Rapport du Mini Projet Nouvelles Architectures

Réalisé par : Maleke Khemiri et Mayssa Bejaoui

Architecture

Le projet est constitué de

app.py : Un fichier Python qui comprend les diverses importations ainsi que les contrôleurs utilisés pour chaque requête HTTP.

DockerFile pour le Front et un **DockerFile** pour le Back

Svm_service.py : qui définit notre service du modèle SVM

Vgg_service.py : qui définit notre service du modèle VGG19

Svm_model : qui définit notre modèle SVM

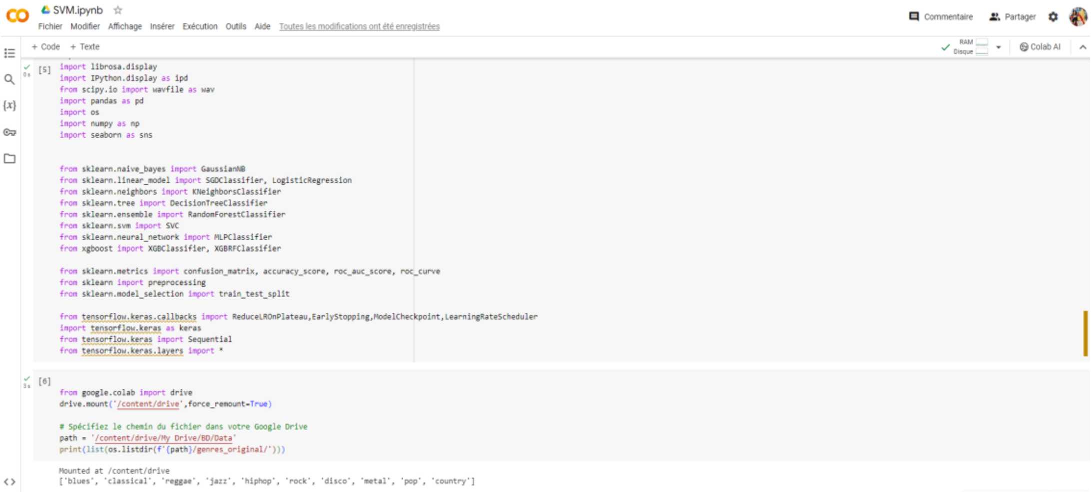
Vgg_model : qui définit notre modèle VGG19

Dossier static : qui contient un fichier **styles.css** et dossier **photos**

Dossier templates : qui contient un fichier index.html

Modèle SVM :

La figure 1 montre les bibliothèques utilisées pour notre modèle SVM.



Après l'entraînement de notre modèle SVM, un fichier **svm_model.pkl** a été généré

La figure ci-dessous montre une partie du code de notre modèle VGG19

```python
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Flatten, Dense, Dropout

def build_vgg16(input_shape, num_classes):
    model = keras.models.Sequential([
        Conv2D(input_shape=input_shape, filters=64, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu"),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding="same"),

        Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding="same"),

        Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding="same"),

        Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding="same"),

        Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"),
        Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"),
        MaxPooling2D(pool_size=(2,2), strides=(2,2), padding="same"),
        Flatten(),
        Dense(4096, activation='relu', kernel_regularizer=keras.regularizers.L1L2(l1=1e-5, l2=1e-4)),
```

Après avoir généré nos modèles SVM et VGG19, nous passons à la création des services pour ces modèles.

Les figures ci-dessous montrent le contenu du fichier **svm_service.py** et **vgg_service.py** en utilisant Flask.

Service SVM

```python
from flask import Flask, request, jsonify
import joblib
import librosa
import numpy as np
import os
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# Load the SVM model
MODEL_PATH = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'svm_model.pkl')
model = joblib.load(MODEL_PATH)

# Map numerical labels to genre names
label_to_genre = {
    0: 'blues',
    1: 'classical',
    2: 'country',
    3: 'disco',
    4: 'hiphop',
    5: 'jazz' ,
    6: 'metal',
    7 : 'pop' ,
    8: 'reggae',
    9: 'rock',
    # Add other genre mappings as needed
}
```

```python
def svm_service():
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'No file provided'})

        file = request.files['file']

        # Save the file to the 'temp' directory
        file_path = os.path.join('temp', 'uploaded_file.wav')
        file.save(file_path)

        # Extract audio features (modify as needed)
        features = extract_features(file_path)

        # Make the prediction with the SVM model
        numerical_label = model.predict([features])[0]

        # Map numerical label to genre name
        predicted_genre = label_to_genre.get(numerical_label, 'Unknown')

        return jsonify({'genre': predicted_genre})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5001)

```

```python
def extract_features(file_path):
    try:
        # Load the audio file using librosa
        y, sr = librosa.load(file_path, sr=None)

        # Extract features (modify as needed)
        # For example, let's extract the mean of the MFCCs
        mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=58)
        mfccs_mean = np.mean(mfccs, axis=1)

        # Concatenate other features if necessary

        return mfccs_mean

    except Exception as e:
        print(f"Error extracting features: {str(e)}")
        return None

@app.route('/svm_service', methods=['POST'])
def svm_service():
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'No file provided'})

        file = request.files['file']

        # Save the file to the 'temp' directory
        file_path = os.path.join('temp', 'uploaded_file.wav')
        file.save(file_path)
```

Service VGG19:

app > vgg_service.py > ...

```python
# vgg19_service.py

from flask import Flask, request, jsonify
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np
from flask_cors import CORS


app = Flask(__name__)
CORS(app)

# Map numerical labels to genre names
label_to_genre = {
    0: 'blues',
    1: 'classical',
    2: 'country',
    3: 'disco',
    4: 'hiphop',
    5: 'jazz' ,
    6: 'metal',
    7 : 'pop' ,
    8: 'reggae',
    9: 'rock',
    # Add other genre mappings as needed
}
# Load the saved VGG19 model

model_vgg19 = tf.saved_model.load('vgg_model.pb')
```

```python

model_vgg19 = tf.saved_model.load('vgg_model.pb')

def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = tf.keras.applications.vgg19.preprocess_input(img_array)
    return img_array

@app.route('/classify_vgg19', methods=['POST'])
def classify_vgg19():
    if 'file' not in request.files:
        return jsonify({'error': 'No file provided'})

    file = request.files['file']
    file_path = '/tmp/uploaded_image.jpg'
    file.save(file_path)

    # Preprocess the image
    img_array = preprocess_image(file_path)

    # Make prediction using the loaded model
    predictions = model_vgg19(img_array)
    decoded_predictions = tf.keras.applications.vgg19.decode_predictions(predictions.numpy(), top=3)[0]

    # Return predictions as JSON
    result = [{'label': label, 'probability': float(prob)} for (_, label, prob) in decoded_predictions]
    return jsonify({'predictions': result})
```

Après avoir crée nos services, nous écrivons un DockerFile afin de créer un conteneur Docker qui contient **svm_service.py** et **vgg_service.py**

DockerFile Front

```
Dockerfile > ...
FROM python:3.8

WORKDIR /app

COPY ./app/svm_service.py /app/
COPY ./app/vgg_service.py /app/
COPY ./app/svm_model.pkl /app/
COPY ./app/vgg_model.pb /app/

RUN pip install Flask flask-cors scikit-learn joblib librosa tensorflow

EXPOSE 5001 5002

CMD ["python", "svm_service.py", "vgg_service.py"]
```

La figure ci-dessous montre la création de notre conteneur avec **docker build**.

```
PS C:\Users\maiss\OneDrive\Bureau\projetDevOps\app> docker build -t back1:latest .
 [+] Building 1.2s (8/11)                                                                   docker:default
  => [internal] load .dockerignore                                                                    0.0s
  => => transferring context: 2B                                                                      0.0s
  => [internal] load build definition from Dockerfile                                                 0.0s
  => => transferring dockerfile: 408B                                                                 0.0s
  => [internal] load metadata for docker.io/library/python:3.8                                        0.8s
  => CANCELED [1/7] FROM docker.io/library/python:3.8@sha256:7264a50439679c2868a99f71a5c9b9831cc082b1d3f05c8643788e1d7914  0.3s
  => => resolve docker.io/library/python:3.8@sha256:7264a50439679c2868a99f71a5c9b9831cc082b1d3f05c8643788e1d7914afeb      0.0s
  => => sha256:8a61cde23424d30d0050cf384b4700250bc5198ce17773c737700fd5e670a06e 7.51kB / 7.51kB       0.0s
  => => sha256:90e5e7d8b87a34877f61c2b86d053db1c4f440b9054cf49573e3be5d6a674a47 0B / 49.58MB          0.3s
  => => sha256:27e1a8ca91d35598fbae8dee7f1c211f0f93cec529f6804a60e9301c53a604d0 0B / 24.05MB          0.3s
```

L’activation de notre conteneur avec **docker run**.

```
PS C:\Users\maiss\OneDrive\Bureau\projetDevOps\app> docker run -p 5001:5001 -p 5002:5002 back1:latest
 * Serving Flask app 'svm_service'
 * Debug mode: on
/usr/local/lib/python3.8/site-packages/sklearn/base.py:348: InconsistentVersionWarning: Trying to unpickle estimator SVC from v
ersion 1.2.2 when using version 1.3.2. This might lead to breaking code or invalid results. Use at your own risk. For more info
 please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://172.17.0.2:5001
```

L’étape suivante consiste à créer notre page web, un code en HTML et en CSS sera écrit pour notre interface web.

Les deux figures ci-dessous montrent une partie du contenu du fichier **index.html** app > templates > index.html > html > head > script > predictImage

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
    <script>
        function predictAudio() {
            // Get the selected audio file
            var audioInput = $("#audioInput")[0].files[0];

            if (audioInput) {
                // Create a FormData object to send the file
                var formData = new FormData();
                formData.append("file", audioInput);

                // Make an AJAX request to the svm_service endpoint
                $.ajax({
                    type: "POST",
                    url: "http://localhost:5001/svm_service",
                    data: formData,
                    contentType: false,
                    processData: false,
                    success: function(response) {
                        // Update the content with the predicted genre
                        console.log(response); // Log the entire response object
                        $("#r").text("Predicted Genre: " + response.genre);

```

```html

                        // Navigate to the result section
                        window.location.href = "#result";
                    },
                    error: function(error) {
                        console.error("Error predicting genre:", error.responseText);
                        $("#r").text("Error predicting genre. Check console for details.");
                    }
                });
            } else {
                alert("Please choose an audio file.");
            }
        }
        function predictImage() {
            var imageInput = $("#imageInput")[0].files[0];

            if (imageInput) {
                var formData = new FormData();
                formData.append("file", imageInput);

                $.ajax({
                    type: "POST",
                    url: "http://localhost:5002/classify_vgg19", // Assuming your VGG19 service is running on
                    data: formData,
                    contentType: false,
                    processData: false,
                    success: function(response){
                        console.log(response);
                        $("#imageResult").text("Predicted Genre: " + response.genre);
                        window.location.href = "#image"; // Scroll to the image result section
```

Front CSS

app > static > # style.css > {} @media only screen and (min-width: 768px) > h1.section-title

```css
@import 'https://fonts.googleapis.com/css?family=Montserrat:300, 400, 700&display=swap';
* {
    padding: 0;
    margin: 0;
    box-sizing: border-box;
}
html {
    font-size: 10px;
    font-family: 'Montserrat', sans-serif;
    scroll-behavior: smooth;
}
a {
    text-decoration: none;
}
.container {
    min-height: 100vh;
    width: 100%;
    display: flex;
    align-items: center;
}
img {
    height: 100%;
    width: 100%;
    object-fit: cover;
}
p {
    color: black;
    font-size: 1.4rem;
    margin-top: 5px;
```

app > static > # style.css > {} @media only screen and (min-width: 768px) > h1.section-title

```css
    line-height: 2.5rem;
    font-weight: 300;
    letter-spacing: 0.05rem;
}
.section-title {
    font-size: 4rem;
    font-weight: 300;
    color: black;
    margin-bottom: 10px;
    text-transform: uppercase;
    letter-spacing: 0.2rem;
    text-align: center;
}
.section-title span {
    color: crimson;
}

.cta {
    display: inline-block;
    padding: 10px 30px;
    color: white;
    background-color: transparent;
    border: 2px solid crimson;
    font-size: 2rem;
    text-transform: uppercase;
    letter-spacing: 0.1rem;
    margin-top: 30px;
    transition: 0.3s ease;
    transition-property: background-color, color;
```

DockerFile Front

L’étape suivante est la création du DockerFile pour notre partie Front

```
app > Dockerfile > CMD
FROM nginx:alpine

RUN rm -rf /usr/share/nginx/html/*

COPY ./static /usr/share/nginx/html/static

COPY ./templates/index.html /usr/share/nginx/html/index.html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

La figure ci-dessous montre la création de notre conteneur avec **docker build**.

```
PROBLEMS 10   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    docker - app

PS C:\Users\maiss\OneDrive\Bureau\projetDevOps\app\app> docker build -t front3:latest .
[+] Building 1.4s (9/9) FINISHED                                                          docker:default
 => [internal] load build definition from Dockerfile                                                0.0s
 => => transferring dockerfile: 259B                                                                0.0s
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                     0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                                     1.3s
 => [1/4] FROM docker.io/library/nginx:alpine@sha256:3923f8de8d2214b9490e68fd6ae63ea604deddd166df2755b788bef04848b9bc  0.0s
 => [internal] load build context                                                                   0.0s
 => => transferring context: 297B                                                                   0.0s
 => CACHED [2/4] RUN rm -rf /usr/share/nginx/html/*                                                 0.0s
 => [3/4] COPY ./static/* /usr/share/nginx/html/static                                              0.0s
 => [4/4] COPY ./templates/index.html /usr/share/nginx/html/index.html                              0.0s
 => exporting to image                                                                              0.1s
 => => exporting layers                                                                             0.0s
 => => writing image sha256:29ec3033b9b8c9b94a8b0b00d812b705f068f0a2680c0fc721d43455736948f3        0.0s
 => => naming to docker.io/library/front3:latest                                                    0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```

L’activation de notre conteneur avec **docker run**.

```
PS C:\Users\maiss\OneDrive\Bureau\projetDevOps\app\app> docker run -p 8080:80 --name container3_front front3:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/12/13 21:49:17 [notice] 1#1: using the "epoll" event method
2023/12/13 21:49:17 [notice] 1#1: nginx/1.25.3
2023/12/13 21:49:17 [notice] 1#1: built by gcc 12.2.1 20220924 (Alpine 12.2.1_git20220924-r10)
2023/12/13 21:49:17 [notice] 1#1: OS: Linux 5.15.133.1-microsoft-standard-WSL2
2023/12/13 21:49:17 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/12/13 21:49:17 [notice] 1#1: start worker processes
2023/12/13 21:49:17 [notice] 1#1: start worker process 30
2023/12/13 21:49:17 [notice] 1#1: start worker process 31
2023/12/13 21:49:17 [notice] 1#1: start worker process 32
```

Docker-Compose

Afin de gérer notre application multi-conteneur. On crée un fichier YAML pour configurer les services afin de faciliter le déploiement et la gestion des conteneurs.

docker-compose.yml

```yaml
version: '3'
services:
  frontend:
    build:
      context: ./app
      dockerfile: Dockerfile
    ports:
      - "80:80"
    volumes:
      - ./app:/app
  backend:
    build:
      context: ./
      dockerfile: Dockerfile
    ports:
      - "5001:5001"
      - "5002:5002"
    volumes:
      - ./app:/app

```

On démarre les conteneurs grâce à **docker-compose up**

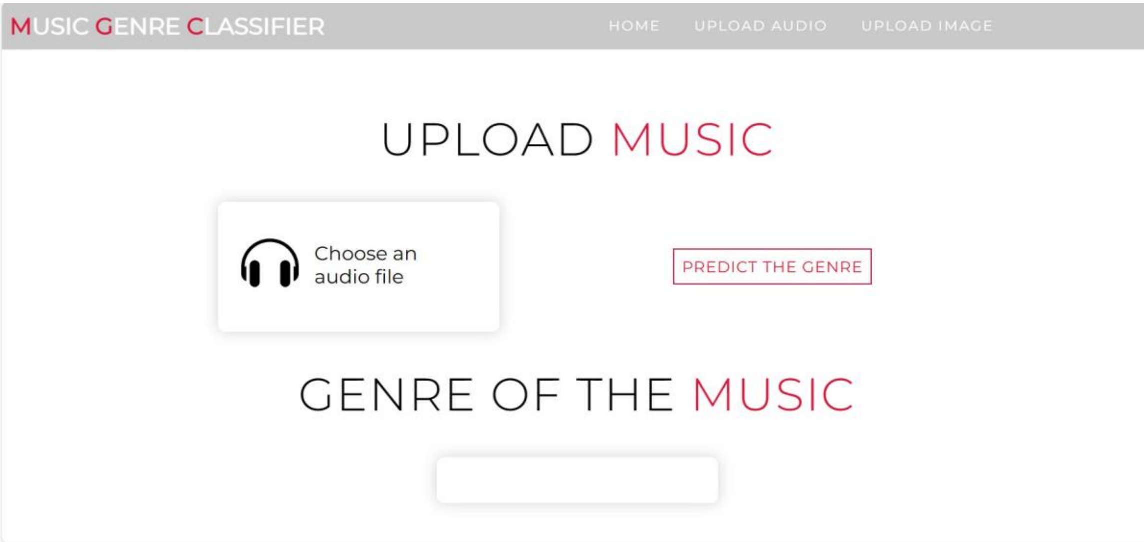
```
PS C:\Users\maiss\OneDrive\Bureau\projetDevOps\app> docker-compose up -d
[+] Building 4.4s (21/21) FINISHED
 => [frontend internal] load build definition from Dockerfile
 => => transferring dockerfile: 259B
 => [frontend internal] load .dockerignore
 => => transferring context: 2B
 => [backend internal] load .dockerignore
 => => transferring context: 2B
 => [backend internal] load build definition from Dockerfile
 => => transferring dockerfile: 352B
 => [backend internal] load metadata for docker.io/library/python:3.8
 => [frontend internal] load metadata for docker.io/library/nginx:alpine
 => [backend internal] load build context
 => => transferring context: 5.04MB
 => [backend 1/7] FROM docker.io/library/python:3.8@sha256:7264a50439679c2868a99f71a5c9b9831cc082b1d3f05c8643788e1
 => [frontend 1/4] FROM docker.io/library/nginx:alpine@sha256:3923f8de8d2214b9490e68fd6ae63ea604deddd166df2755b788
 => [frontend internal] load build context
 => => transferring context: 2.18MB
 => CACHED [frontend 2/4] RUN rm -rf /usr/share/nginx/html/*
 => CACHED [frontend 3/4] COPY ./static/* /usr/share/nginx/html/static
 => [frontend 4/4] COPY ./templates/index.html /usr/share/nginx/html/index.html
 => [frontend] exporting to image
 => => exporting layers
 => => writing image sha256:910b094b95b6268285f8e5ba348f1fd6863fe606c99b9795272418ea8e07ccd2
 => => naming to docker.io/library/app-frontend
 => CACHED [backend 2/7] WORKDIR /app
 => CACHED [backend 3/7] COPY ./app/svm_service.py /app/
 => CACHED [backend 4/7] COPY ./app/vgg_service.py /app/
 => CACHED [backend 5/7] COPY ./app/svm_model.pkl /app/
 => CACHED [backend 6/7] COPY ./app/vgg_model.pb /app/
 => CACHED [backend 7/7] RUN pip install Flask flask-cors scikit-learn joblib librosa tensorflow
 => [backend] exporting to image
```


```
 => CACHED [backend 2/7] WORKDIR /app
 => CACHED [backend 3/7] COPY ./app/svm_service.py /app/
 => CACHED [backend 4/7] COPY ./app/vgg_service.py /app/
 => CACHED [backend 5/7] COPY ./app/svm_model.pkl /app/
 => CACHED [backend 6/7] COPY ./app/vgg_model.pb /app/
 => CACHED [backend 7/7] RUN pip install Flask flask-cors scikit-learn joblib librosa tensorflow
 => [backend] exporting to image
```

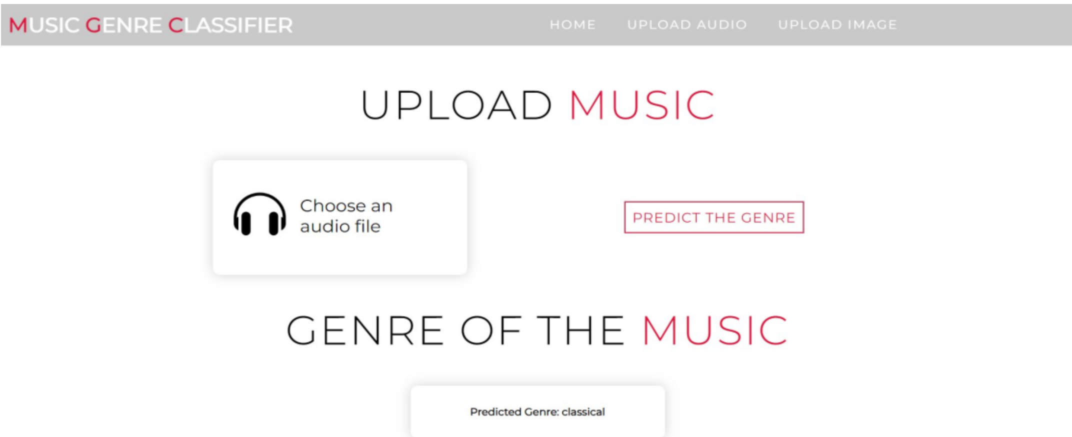

```
 => => exporting layers
 => => writing image sha256:4e39ba6631f7598af8989cc49a3c26113decc738e7dcc6944642deabea987a79
 => => naming to docker.io/library/app-backend
[+] Running 3/3
 ✓ Network app_default        Created
 ✓ Container app-backend-1    Started
 ✓ Container app-frontend-1   Created
Error response from daemon: Ports are not available: exposing port TCP 0.0.0.0:80 -> 0.0.0.0:0: listen tcp 0.0.0.0:8
 attempt was made to access a socket in a way forbidden by its access permissions.
PS C:\Users\maiss\OneDrive\Bureau\projetDevOps\app>
```

Notre interface web permet de prédire le genre musical à travers un son ou une image.

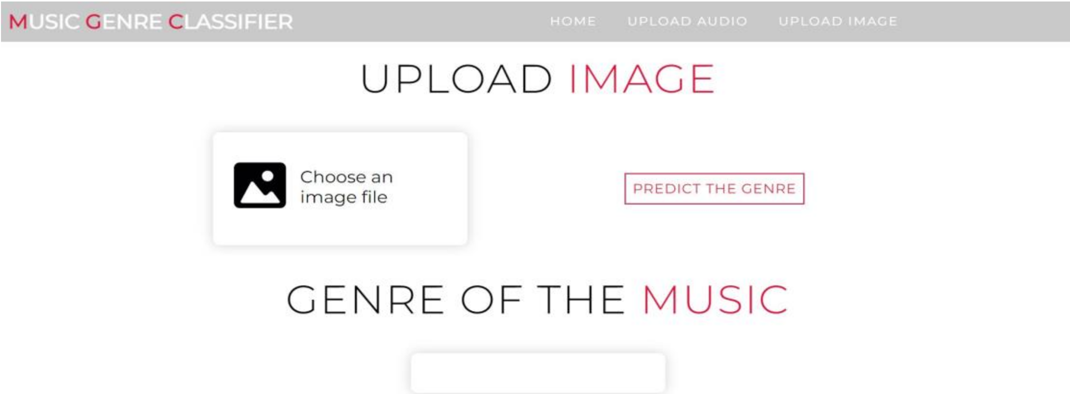
Il suffit de télécharger le son.

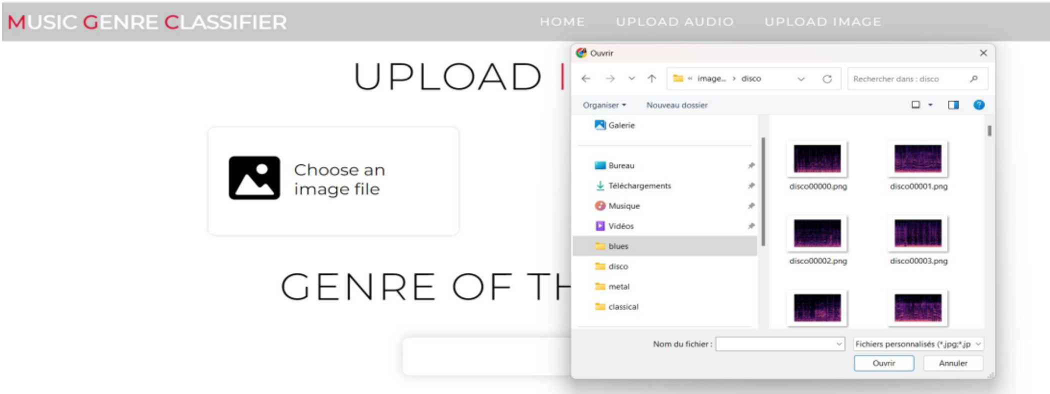


Le résultat sera affiché comme suit.

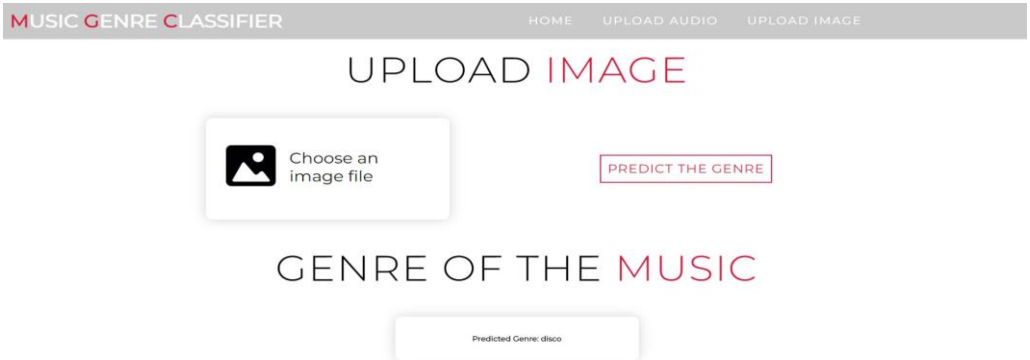


On peut aussi prédire en cliquant sur « choose an image file »

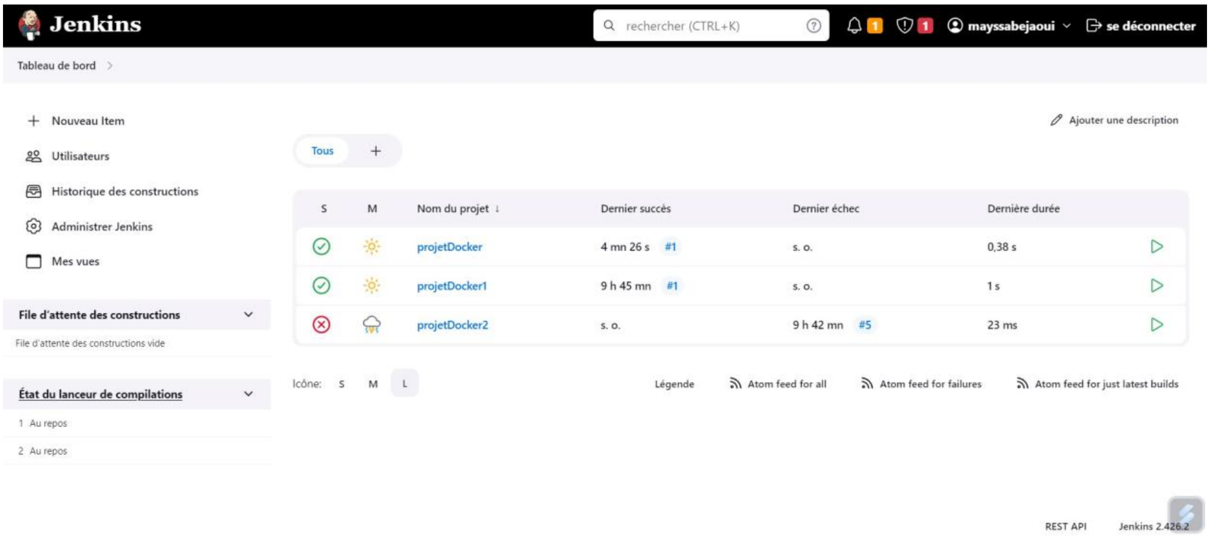




Le résultat sera affiché comme suit.



Pour la partie tests, nous avons choisi Jenkins comme plateforme, la figure ci-dessous montre les tests effectués.


Jenkins
rechercher (CTRL+K)
1
1
mayssabejaoui
se déconnecter
Tableau de bord
Nouveau Item
Utilisateurs
Historique des constructions
Administrer Jenkins
Mes vues
File d'attente des constructions
File d'attente des constructions vide
État du lanceur de compilations
1 Au repos
2 Au repos
Ajouter une description
Tous
S
M
Nom du projet ↓
Dernier succès
Dernier échec
Dernière durée
projetDocker
4 mn 26 s #1
s. o.
0,38 s
projetDocker1
9 h 45 mn #1
s. o.
1 s
projetDocker2
s. o.
9 h 42 mn #5
23 ms
Icône: S M L
Légende
Atom feed for all
Atom feed for failures
Atom feed for just latest builds
REST API
Jenkins 2.426.2