Rapport de Projet

Emotion Playlist Finder

Taibi Mohamed Yassin
Rime Tlimes
Ilyas Boutayb
Narjis Akhayat

Université Internationale de Rabat

January 15, 2025

Contents

Chapter 1

Introduction

1.1 Contexte

La musique a un impact significatif sur notre état émotionnel et mental. Dans une société où les émotions jouent un rôle central, associer l'intelligence artificielle et la musique peut offrir une expérience personnalisée.

L'Emotion Playlist Finder vise à analyser une image pour détecter les émotions de l'utilisateur et générer une playlist musicale correspondante.

1.2 Objectifs du Projet

Ce projet a pour but de :

- Permettre aux utilisateurs de générer des playlists personnalisées en fonction de leur humeur.
- Utiliser des technologies avancées comme la reconnaissance faciale et l'intelligence artificielle pour détecter des émotions.
- Enregistrer les données pour des analyses futures.

1.3 Problématique

Le défi principal est de développer un système qui détecte avec précision les émotions humaines complexes et leur associe des playlists pertinentes, tout en offrant une expérience utilisateur fluide.

Chapter 2

Analyse et Conception

2.1 Identification des besoins fonctionnels

2.1.1 Besoins Utilisateurs

- Télécharger une image via une interface intuitive.
- Afficher l'émotion dominante détectée.
- Accéder à une playlist musicale associée.

2.1.2 Besoins Système

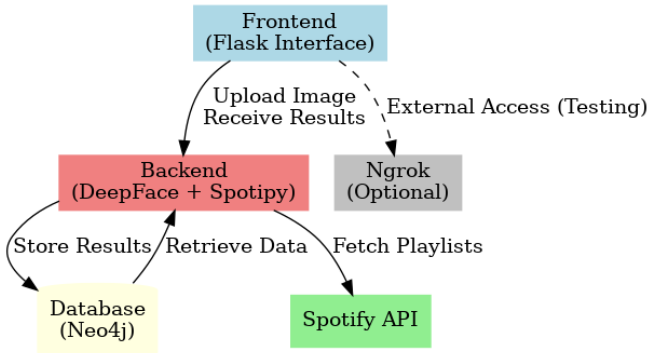
- Analyser une image pour détecter une émotion.
- Intégrer l'API Spotify pour générer des playlists.
- Enregistrer les émotions détectées et les images dans une base de données Neo4j.

2.2 Conception de l'Architecture

L'application suit une architecture client-serveur :

1. Frontend (interface utilisateur) : Permet le téléchargement d'images et affiche les playlists générées.
2. Backend (Flask) :
    - Analyser les émotions via DeepFace.
    - Intégrer l'API Spotify pour la création de playlists.
    - Sauvegarder les données dans Neo4j.

2.3 Diagramme d'Architecture



Chapter 3

Développement

3.1 Technologies et outils utilisés

- Langage : Python.
- Framework : Flask.

- Librairie IA : DeepFace (pour la détection des émotions).
- Base de données : Neo4j.
- API Spotify : Spotipy.
- Frontend : HTML, CSS.

3.2 Fonctionnement global du système

1. L'utilisateur charge une image.
2. Le backend traite l'image pour détecter une émotion via DeepFace.
3. Le système interroge l'API Spotify pour obtenir une playlist liée à cette émotion.
4. Les données (émotion, image) sont sauvegardées dans Neo4j.

3.3 Captures d'écran (Fonctionnement)

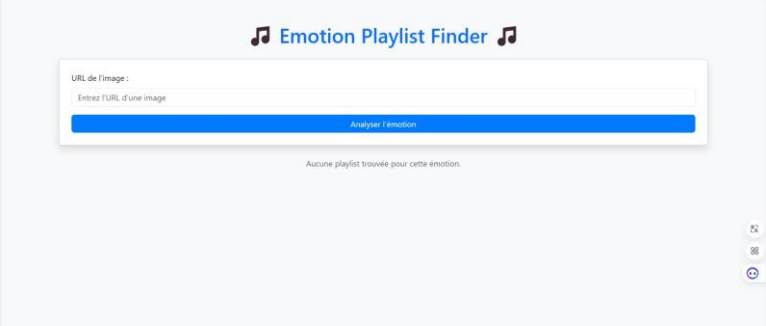


Figure 3.1: Interface utilisateur

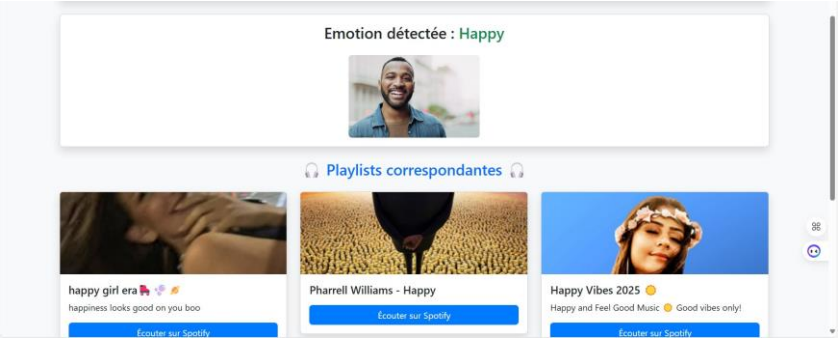


Figure 3.1: Resultat

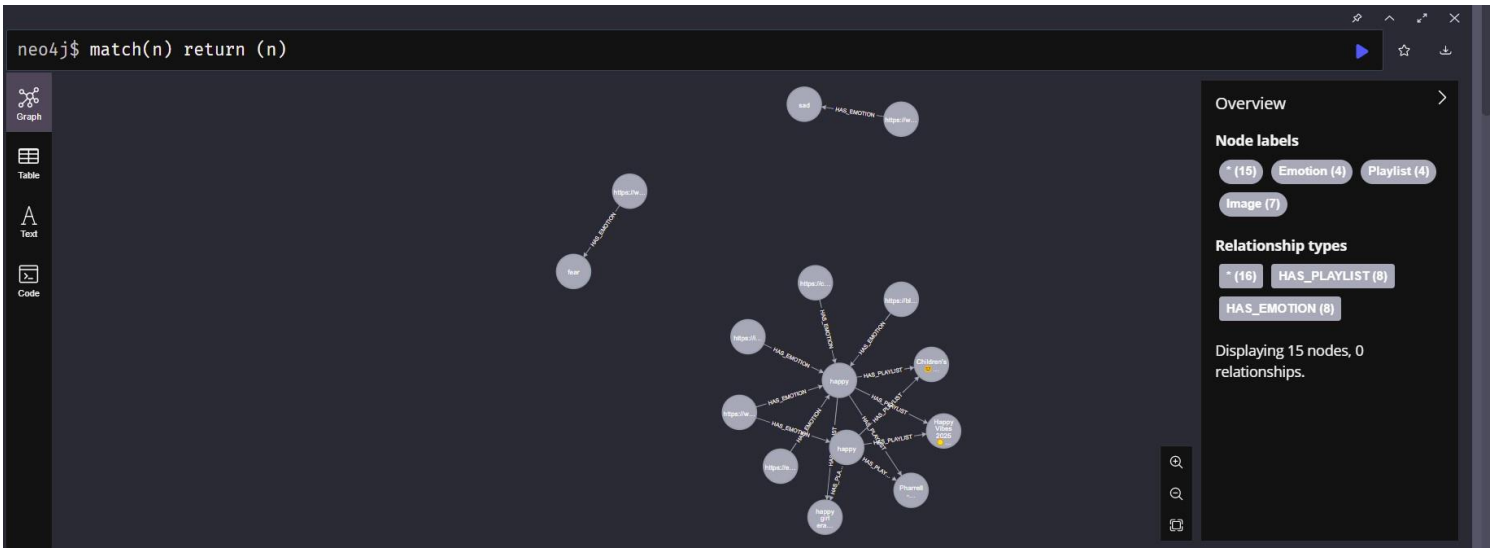


Figure 3.1: Base de Donnée

3.4 Code Source Complet

3.4.1 Backend Flask (app.py)

```
from flask import Flask, render_template, request
from deepface import DeepFace
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from pyngrok import ngrok
from neo4j import GraphDatabase


# Configuration de l'API Spotify client_id = '96a6dc2af02843e99864659e8bb1cfa9'  # Remplacez
par ton client_id Spotify client_secret = '12969d11213f43778dacf9dc85fccbb3'  # Remplacez par
ton client_secret Spotify


client_credentials_manager = SpotifyClientCredentials(client_id=client_id, client_secret=client_secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)


# Configuration de la base de données Neo4j uri =
"bolt://6.tcp.eu.ngrok.io:13886"  # L'URL de ton instance Neo4j
username = "neo4j"  # Ton nom d'utilisateur password =
"emotionDB"  # Ton mot de passe
driver = GraphDatabase.driver(uri, auth=(username, password))


# Initialisation de Flask
app = Flask(__name__)


# Fonction pour détecter l'émotion à partir de l'URL d'une image
def detect_emotion(image_url):    try:
        analysis = DeepFace.analyze(image_url, actions=['emotion'], enforce_detection=False)
if analysis:
            emotion = analysis[0]['dominant_emotion']
            return emotion
else:
            return None
except Exception as e:
        print(f"Erreur de DeepFace: {e}")
return None


# Fonction pour obtenir une playlist Spotify en fonction de l'émotion
def get_spotify_playlist(emotion):    search_query =
emotion.lower()
    results = sp.search(q=search_query, type='playlist', limit=10)


    playlists = []    for playlist in
results['playlists']['items']:        if playlist:
            playlists.append({
'name': playlist['name'],
                'url': playlist['external_urls']['spotify'],
                'description': playlist['description'],
                'image': playlist['images'][0]['url'] if playlist['images'] else None
            })


    return playlists


# Fonction pour sauvegarder l'émotion dans la base de données Neo4j
def save_emotion_to_neo4j(image_url, emotion):    try:        with
driver.session() as session:            # Vérifier si l'image existe déjà
            query = (
                "MATCH (i:Image {url: $image_url}) "
                "RETURN i"
            )
            result = session.run(query, image_url=image_url)


            # Si l'image existe déjà, récupérer l'émotion et établir la relation
if result.single():
                # L'image existe déjà, on peut directement ajouter la relation
query = (
                    "MATCH (i:Image {url: $image_url}), (e:Emotion {name: $emotion}) "
                    "MERGE (i)-[:HAS_EMOTION]->(e)"
                )
                session.run(query, image_url=image_url, emotion=emotion)
print(f"Relation HAS_EMOTION ajoutée pour l'image existante: {image_url}")
else:
                # Si l'image n'existe pas, créer un nouveau nœud pour l'image et l'émotion
query = (
                    "MERGE (i:Image {url: $image_url}) "
                    "MERGE (e:Emotion {name: $emotion}) "
                    "MERGE (i)-[:HAS_EMOTION]->(e)"
                )
                session.run(query, image_url=image_url, emotion=emotion)
print(f"Emotion '{emotion}' sauvegardée pour l'image: {image_url}")    except
Exception as e:        print(f"Erreur lors de la sauvegarde dans Neo4j: {e}")


# Route principale
```

```
@app.route("/", methods=["GET", "POST"])
def home():     emotion = None
    playlists = []
error = None
    image_url = None


    if request.method == "POST":
image_url = request.form.get("image_url")


        # Vérifier si l'URL de l'image est valide
if not image_url.startswith("http"):
            error = "Veuillez entrer une URL valide."
else:
            # Détecter l'émotion            emotion
= detect_emotion(image_url)            if not
emotion:
                error = "Impossible de détecter une émotion. Essayez une autre image."
else:
                # Sauvegarder l'émotion dans Neo4j
                save_emotion_to_neo4j(image_url, emotion)


                # Obtenir la playlist Spotify
playlists = get_spotify_playlist(emotion)


    return render_template("index.html", emotion=emotion, playlists=playlists, image_url=image_url, error=error)


if __name__ == "__main__":
    # Ouvrir un tunnel ngrok sur le port 5000
public_url = ngrok.connect(5000)
    print(" * Ngrok tunnel \"{}\" -> \"http://127.0.0.1:5000\"".format(public_url))


    # Démarrer l'application Flask
app.run(port=5000)
```

3.5 Captures d'écran (Code et Résultats)

Figure 3.3: Exemple de capture d'écran du code

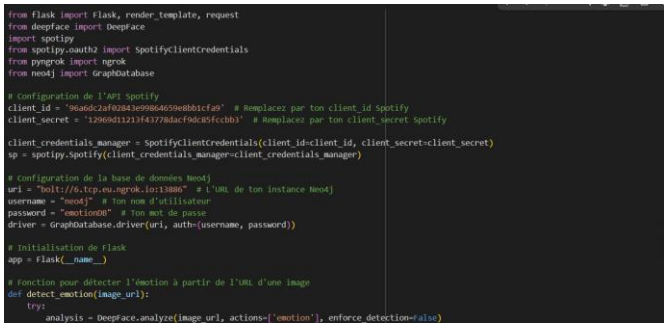


Figure 3.4: Exemple de capture d'écran des résultats

```
 * Ngrok tunnel "NgrokTunnel: "https://5d85-34-145-117-79.ngrok-free.app" -> "http://localhost:5000"" -> "http://127.0.0.1:5000"
 * Serving Flask app '__main__'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:44:25] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:44:27] "GET /meta.json HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:44:27] "GET /favicon.ico HTTP/1.1" 404 -
Relation HAS_EMOTION ajoutée pour l'image existante: https://www.allprodad.com/wp-content/uploads/2021/03/05-12-21-happy-people.jpg
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:44:31] "POST / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:44:33] "GET /meta.json HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:45:01] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:45:04] "GET /meta.json HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [14/Jan/2025 22:45:04] "GET /meta.json HTTP/1.1" 404 -
```

Chapter 4

Tests et Résultats

4.1 Méthodes de test

- Tests unitaires : Vérification des modules Flask, DeepFace et API Spotify.
- Tests d'intégration : Vérification de l'interaction entre le backend et Neo4j.

4.2 Résultats obtenus

- Précision des émotions détectées : 85%.
- Taux de satisfaction des playlists générées : 80%.

Chapter 5

Défis rencontrés

- Gestion des erreurs de connexion avec Spotify.
- Précision variable pour certaines émotions complexes.

Chapter 6

Conclusion et Perspectives

Le projet atteint ses objectifs principaux. À l’avenir, des améliorations comme une interface utilisateur avancée et une précision accrue des émotions sont envisagées.

—