grid(row=1, column=1, padx=5, pady=5)
        self.etiquette_nom.grid(row=2, column=0, padx=5, pady=5)
        self.saisie_nom.grid(row=2, column=1, padx=5, pady=5)

class ApplicationPrincipale:
    def __init__(self, maitre):
        self.maitre = maitre
        self.cadres_saisie_acteur = []  # Liste pour stocker les cadres de saisie d'acteur
        self.creer_widgets()

    def creer_widgets(self):
        # Widgets pour la saisie des informations du film
        self.etiquette_nom_film = tk.Label(self.maitre, text="Nom du Film:")
        self.saisie_nom_film = tk.Entry(self.maitre)
        self.etiquette_description = tk.Label(self.maitre, text="Description du Film:")
        self.saisie_description = tk.Entry(self.maitre)
        self.etiquette_langue = tk.Label(self.maitre, text="Langue du Film:")
        self.saisie_langue = tk.Entry(self.maitre)
        self.etiquette_nb_acteurs = tk.Label(self.maitre, text="Nombre d'acteurs:")
        self.saisie_nb_acteurs = tk.Entry(self.maitre)
        self.bouton_soumettre = tk.Button(self.maitre, text="Soumettre",
command=self.afficher_interface_saisie_acteur)

        # Placement des widgets dans la grille
        self.etiquette_nom_film.grid(row=0, column=0, padx=10, pady=10)
        self.saisie_nom_film.grid(row=0, column=1, padx=10, pady=10)
        self.etiquette_description.grid(row=1, column=0, padx=10, pady=10)
        self.saisie_description.grid(row=1, column=1, padx=10, pady=10)
        self.etiquette_langue.grid(row=2, column=0, padx=10, pady=10)
        self.saisie_langue.grid(row=2, column=1, padx=10, pady=10)
        self.etiquette_nb_acteurs.grid(row=3, column=0, padx=10, pady=10)
        self.saisie_nb_acteurs.grid(row=3, column=1, padx=10, pady=10)
        self.bouton_soumettre.grid(row=4, column=0, columnspan=2, pady=10)

    def afficher_interface_saisie_acteur(self):
        try:
            # Récupérer le nombre d'acteurs depuis l'entrée utilisateur
            nb_acteurs = int(self.saisie_nb_acteurs.get())
```

```
        # Créer une fenêtre de saisie pour les acteurs
        interface_saisie_acteur = tk.Toplevel(self.maitre)
        interface_saisie_acteur.title("Saisie des Acteurs")

        # Créer un dictionnaire pour stocker les données du film
        donnees_film = {
            "nom": self.saisie_nom_film.get(),
            "description": self.saisie_description.get(),
            "langue": self.saisie_langue.get(),
            "acteurs": []
        }

        # Afficher les informations du film dans la fenêtre de saisie d'acteur
        tk.Label(interface_saisie_acteur, text=f"Nom du Film: {donnees_film['nom']}").pack()
        tk.Label(interface_saisie_acteur, text=f"Description du Film: {donnees_film['description']}").pack()
        tk.Label(interface_saisie_acteur, text=f"Langue du Film: {donnees_film['langue']}").pack()

        # Réinitialiser la liste des cadres de saisie d'acteur
        self.cadres_saisie_acteur = []

        # Créer les cadres de saisie d'acteur en fonction du nombre d'acteurs
        for i in range(nb_acteurs):
            cadre_saisie_acteur = CadreSaisieActeur(interface_saisie_acteur)
            cadre_saisie_acteur.pack(pady=10)
            self.cadres_saisie_acteur.append(cadre_saisie_acteur)

        # Ajouter un bouton pour enregistrer les données
        tk.Button(interface_saisie_acteur, text="Enregistrer", command=lambda: self.recueillir_et_enregistrer_donnees(donnees_film)).pack()

    except ValueError:
        messagebox.showerror("Erreur", "Veuillez entrer un nombre valide d'acteurs.")

def recueillir_et_enregistrer_donnees(self, donnees_film):
    # Parcourir chaque cadre de saisie d'acteur et collecter les données
    for cadre_saisie_acteur in self.cadres_saisie_acteur:
        donnees_acteur = {
            "id_acteur": cadre_saisie_acteur.saisie_id_acteur.get(),
            "prenom": cadre_saisie_acteur.saisie_prenom.get(),
            "nom": cadre_saisie_acteur.saisie_nom.get()
        }
        # Ajouter les données de l'acteur à la liste d'acteurs dans le dictionnaire du film
        donnees_film["acteurs"].append(donnees_acteur)

    # Enregistrez localement dans un fichier JSON
    nom_fichier = f"{donnees_film['nom']}_donnees.json"
    with open(nom_fichier, 'w') as fichier_json:
        json.dump(donnees_film, fichier_json, indent=2)

    # Enregistrez dans MongoDB
    client = MongoClient('localhost', 27017)  # Mettez les paramètres appropriés
    db = client['pagila']
```

```
        collection = db[‘acteur’]
        collection.insert_one(donnees_film)

        # Afficher un message de succès
        messagebox.showinfo("Succès", f"Données enregistrées localement dans {nom_fichier} et dans MongoDB.")

if __name__ == "__main__":
    racine = tk.Tk()
    racine.title("Saisie de Données")

    application = ApplicationPrincipale(racine)

    racine.mainloop()
```