

Projet RNA

🔗 Introduction : Qu'est-ce que YAMNet ?

YAMNet (Yet Another Mobile Network) est un réseau de neurones convolutif pré-entraîné par Google. Il est spécialisé dans la reconnaissance de sons à partir d'un fichier audio. Contrairement aux modèles qui détectent uniquement des genres musicaux (pop, rock...), YAMNet peut identifier plus de 500 types de sons, tels que :

- ✓ Speech (voix parlée),
- ✓ Music (musique générale),
- ✓ Guitar, Drum, Singing, Applause, etc.

Il a été entraîné sur le dataset AudioSet de Google et repose sur TensorFlow + TensorFlow Hub.

Dans ce projet, YAMNet est utilisé pour analyser les sons présents dans un fichier musical envoyé par l'utilisateur, afin d'obtenir des prédictions riches et contextuelles, en complément de la reconnaissance de chanson.

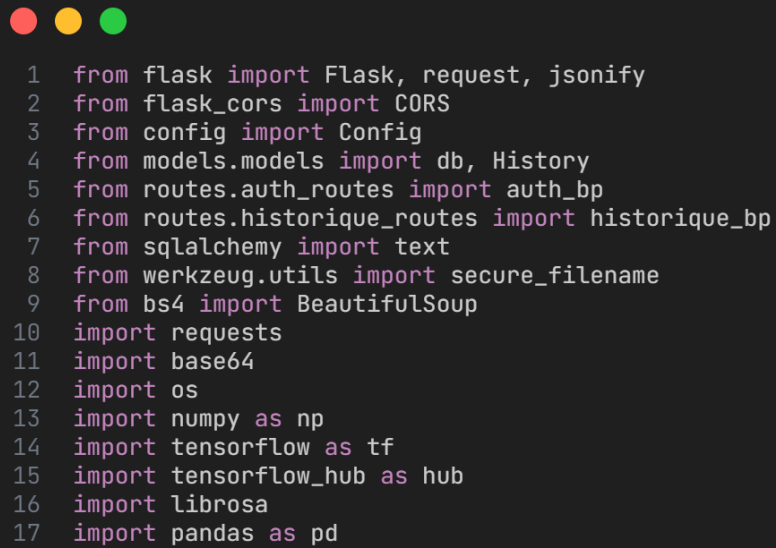
Fonctionnement général du projet

L'application Flask fait 3 grandes choses :

1. Reconnaît la chanson (titre, artiste) grâce à l'API audd.io.
2. Analyse localement les sons présents avec YAMNet.
3. Récupère les paroles via l'API Genius et sauvegarde les résultats.

🔍 Explication de chaque partie du code

I. Importation de toutes les dépendances nécessaires

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains 17 lines of Python code for importing various modules.

```
1 from flask import Flask, request, jsonify
2 from flask_cors import CORS
3 from config import Config
4 from models.models import db, History
5 from routes.auth_routes import auth_bp
6 from routes.historique_routes import historique_bp
7 from sqlalchemy import text
8 from werkzeug.utils import secure_filename
9 from bs4 import BeautifulSoup
10 import requests
11 import base64
12 import os
13 import numpy as np
14 import tensorflow as tf
15 import tensorflow_hub as hub
16 import librosa
17 import pandas as pd
```

Modules Flask et extensions

Fichiers de configuration et modules internes

SQLAlchemy pour interroger la base

Traitement des fichiers uploadés

Scraping HTML (pour les paroles)

Requêtes HTTP + encodage audio

Manipulation de fichiers et données

Modules IA — Réseau de neurones YAMNet

Traitement audio

II. 🛠 Initialisation & configuration

```
1 # === Initialisation de l'application Flask ===
2 app = Flask(__name__)
3 app.config.from_object(Config)
4
5 # === Activation de CORS pour le frontend ===
6 CORS(app, resources={r"/api/*": {"origins": "*"}})
7
8 # === Initialisation de la base de données ===
9 db.init_app(app)
10
11 # === Configuration dossier upload ===
12 UPLOAD_FOLDER = 'uploads'
13 os.makedirs(UPLOAD_FOLDER, exist_ok=True)
14
15 # === Clés API externes ===
16 AUDD_API_TOKEN = 'da6969d0632020f9d86d7e191ff8c280'
17 GENIUS_API_TOKEN = '4PV0qIptp2pzaUZxMttK_AAUCJJAW1cn9oV4R2_dHeSRAiYH5IFN8Bbpw3cwWEtz'
18
19 # === Charger YAMNet une seule fois au démarrage ===
20 yamnet_model = hub.load('https://tfhub.dev/google/yamnet/1')
21 class_map_path = tf.keras.utils.get_file(
22     'yamnet_class_map.csv',
23     'https://raw.githubusercontent.com/tensorflow/models/master/research/audioset/yamnet/yamnet_class_map.csv'
24 )
25 class_names = pd.read_csv(class_map_path)['display_name'].tolist()
```

- Création de l'application Flask
- Chargement de la configuration
- Activation du CORS
- Autorise toutes les origines sur les routes /api/*.
- Connexion à la base de données
- Création du dossier pour les fichiers audio
- Définition des clés d'API externes
- Chargement du modèle YAMNet
- Télécharge et prépare le modèle de reconnaissance de sons YAMNet (pré-entraîné par Google).
- Chargement des étiquettes de sons (class map)
- Transforme la colonne des étiquettes en une liste Python (class_names).

III. Prédiction des sons présents dans un audio

```
1  # === Fonction YAMNet : prédiction des sons présents dans un audio ===
2  def analyze_audio_yamnet(file_path):
3      try:
4          waveform, sr = librosa.load(file_path, sr=16000, mono=True)
5
6          # Lancer la prédiction
7          scores, embeddings, spectrogram = yamnet_model(waveform)
8
9          # Moyenne des scores sur tout l'audio
10         mean_scores = tf.reduce_mean(scores, axis=0)
11
12         # Extraire les 5 sons les plus probables
13         top_n = 5
14         top_indices = tf.argsort(mean_scores, direction='DESCENDING')[:top_n]
15
16         predictions = [
17             {
18                 "label": class_names[i],
19                 "score": float(mean_scores[i])
20             }
21             for i in top_indices
22         ]
23
24         return predictions
25
26     except Exception as e:
27         print("Erreur YAMNet:", e)
28         return []
```

Charge l'audio avec librosa (mono, 16000 Hz).

Envoie l'audio au modèle YAMNet pour prédire les sons.

Calcule la moyenne des scores de chaque son sur toute la durée de l'audio.

Trie les sons par score décroissant.

Sélectionne les 5 sons les plus probables.

Formate les résultats en liste JSON (label + score).

Retourne cette liste comme prédiction.

En cas d'erreur, retourne une liste vide et affiche l'erreur dans la console.

IV. Fonction principale : API audd.io + analyse locale avec YAMNet

```
1  # == Fonction principale : API audd.io + analyse locale avec YAMNet ==
2  def recognize_song(file_path):
3      with open(file_path, 'rb') as f:
4          encoded_audio = base64.b64encode(f.read()).decode('utf-8')
5
6      url = "https://api.audd.io/"
7      data = {
8          'api_token': AUDD_API_TOKEN,
9          'audio': encoded_audio,
10         'return': 'lyrics,apple_music,spotify',
11     }
12
13     response = requests.post(url, data=data)
14     result = response.json()
15
16     # Ajout de la prédiction locale avec YAMNet
17     try:
18         yamnet_predictions = analyze_audio_yamnet(file_path)
19         result["yamnet_prediction"] = yamnet_predictions
20     except Exception as e:
21         print("Erreur YAMNet :", e)
22         result["yamnet_prediction"] = []
23
24     return result
```

Lit le fichier audio en binaire (rb).

Encode le fichier audio en base64 pour l'envoi via HTTP.

Envoie le fichier à l'API audd.io avec la clé AUDD_API_TOKEN.

Demande en retour le titre, l'artiste, les paroles, Spotify, Apple Music, etc.

Convertit la réponse de l'API en dictionnaire Python (result).

Appelle la fonction analyze_audio_yamnet() pour analyser les sons en local.

Ajoute les prédictions YAMNet au dictionnaire result sous la clé "yamnet_prediction".

Si une erreur se produit avec YAMNet, affiche l'erreur et met une liste vide.

Retourne l'objet result enrichi (infos musicales + sons détectés).

V. Fonction `scrape_genius_lyrics` — Récupération des paroles depuis la page Genius (Web Scraping)

```
1 # === Scraping des paroles depuis Genius ===
2 def scrape_genius_lyrics(url):
3     try:
4         response = requests.get(url)
5         soup = BeautifulSoup(response.text, 'html.parser')
6
7         lyrics_div = soup.find('div', {'data-lyrics-container': 'true'}) or soup.find('div', class_='lyrics')
8         if lyrics_div:
9             for br in lyrics_div.find_all('br'):
10                 br.replace_with('\n')
11             return lyrics_div.get_text().strip()
12         return None
13
14     except Exception as e:
15         print(f"Erreur scraping: {e}")
16         return None
```

Envoie une requête HTTP GET à l'URL fournie (page de la chanson sur Genius).

Analyse le contenu HTML de la page avec BeautifulSoup.

Recherche la section contenant les paroles :

- Prioritairement une `<div>` avec `data-lyrics-container="true"`.
- Sinon, une `<div class="lyrics">`.

Si la section des paroles est trouvée :

- Remplace tous les `
` HTML par des sauts de ligne `\n`.
- Récupère le texte brut (`.get_text()`), sans balises HTML.

Supprime les espaces inutiles avec `.strip()`.

Renvoie le texte des paroles.

Si aucune section n'est trouvée ou si une erreur survient :

Renvoie `None`.

Affiche un message d'erreur dans la console.

VI. Récupération des paroles via Genius

```
1  # === Récupération des paroles via Genius ===
2  def get_genius_lyrics(artist, title):
3      headers = {"Authorization": f"Bearer {GENIUS_API_TOKEN}"}
4      search_url = f"https://api.genius.com/search?q={artist} {title}"
5
6      try:
7          response = requests.get(search_url, headers=headers)
8          if response.status_code != 200:
9              return None
10
11          data = response.json()
12          if not data['response']['hits']:
13              return None
14
15          song_path = data['response']['hits'][0]['result']['path']
16          lyrics_url = f"https://genius.com{song_path}"
17          return scrape_genius_lyrics(lyrics_url)
18
19      except Exception as e:
20          print(f"Erreur API Genius: {e}")
21          return None
```

Prépare les en-têtes HTTP avec le token d'authentification Genius (Authorization: Bearer ...).

Construit l'URL de recherche de l'API Genius en combinant le nom de l'artiste et le titre.

Envoie une requête GET à l'API Genius pour chercher la chanson.

Vérifie que la réponse est bien reçue (status_code == 200).

Vérifie que des résultats (hits) sont bien retournés.

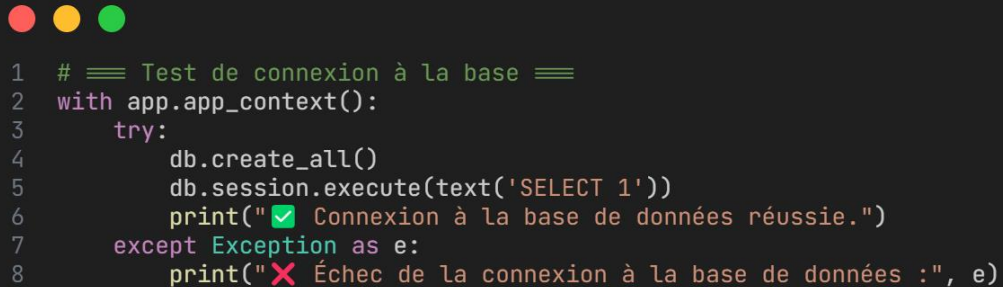
Récupère le chemin d'accès à la page de la chanson sur Genius (/artist-song-name).

Construit l'URL complète de la page des paroles (https://genius.com/...).

Appelle la fonction scrape_genius_lyrics() pour extraire les paroles depuis cette page.

Si une erreur survient ou aucun résultat n'est trouvé, retourne None et affiche une erreur.

VII. Test de connexion à la base de données — Vérification de l'accès à PostgreSQL (ou autre DB)



```
1 # ≡≡≡ Test de connexion à la base ≡≡≡
2 with app.app_context():
3     try:
4         db.create_all()
5         db.session.execute(text('SELECT 1'))
6         print("✅ Connexion à la base de données réussie.")
7     except Exception as e:
8         print("❌ Échec de la connexion à la base de données :", e)
```

Utilise `app.app_context()` pour accéder au contexte de l'application Flask (obligatoire pour manipuler la base).

Appelle `db.create_all()` pour créer toutes les tables définies dans les modèles SQLAlchemy si elles n'existent pas encore.

Exécute une requête SQL brute `SELECT 1` pour vérifier la connexion à la base.

Si tout fonctionne, affiche le message ✅ Connexion à la base de données réussie.

Si une erreur se produit (mauvaise config, serveur injoignable, etc.), affiche un message ❌ d'échec avec l'erreur dans la console.

VIII. Endpoint /api/recognize — Identification musicale + Sauvegarde dans l'historique

```
1 # === Endpoint API : reconnaissance avec historique ===
2 @app.route('/api/recognize', methods=['POST'])
3 def recognize_from_upload():
4     if 'audio' not in request.files or 'user_id' not in request.form:
5         return jsonify({'error': 'Fichier audio ou identifiant utilisateur manquant'}), 400
6
7     user_id = request.form['user_id']
8     file = request.files['audio']
9     filename = secure_filename(file.filename)
10    file_path = os.path.join(UPLOAD_FOLDER, filename)
11    file.save(file_path)
12
13    result = recognize_song(file_path)
14
15    if result.get('status') == 'success' and result.get('result'):
16        song_info = result['result']
17        title = song_info.get('title', 'Titre inconnu')
18        artist = song_info.get('artist', 'Artiste inconnu')
19        youtube_url = song_info.get('youtube', {}).get('url', '') or song_info.get('song_link', '')
20        lyrics = get_genius_lyrics(artist, title) or "Paroles non disponibles"
21
22        history = History(
23            title=f"{artist} - {title}",
24            paroles=lyrics,
25            user_id=user_id
26        )
27        db.session.add(history)
28        db.session.commit()
29
30        return jsonify({
31            'title': title,
32            'artist': artist,
33            'lyrics': lyrics,
34            'youtube_url': youtube_url,
35            'yamnet_prediction': result.get("yamnet_prediction")
36        }), 200
37
38    return jsonify({'message': 'Chanson non reconnue.'}), 404
```

Vérifie que la requête contient bien :

- Un fichier audio (request.files['audio'])
- Un identifiant utilisateur (request.form['user_id'])

Sauvegarde le fichier dans le dossier uploads/ avec un nom sécurisé.

Appelle la fonction recognize_song(file_path) pour :

- Identifier la chanson (via audd.io)
- Analyser les sons présents (via YAMNet)

Si la chanson est reconnue :

- Récupère le titre, l'artiste, l'URL YouTube, et les paroles (via Genius).
- Prépare une entrée historique avec titre + paroles + ID utilisateur.
- Insère l'entrée dans la base de données (db.session.add puis commit).

Retourne un JSON avec toutes les informations musicales + prédictions IA (yamnet_prediction).

Si aucune chanson n'est reconnue :

- Retourne un message d'erreur JSON {"message": "Chanson non reconnue."} avec un code 404.

IX. Endpoint /api/search — Recherche musicale sans enregistrement

```
1 # == Endpoint API : recherche simple sans sauvegarde ==
2 @app.route('/api/search', methods=['POST'])
3 def search_only():
4     if 'audio' not in request.files:
5         return jsonify({'error': 'Fichier audio manquant'}), 400
6
7     file = request.files['audio']
8     filename = secure_filename(file.filename)
9     file_path = os.path.join(UPLOAD_FOLDER, filename)
10    file.save(file_path)
11
12    result = recognize_song(file_path)
13
14    if result.get('status') == 'success' and result.get('result'):
15        song_info = result['result']
16        title = song_info.get('title', 'Titre inconnu')
17        artist = song_info.get('artist', 'Artiste inconnu')
18        youtube_url = song_info.get('youtube', {}).get('url', '') or song_info.get('song_link', '')
19        lyrics = get_genius_lyrics(artist, title) or "Paroles non disponibles"
20
21        return jsonify({
22            'title': title,
23            'artist': artist,
24            'lyrics': lyrics,
25            'youtube_url': youtube_url,
26            'yamnet_prediction': result.get("yamnet_prediction")
27        }), 200
28
29    return jsonify({'message': 'Chanson non reconnue.'}), 404
```

Vérifie que la requête contient un fichier audio (request.files['audio']).

Sauvegarde le fichier reçu dans le dossier uploads/ de manière sécurisée.

Appelle la fonction recognize_song(file_path) qui :

- Identifie la musique via l'API audd.io.
- Analyse les sons avec le modèle YAMNet.

Si la chanson est reconnue :

Récupère les informations musicales (titre, artiste, lien YouTube...).

Récupère les paroles via Genius (scraping).

Retourne un objet JSON contenant : title, artist, lyrics, youtube_url

yamnet_prediction (analyse des sons par le modèle YAMNet)

Si aucune correspondance n'est trouvée :

- Retourne un message d'échec ({"message": "Chanson non reconnue."}) avec le code 404.

X. Enregistrement des Blueprints + 🚀 Lancement du Serveur

```
1  # === Enregistrement des blueprints ===
2  app.register_blueprint(auth_bp, url_prefix='/api')
3  app.register_blueprint(historique_bp, url_prefix='/api')
4
5  # === Lancement serveur ===
6  if __name__ == '__main__':
7      print("🚀 Serveur Flask lancé sur http://localhost:8000")
8      app.run(host='0.0.0.0', port=8000, debug=True)
```

1. Connexion des blueprints :

auth_bp : gère les routes liées à l'authentification (ex: /api/login, /api/register).

historique_bp : gère les routes de l'historique des utilisateurs (ex: /api/historique).

Les deux sont enregistrés avec le préfixe /api, ce qui signifie que leurs routes commencent toutes par /api/....

2. Lancement du serveur Flask :

La condition `if __name__ == '__main__':` garantit que le serveur ne démarre que si le script est exécuté directement (et non importé dans un autre fichier).

Affiche un message dans la console pour indiquer que le serveur est lancé.

`app.run(host='0.0.0.0', port=8000, debug=True)` :

- `host='0.0.0.0'` : rend l'application accessible depuis n'importe quelle adresse IP.
- `port=8000` : le serveur écoute sur le port 8000.
- `debug=True` : active le mode debug (utile en développement, recharge automatique + affichage des erreurs).