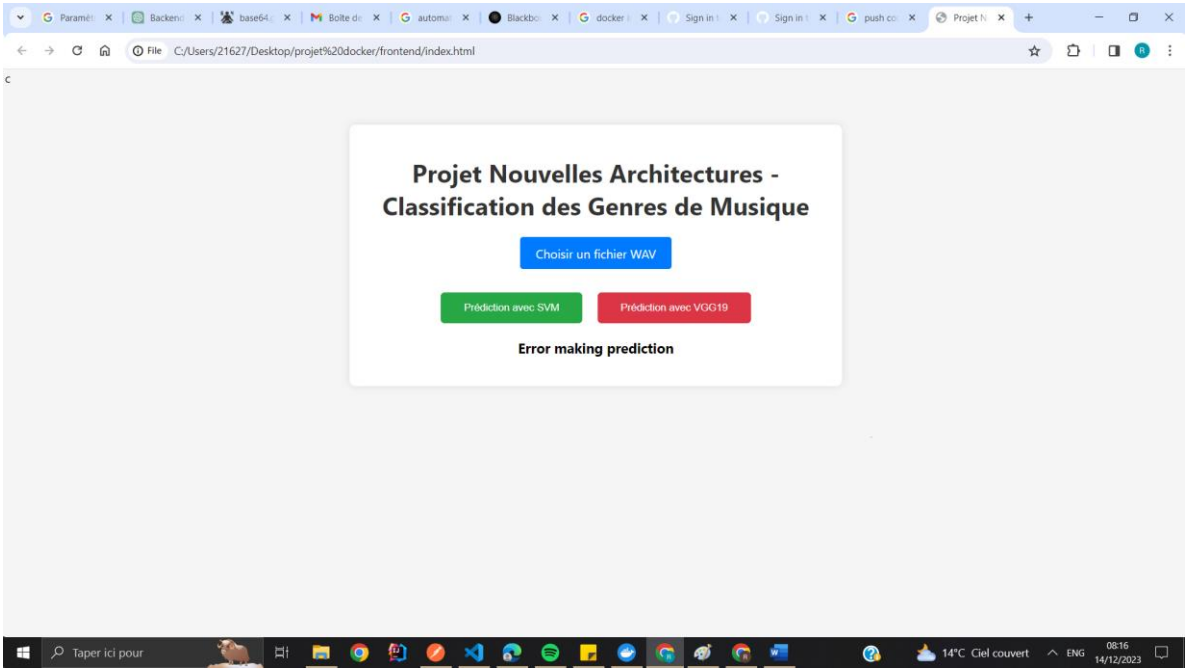
Rapport projet Classification des genres musicaux

1. Création du front-end:



2. Connection du container front-end avec le container backend avec l'adresse IP :

2.1. Connection du bouton « Prédiction avec SVM » avec le container «SVM_backend_container » :

```
// Replace 'backend_ip_address' and 'backend_port' with the actual values
fetch('http://172.17.0.0:5000/predict_SVM', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
```

2.2. Connection du bouton « Prédiction avec VGG19 » avec le container «VGG19_backend_container » :

```
// and 'vgg19_backend_port' with the port on which your backend server is listening.
fetch('http:// 172.17.0.0:5001/predict_vgg', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
```

3. Conversion du fichier WAV en base64 dans le fichier javascript :

```
function predictWithSVM() {
    const audioInput = document.getElementById('audioFile');
    if (audioInput.files.length === 0) {
      alert('Please select a WAV file first.');
      return;
    }
    const audioFile = audioInput.files[0];

    // Convert the audio file to base64
    const reader = new FileReader();
    reader.readAsDataURL(audioFile);
    reader.onload = function() {
      const base64Audio = reader.result.split(',')[1]; // Remove the content type prefix

      const data = { audio: base64Audio };
```

4. L'API reçoit des données de type base64, la convertir en WAV et faire la prédiction :

```
@app.route('/predict_SVM', methods=['POST'])
def predict():
    # Get the base64 encoded audio file from the request
    content = request.json
    audio_base64 = content['audio']
    audio_bytes = base64.b64decode(audio_base64)
    audio_path = 'temp_audio.wav'

    # Save the temporary audio file
    with open(audio_path, 'wb') as audio_file:
        audio_file.write(audio_bytes)

    # Make the prediction
    genre = predict_genre(audio_path)

    # Clean up the temporary file
    os.remove(audio_path)

    # Return the prediction result
    # Convert NumPy types to Python types before returning JSON response
    return jsonify({'genre': genre})  # Explicit conversion to int
```

5. Entrainement et test des modèles SVM et VGG19 :

Pour entraîner et tester un modèle de machine learning pour la prédiction de genres musicaux à partir de fichiers audio WAV, voici une description générale des étapes à suivre pour les deux modèles, SVM et VGG19:

5.1. Prétraitement des données:
- Collecte d'un ensemble de données de fichiers WAV représentant différents genres musicaux.
- Extraction des caractéristiques audio pertinentes pour le genre musical, telles que le spectre de fréquence, le tempo, et les coefficients cepstraux de fréquence de Mel (MFCCs).

5.2. Entraînement du modèle SVM (Support Vector Machine):
- Division de l'ensemble de données en ensembles d'entraînement et de test.
- Normalisation des caractéristiques extraites pour l'ensemble d'entraînement.
- Utilisation de ces caractéristiques pour entraîner le modèle SVM.
- Réglage des hyperparamètres du modèle SVM pour améliorer la performance.

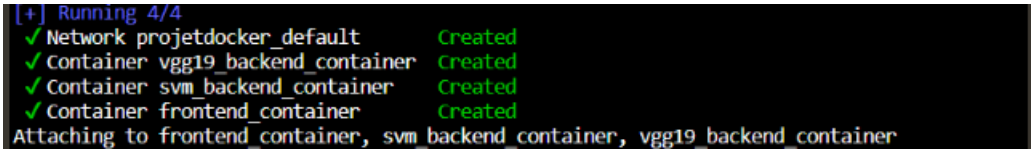
5.3. Entraînement du modèle VGG19:
- Utilisation des spectogrammes générés à partir des fichiers WAV comme input pour le réseau de neurones VGG19.
- Fine-tuning du modèle VGG19 pré-entraîné avec les spectogrammes de l'ensemble d'entraînement.
- Ajustement des dernières couches du réseau VGG19 pour qu'elles correspondent au nombre de genres musicaux à classifier.

5.4. Test des modèles :
- Évaluation des modèles entraînés en utilisant l'ensemble de test pour calculer la précision, le rappel, et le score F1.
- Analyse des résultats pour identifier les genres bien prédits et ceux qui nécessitent des améliorations.

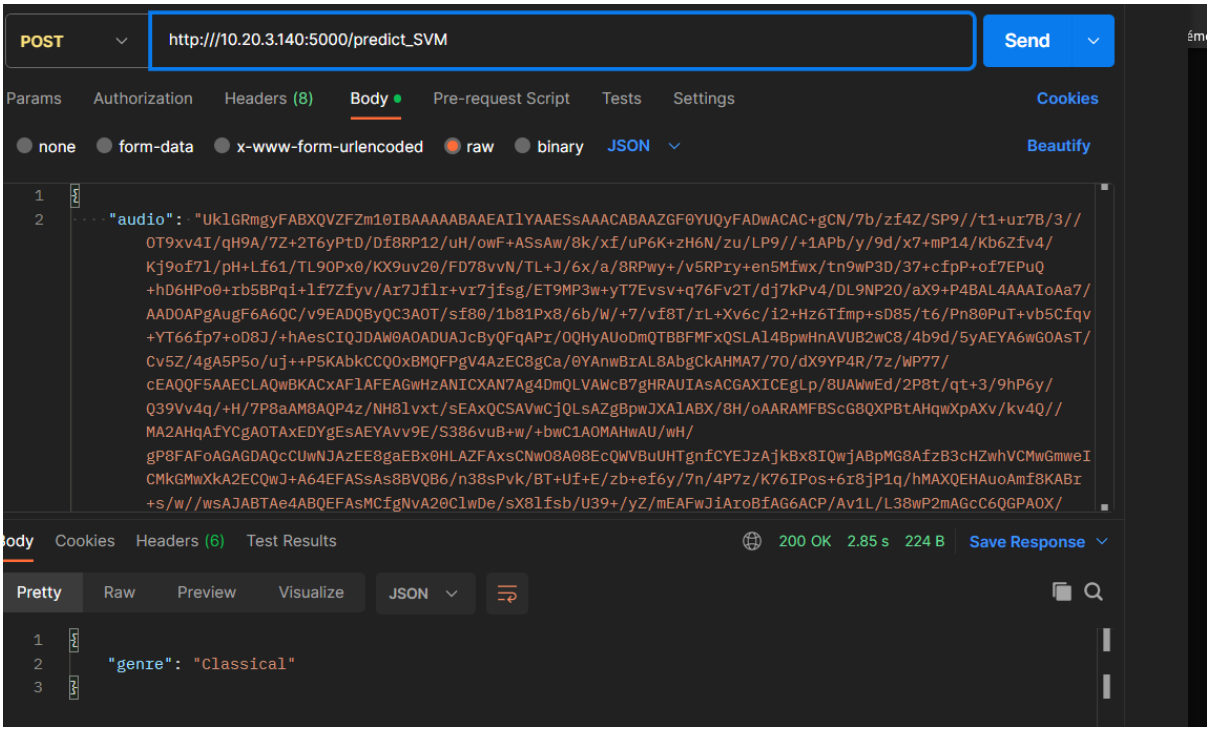
5. 5 Déploiement de l'Endpoint de prédiction:
- Création d'une API REST avec Flask pour servir les modèles entraînés.
- Mise en place d'un endpoint `/predict_SVM` qui reçoit un fichier base64, extrait les caractéristiques nécessaires, et retourne la prédiction du genre musical.
- Containerisation de l'API avec Docker pour faciliter le déploiement.

Chaque modèle aura son propre endpoint : `/predict_svm` pour le modèle SVM et `/predict_vgg19` pour le modèle VGG19. Ces endpoints seront utilisés pour envoyer des requêtes avec des fichiers base64 et recevoir les prédictions de genres musicaux.

6. Création des containers : on a 3 containers :
    - Svm_backend_container
    - Vgg_backend_container
    - Frontend_container



7. Tester les endpoints avec postman :



8. Test avec Jenkins :

Pour tester un modèle de classification de genres musicaux avec Jenkins, voici les étapes et commandes qu'on a faites :

1. Création des Jobs Jenkins :
 - Configuration de deux jobs distincts dans Jenkins, un pour le modèle SVM et un pour le modèle VGG19.

2. Script de Test :
  - Écriture d'un script de test pour chacun des modèles. Ces scripts activent l'environnement virtuel, installent les dépendances, chargent le modèle, et exécutent les tests.

3. Intégration dans Jenkins :
  - Ajout d'une étape de build dans Jenkins pour exécuter les scripts de test.
  - Configuration des actions post-build pour collecter et archiver les résultats des tests.

```bash
# Activation de l'environnement virtuel
source venv/bin/activate

# Installation des dépendances
pip install-r requirements.txt

# Exécution des tests du modèle SVM
python-m unittest discover-s tests-p '*_svm.py'
```

Et pour le modèle VGG19 :

```bash
# Activation de l'environnement virtuel
source venv/bin/activate

# Installation des dépendances
pip install-r requirements.txt

# Exécution des tests du modèle VGG19
python-m unittest discover-s tests-p '*_vgg19.py'
```

Ces commandes sont à placer dans la configuration de l'étape de build du job Jenkins correspondant à chaque modèle. Après l'exécution des tests, Jenkins affichera les résultats et, si configuré, enverra des notifications en cas de succès ou d'échec des tests.

9. Tester avec docker-compose ps :

```
C:\Users\21627\Desktop\projet docker>docker-compose ps
NAME                    IMAGE                     COMMAND                  SERVICE          CREATED          STATUS                          PORTS
frontend_container      projetdocker-frontend     "/docker-entrypoint.…"   frontend         2 hours ago      Up 12 minutes                   0.0.0.0:8
0->80/tcp
svm_backend_container   projetdocker-svm_backend  "flask run"              svm_backend      2 hours ago      Restarting (2) 16 seconds ago
```