Ecole Nationale
Supérieure
d'Ingénieurs
de Tunis

الجمهورية التونسية
وزارة التعليم العالي والبحث العلمي
جامعة تونس
المدرسة الوطنية العليا للمهندسين بتونس

Rapport du mini projet
Nouvelles Architectures

Troisième année en Génie Informatique

Par

Wiem BEN SABER

Prédiction des genres des fichiers audio

Composition du jury

Monsieur BOULARES Mehrez ..... **Président**

Année universitaire : 2023-2024

5, Avenue Taha Hussein – Tunis
B. P. 56, Bab Menara 1008

Tel. : 71 . 496 . 066 الهاتف:
Fax : 71 . 391. 166 فاكس :

5، شارع طه حسين ـ تونس
ص . ب . : 56 باب منارة 1008

Table de matières

Liste des figures

Chapitre 1. Cadre du mini projet

1. Context du mini projet

Le monde de la technologie évolue rapidement, et l'intégration de solutions innovantes est essentielle pour relever les défis complexes auxquels nous sommes confrontés. Dans cette optique, notre projet s'articule autour de l'utilisation de Docker, Python et Flask pour créer un environnement Machine Learning dédié à la classification des genres musicaux.

2. Objectif du mini projet

Notre mission est de concevoir, développer et mettre en œuvre un environnement complet basé sur la technologie Docker, capable d'intégrer et tester des modèles de Machine Learning dédiés à la classification des genres musicaux. Ce projet se fixe également pour objectif de fournir deux services web Flask, l'un basé sur le modèle SVM et l'autre sur le modèle VGG19, permettant ainsi de répondre de manière efficace et précise aux besoins de classification audio.

À travers ce mini projet, nous visons à démontrer la puissance de l'intégration des technologies modernes dans le domaine du Machine Learning tout en garantissant des résultats de classification fiables. La mise en place d'un environnement d'intégration continue avec Jenkins vise à automatiser les processus de tests, fournissant ainsi une assurance qualité continue tout au long du cycle de vie du projet.

Chapitre 2. Environnement Machine Learning

1. Dataset

Le choix d'un dataset approprié est crucial pour la réussite d'un projet de Machine Learning. Le dataset Kaggle "GTZAN Music Genre Classification" a été retenu en raison de sa diversité. Il comprend un ensemble d'échantillons audio soigneusement étiquetés, couvrant une large gamme de genres musicaux. Cette variété permet une formation robuste des modèles, garantissant une classification précise dans des scénarios réels.

2. Choix des Modèles de Machine Learning

Le choix des modèles de Machine Learning est stratégique pour atteindre les objectifs de classification audio. Deux modèles ont été sélectionnés en fonction de leurs caractéristiques distinctes :

- **Support Vector Machine (SVM)**

Le modèle SVM a été choisi pour sa simplicité et son efficacité dans la classification de données. En utilisant des vecteurs de support, le SVM est capable de séparer efficacement les différentes classes de genres musicaux.

Le code que j'ai généré le modèle svm avec est **model_svm.py** dans sous le dossier models.

- **VGG19**

Le modèle VGG19, initialement conçu pour la classification d'images, a été adapté pour la classification audio. Son architecture profonde permet d'extraire des caractéristiques complexes des données sonores, offrant ainsi une précision accrue dans la prédiction des genres musicaux.

Le code que j'ai généré le modèle svm avec est **model_vgg19.py** dans sous le dossier models.

Chapitre 3. Services Web Flask

Introduction

Flask est un micro framework web en Python, fournissent une architecture légère mais puissante pour exposer les fonctionnalités de nos modèles de Machine Learning.

1. Service SVM

Le service SVM a été créé pour assurer la classification des genres musicaux. Son fonctionnement repose sur les étapes suivantes :

1) Réception d'une requête HTTP avec un fichier audio WAV encodé en base64 en tant que paramètre.
2) Décodage du fichier WAV pour obtenir les données audio brutes.
3) Utilisation du modèle SVM préalablement entraîné pour prédire le genre musical associé.
4) Renvoi de la prédiction résultante.

Le fichier **app.py** du **service SVM** est le cœur du backend qui prend en charge la classification des genres musicaux à partir de fichiers audio encodés en base64. Ce script Flask expose une API RESTful, répondant aux requêtes POST envoyées à l'endpoint /predict.

Chargement des Modèles

Au démarrage, le script charge le modèle SVM pré-entraîné ainsi que le scaler à partir des fichiers modeltest.pkl et scalertest.pkl. Ces éléments sont essentiels pour la prédiction précise des genres musicaux.

```
svm_model = joblib.load('modeltest.pkl')
scaler = joblib.load('scalertest.pkl')
```

Fonctions d'Extraction et d'Encodage des Caractéristiques

Les fonctions **extract_and_encode_features** et **decode_features** sont responsables de la conversion des fichiers audio en données exploitables par le modèle SVM.

La fonction **extract_and_encode_features** prend un fichier audio, extrait les caractéristiques MFCC (Mel-frequency cepstral coefficients), les encode en base64 et retourne la représentation base64. La fonction **decode_features** effectue l'opération inverse, décodant la représentation base64 pour obtenir les caractéristiques MFCC.

Route de Prédiction (/predict)

La route /predict écoute les requêtes POST contenant des fichiers audio encodés en base64. Le processus de prédiction commence en enregistrant temporairement le fichier audio, puis en extrayant et encodant ses caractéristiques.

```
audio_file = request.files['audio_file']
temp_file_path = 'audio_to_predict.wav'
audio_file.save(temp_file_path)
new_audio_features = extract_and_encode_features(temp_file_path)
```

La représentation base64 des caractéristiques est ensuite décodée et les prédictions sont effectuées à l'aide du modèle SVM.

```
features = decode_features(new_audio_features)
predicted_genre = svm_model.predict(features.reshape(1, -1))
```

Gestion des Erreurs

Des vérifications sont effectuées pour s'assurer que le fichier audio contient le nombre attendu de caractéristiques. Si ce n'est pas le cas, un message d'erreur est renvoyé.

Lancement du Service

Le service SVM est configuré pour écouter sur l'adresse IP 0.0.0.0 et le port 8081.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8081)
```

2. Service VGG19

Le service vgg19 opère de manière similaire au service SVM, mais utilise un modèle VGG19 préalablement adapté à la classification audio. Les étapes du fonctionnement incluent:

1) Réception d'une requête HTTP avec un fichier audio WAV encodé en base64 en tant que paramètre.
2) Décodage du fichier WAV pour extraire les données audio brutes.
3) Utilisation du modèle VGG19 adapté pour prédire le genre musical du fichier audio.
4) Renvoi de la prédiction résultante.

Le fichier app.py du service VGG19 (vgg19_service) représente le backend responsable de la classification des genres musicaux à partir de fichiers audio. Comme pour le SVM_service, ce script Flask expose une API RESTful, répondant aux requêtes POST envoyées à l'endpoint /vgg.

Chargement du Modèle VGG19

Au démarrage, le script charge le modèle VGG19 pré-entraîné à partir du fichier vgg19_model.h5. Cette étape est cruciale pour permettre au service de réaliser des prédictions basées sur les caractéristiques extraites des fichiers audio.

```
model = load_model('vgg19_model.h5')
```

Fonction de Prédiction (predict_genre)

La fonction **predict_genre** est responsable du prétraitement des fichiers audio et de la réalisation des prédictions. Les étapes comprennent :

- ➢ Chargement du fichier audio avec Librosa.
- ➢ Création d'un spectrogramme mel à l'aide de Librosa.
- ➢ Redimensionnement du spectrogramme pour correspondre à la forme d'entrée attendue par le modèle VGG19.
- ➢ Reshape des données d'entrée pour la prédiction.
- ➢ Prédiction du genre musical à l'aide du modèle VGG19.

Les résultats sont ensuite interprétés pour obtenir le genre musical prédit.

Route de Prédiction (/vgg)

La route /vgg écoute les requêtes POST contenant des fichiers audio. Le script vérifie la présence du fichier audio dans la requête, puis appelle la fonction de prédiction.

```
audio_file = request.files['audio_file']
predicted_genre = predict_genre(audio_file)
return jsonify({'predicted_genre': predicted_genre})
```

En cas de succès, le genre prédit est renvoyé en réponse à la requête.

Gestion des Erreurs

Le script est conçu pour gérer les erreurs de manière explicite, renvoyant un message d'erreur détaillé si un problème survient lors de la prédiction.

Lancement du Service

Le service VGG19 est configuré pour écouter sur l'adresse IP 0.0.0.0 et le port 8082.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8082)
```

3. Service FrontEnd

Le fichier app.py du service Frontend représente la partie utilisateur de notre application, offrant une interface web permettant de télécharger des fichiers audio, de les envoyer aux

services de classification (SVM et VGG19), et d'afficher les résultats de prédiction. Voici une explication détaillée de son fonctionnement :

Définition des URLs des Services Backend

Les URL des services backend SVM (svm_backend_url) et VGG19 (vgg19_backend_url) sont définies pour permettre au frontend de communiquer avec les services de classification.

```
svm_backend_url = 'http://10.20.2.11:8081/predict'
vgg19_backend_url = 'http://10.20.2.11:8082/vgg'
```

Route pour la Page d'Accueil (/)

La route principale / gère les requêtes GET et POST. Lorsqu'un utilisateur accède à la page d'accueil, le formulaire permettant de choisir le type de prédiction (SVM ou VGG19) est affiché. Lorsqu'un fichier audio est soumis, le backend correspondant est appelé pour effectuer la prédiction.

Envoi de Requêtes aux Services Backend

Lorsqu'un utilisateur sélectionne le type de prédiction et soumet un fichier audio, une requête POST est envoyée au service backend correspondant (SVM ou VGG19) avec le fichier audio. Le résultat de la prédiction est récupéré à partir de la réponse du backend.

Affichage des Résultats

Les résultats de la prédiction sont affichés sur la page d'accueil, permettant à l'utilisateur de visualiser le genre musical prédit.

Lancement du Service

Le service Frontend est configuré pour écouter sur l'adresse IP 0.0.0.0 et le port 8083.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8083)
```

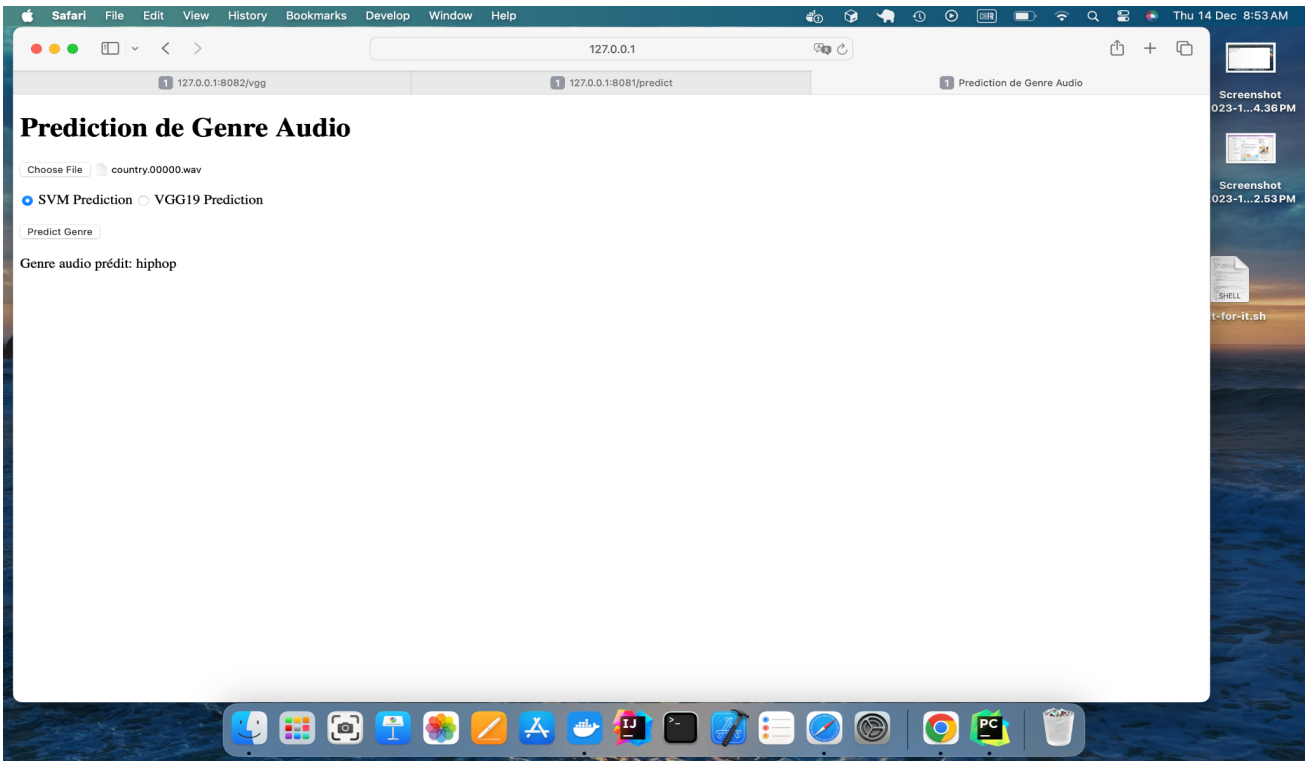

Figure 1. Prédiction avec SVM

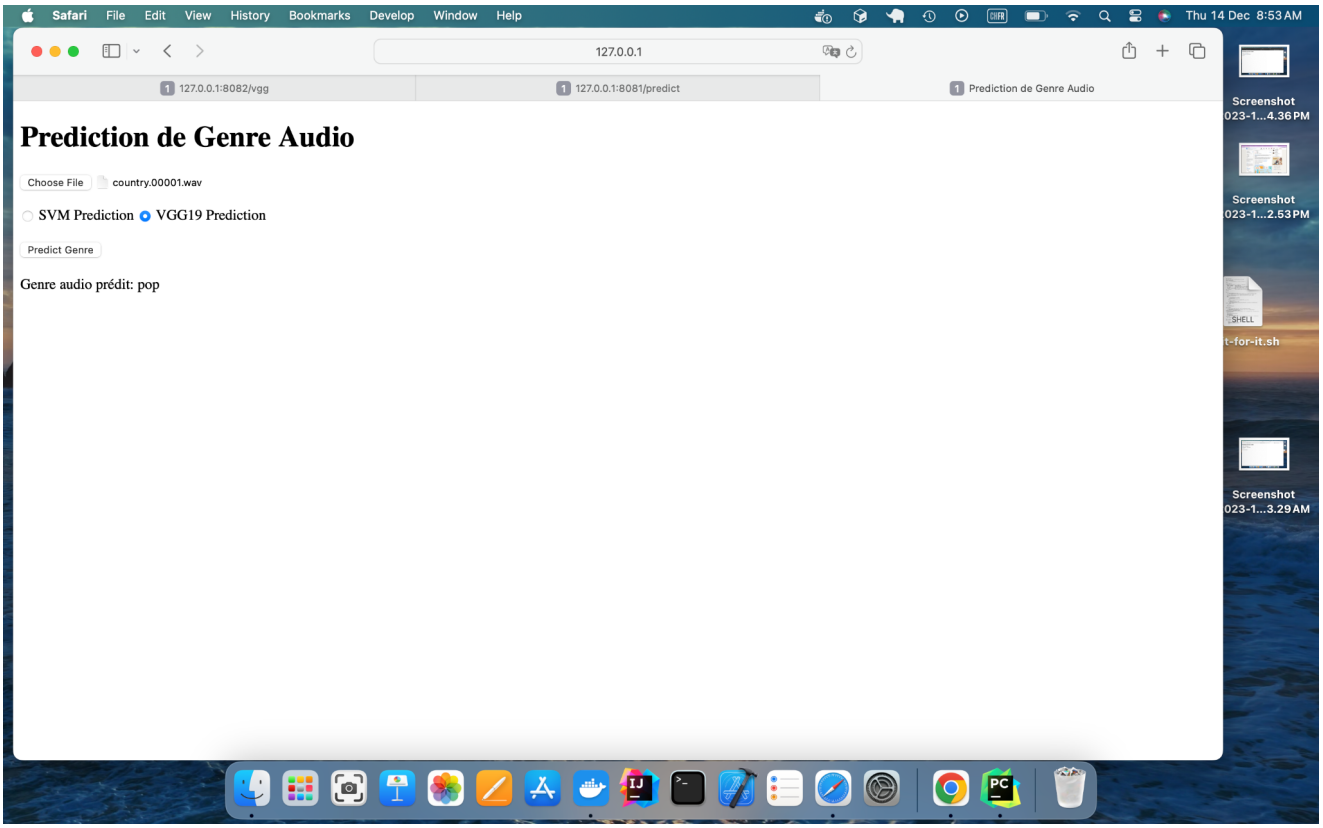

Figure 2. Prédiction avec VGG19

Chapitre 4 . Docker

Introduction

L'utilisation de Docker dans ce mini-projet apporte plusieurs avantages significatifs en termes de développement et gestion de l'environnement Machine Learning basé sur Flask pour la classification des genres musicaux. Il permet d'isoler chaque composant de l'application, y compris les services Flask, les modèles ML, et les dépendances. Chaque service fonctionne dans son propre conteneur, garantissant une isolation complète et évitant les conflits entre les dépendances.

1. Dockerfiles

1) Service SVM (Back-end SVM)

Le Dockerfile pour le service SVM commence par utiliser l'image Python 3.9 officielle. Il installe les dépendances nécessaires, copie les fichiers backend dans le conteneur, installe les packages requis, expose le port 8081, définit l'environnement Flask en mode production, et enfin, lance l'application Flask.

```
# Use an official Python runtime as a parent image
FROM python:3.9
# Set the working directory to /app
WORKDIR /app
# Install the required packages
RUN apt-get update && apt-get install -y libsndfile1
# Copy the necessary backend files to the container
COPY app.py modeltest.pkl requirements.txt scalertest.pkl /app/
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt
# Expose the port used by your Flask application
EXPOSE 8081
# Define environment variable for Flask to run in production mode
ENV FLASK_ENV=production
# Command to start the application
CMD ["python", "app.py"]
```

2) Service VGG19 (Back-end VGG19)

Le Dockerfile pour le service VGG19 utilise également une image Python 3.9-slim. Il installe les dépendances nécessaires, copie les fichiers backend dans le conteneur, installe les packages requis, expose le port 8082, et définit la commande pour lancer l'application Flask.

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim
# Set the working directory to /app
WORKDIR /app
# Install the required packages
RUN apt-get update && apt-get install -y libsndfile1
# Copy the necessary backend files to the container
COPY app.py  requirements.txt vgg19_model.h5 /app/
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt
# Expose the port for Flask application
EXPOSE 8082
# Command to start the Flask application
CMD ["python", "app.py"]
```

3) Service Frontend

Le Dockerfile pour le service Frontend utilise une image Python 3.9-slim. Il copie tous les fichiers frontend dans le conteneur, installe les dépendances spécifiées dans requirements.txt, expose le port 8083, et définit la commande pour exécuter l'application Flask.

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim
# Create and set the working directory
WORKDIR /app
# Copy frontend files into the container
COPY . .
# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
# Expose the port on which the Flask application will run
EXPOSE 8083
# Command to run the Flask application
CMD ["python", "app.py"]
```

On a utilisé les commandes suivantes pour construire nos images et exécuter nos conteneurs; **docker build -t <nom de l'image> .** pour construire nos images Docker

à partir des Dockerfiles spécifiques à chaque service et **docker run -p** pour exécuter et lancer les conteneurs.

2. Docker-compose et volumes

Le fichier docker-compose.yml définit et orchestre les services de votre application.

```
//Service front
 frontend:
   build:
     context: ./Front-end  # Path to the directory containing frontend
Dockerfile
   ports:
     - "8083:8083"
   volumes:
     - ./Front-end:/app
   depends_on:
     - backend
     - vgg19_backend
```

build: Spécifie le chemin vers le répertoire contenant le Dockerfile pour construire l'image du service Frontend.

ports: Mappe le port 8083 du conteneur vers le port 8083 de la machine hôte.

volumes: Montre le répertoire local "./Front-end" dans le conteneur sous "/app". Cela permet de synchroniser les fichiers entre le conteneur et le système local pendant le développement.

depends_on: Indique que ce service dépend des services Backend et VGG19_Backend. Cela signifie que Docker Compose attendra que ces services soient prêts avant de démarrer le service Frontend.

```
//Service backend svm
 backend:
   build:
       context: ./Back-end-svm   # Path to the directory containing
backend Dockerfile
   ports:
     - "8081:8081"
   volumes:
     - backend_data:/Back-end-svm  # Stocke les modèles SVM
```

build: Spécifie le chemin vers le répertoire contenant le Dockerfile pour construire l'image du service Backend.
ports: Mappe le port 8081 du conteneur vers le port 8081 de la machine hôte.
volumes: Crée un volume nommé "backend_data" pour stocker les modèles SVM utilisés par le service Backend.

```
//service backend vgg19
 vgg19_backend:
     build:
         context: ./VGG_19   # Path to the directory containing VGG19
backend Dockerfile
     ports:
       - "8082:8082"
```

build: Spécifie le chemin vers le répertoire contenant le Dockerfile pour construire l'image du service VGG19_Backend.
ports: Mappe le port 8082 du conteneur vers le port 8082 de la machine hôte.

```
volumes:
   backend_data:
```

Définit un volume nommé "backend_data". Ce volume est utilisé par le service Backend pour stocker les modèles SVM. Les volumes Docker garantissent la persistance des données même si les conteneurs sont détruits.

On lance **docker-compose build** pour construire les images Docker pour tous les services, puis **docker-compose up** pour orchestrer le démarrage des conteneurs basés sur ces images, lançant ainsi l'environnement de l'application.

Chapitre 5 . Environnement d'Intégration Continue

1. Mise en Place de Jenkins

Jenkins a été installé et configuré en tant qu'outil d'intégration continue. Son rôle principal est de surveiller le dépôt de code source associé au projet. Une fois un changement détecté, Jenkins déclenche automatiquement le processus d'intégration continue.

Pour automatiser le flux de travail, un pipeline Jenkins a été mis en place. Ce pipeline définit les étapes à suivre pour intégrer les modifications, tester l'application. Les configurations de ce pipeline sont ajustées pour refléter les besoins spécifiques de l'application, assurant une exécution cohérente et fiable du processus.

Le fichier Jenkinsfile ci-dessous définit un pipeline Jenkins de l'application avec des étapes de construction, de lancement de conteneurs, et d'exécution de tests.

```
pipeline {
    agent any
    stages {
        stage('Build and Launch Containers') {
            steps {
                script {
                    // Ensure the script uses the correct PATH
                    env.PATH = "/usr/local/bin:${env.PATH}"
            // Build and launch Docker containers using docker-compose
                    sh '/usr/local/bin/docker-compose up -d --build'
                    sh '/usr/local/bin/docker-compose logs'
                    sh '/usr/local/bin/docker-compose ps'
                }
            }
        }
        stage('Run Tests') {
            steps {
                script {
          // Wait for a few seconds to allow containers to fully start
```

```
                    sleep 10
                    sh 'python3 tests/prediction_tests.py'
        } }}}}
```

2. Scénarios de Tests Automatisés

Les tests automatisés jouent un rôle crucial dans la validation et la vérification de la robustesse des services de prédiction au sein de notre application Flask. Le code du test existe sous **tests/prediction_tests.py**

La classe de tests **TestPredictions** a été élaborée pour évaluer spécifiquement les performances des services Backend SVM (/predict) et VGG19 (/vgg).
Dans la méthode **setUp**, une liste de genres musicaux possibles a été créée, servant de référence pour valider les prédictions futures.

Le test **test_backend_prediction** simule le processus de prédiction du service SVM en envoyant une requête POST avec un fichier audio spécifique. Ce test vérifie la validité de la réponse, en s'assurant qu'elle a un code de statut HTTP 200 et contient la clé "predicted_genre". De plus, il vérifie que la prédiction faite par le service appartient à une liste prédéfinie de genres, incluant une catégorie spécifique "expected_prediction_vgg19".

De manière similaire, le test **test_vgg19_backend_prediction** évalue le service Backend VGG19 en suivant le même schéma. Il s'assure que le fichier audio existe, envoie une requête POST simulée, vérifie la réponse, et valide la prédiction parmi les genres attendus, y compris la catégorie spécifique "expected_prediction_vgg19".

L'inclusion de ces tests automatisés dans notre processus d'intégration continue garantit une évaluation constante de la précision des services de prédiction à mesure que le code évolue.

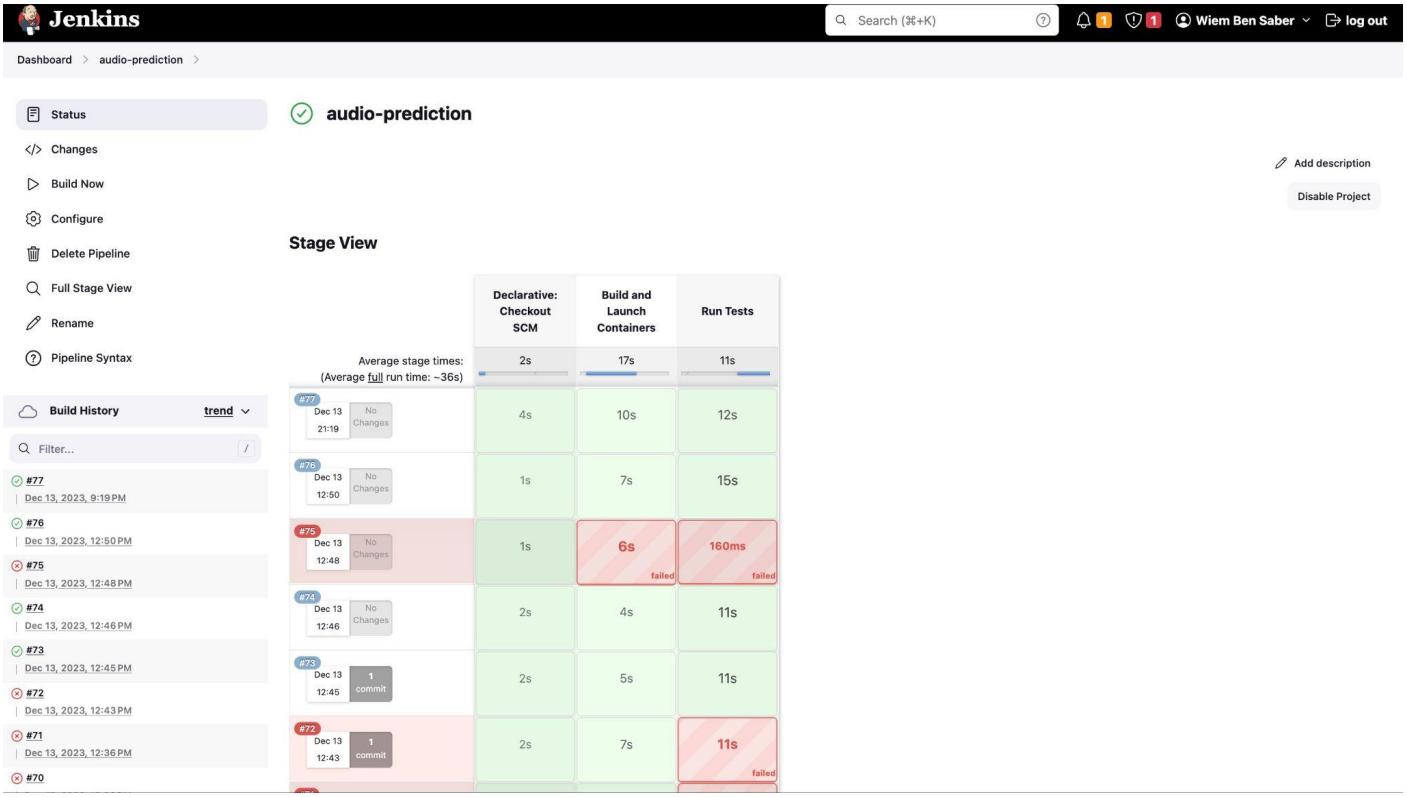

Figure 3. Jenkins pipeline