
Rapport de Mini Projet

Nouvelles Architectures

(Docker)

Troisième année en Génie Informatique
(GLID)

Réalisé par
Dridi Rahma

Année universitaire :2023/2024

Introduction générale

Notre mini projet consiste à créer un environnement Machine Learning et deep learning permettant de classifier les genres musicaux à partir d'une base de donnée spécifique. Pour ce faire, nous utilisons comme outil de conteneurisation Docker pour faciliter l'intégration, le déploiement et les tests d'un environnement complet de classification musicale basé sur Python et Flask. L'objectif principal est de développer de trois services web distincts, un pour le support vector machine (SVM) , un autre pour le modèle VGG19 et un pour la partie frontend de projet qui seront intégrés dans trois conteneurs différents Docker pour simplifier la gestion et l'accessibilité à travers une communication entre les différents containers pour assurer les fonctionnalités demandées par le mini projet .

Chapitre 1

Présentation du projet

Introduction

En cours de ce chapitre, on va représenter le travail demandé et les différentes technologies utilisées au cours de la réalisation de ce mini projet général sur l'architecture du projet et son fonctionnement.

1. Explication de travail demandé :

Étape 1 : création de l'architecture de projet

Organisez notre projet en créant des répertoires (app, model, tests, etc.) et des fichiers (svm_service.py, vgg19_service.py, Dockerfile, docker-compose.yml, requirements.txt).

Étape 2: Mise en place des services Flask

Définissez les services Flask :

Dans les fichiers svm_service.py et vgg19_service.py, écrivez le code Flask nécessaire pour créer des services web qui utiliseront les modèles de Machine Learning et deep learning pour classer les genres musicaux après le training en utilisant la base de données suivantes <https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre->.

Étape 3: Dockerization

Créez les fichiers Dockerfile pour les trois containers de svm vgg19 et de la partie frontend :

Le fichier Dockerfile est un script qui définit comment construire l'image Docker. Il commence par choisir une image de base, installe les dépendances nécessaires à partir de requirements.txt, copie les fichiers du projet dans le conteneur, et définit la commande pour lancer le service Flask.

Définissez les dépendances dans requirements.txt :

Enumérez les dépendances Python nécessaires, telles que Flask, scikit-learn, TensorFlow, etc., dans le fichier requirements.txt.

Étape 4: Docker Compose

Créez un fichier docker-compose.yml dans le fichier racine de notre projet :

Le fichier docker-compose.yml décrit les services, les réseaux, et les volumes à utiliser. Dans notre cas, il définit deux services, svm_service et vgg19_service, basés sur le Dockerfile, en spécifiant les ports, l'environnement et aussi les images pour chaque service.

Étape 5: Intégration Continue avec Jenkins

Configurez Jenkins :

Installez Jenkins sur un serveur accessible et assurez-vous que Docker est également installé. Installez le plugin Docker sur Jenkins pour faciliter l'intégration avec Docker.

Créez un nouveau projet Jenkins :

Dans l'interface Jenkins, créez un nouveau projet de type Freestyle ou Pipeline selon vos préférences.

Configurez le pipeline ou le travail :

Définissez les étapes du pipeline, comme récupérer le code depuis le référentiel, construire l'image Docker, pousser l'image sur un registre Docker, et exécuter les tests automatisés.

Déclenchez le travail à chaque modification du code :

Configurez Jenkins pour surveiller les changements dans votre référentiel Git et déclencher automatiquement le travail à chaque commit.

Étape 6: Exécution du projet

Exécutez le projet avec Docker Compose :

Utilisez la commande `docker-compose up` pour lancer les services définis dans le fichier `docker-compose.yml`. Cela créera les conteneurs Docker pour vos services Flask.

Ces étapes permettent la mise en place d'un environnement complet avec des services Flask dans des conteneurs Docker, orchestrés avec Docker Compose, et gérés de manière continue avec Jenkins. Chaque étape joue un rôle crucial dans la création d'un pipeline automatisé pour le déploiement et les tests de votre projet Machine Learning.

2. Les technologies utilisées :

- ❖ **Docker** : Une plate-forme de conteneurisation offrant une manière de créer, de distribuer et d'exécuter des applications de manière isolée et portable.
- ❖ **Python** : Langage de programmation utilisé pour le développement des modèles de Machine Learning, la création des services web avec Flask, et la manipulation des données.
- ❖ **Flask** : Framework web minimaliste en Python, idéal pour construire des services web RESTful et facilement intégrable avec d'autres bibliothèques Python.
- ❖ **Bibliothèques ML (ex. scikit-learn, TensorFlow/Keras)** : Utilisées pour la création, l'entraînement et le déploiement des modèles de classification tels que SVM (avec scikit-learn) et VGG19 (avec TensorFlow/Keras).

Conclusion

Le projet consiste à classifier les genres musicaux à partir d'une collection de données audio.

Chapitre 2

Architecture du projet

Introduction

L'architecture de l'application repose sur l'utilisation des conteneurs Docker pour héberger trois services Flask distincts, SVM_service et vgg19_service et la partie frontend. Ces services interagissent avec des modèles de classification (SVM et VGG19) après la réalisation de training de notre algorithme sur la base de données suivante de <https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification> pour effectuer une classification des genres musicaux à partir de fichiers audio en entrée. Le Docker-compose est utilisé pour orchestrer et gérer ces services dans un environnement cohérent.

1. L'architecture :

Voici l'architecture globale de mon projet réalisée :

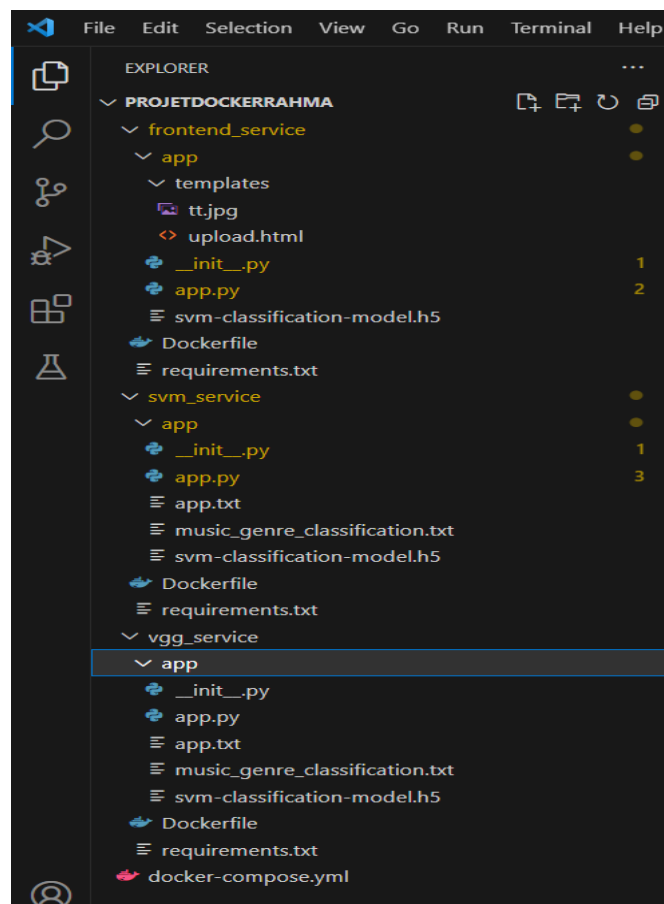


Figure 1: Architecture globale du projet

2. Les différents composants du projet :

Implémentation des services Flask

Détails sur la mise en œuvre des services SVM et VGG19 :

➤ **SVM_service :**

Ce service reçoit des fichiers audio au format base64, les traite pour en extraire des caractéristiques pertinentes, puis utilise le modèle SVM pour classifier le genre musical.

➤ **Vgg19_service :**

Similaire au SVM_service, ce service prend en charge les fichiers audio au format base64, extrait des caractéristiques spécifiques et utilise le modèle VGG19 (réseau neuronal convolutifs) pour la classification.

Création du Docker-compose

Explication du fichier Docker-compose :

Le fichier Docker-compose définit et orchestre les services et leurs dépendances. Il spécifie les images à utiliser, configure les services (SVM_service, vgg19_service) avec leurs ports, leurs volumes, et gère les réseaux pour la communication entre les conteneurs.

Configuration et interconnexion des services :

➤ **Réseaux Docker :**

Les services sont connectés à des réseaux Docker spécifiques pour permettre la communication entre eux.

➤ **Définition des services :**

Chaque service est configuré avec des paramètres spécifiques tels que les dépendances, les variables d'environnement et les ports d'exposition.

Création de l'image Docker

Processus de création de l'image Docker :

➤ Dockerfile :

Ce fichier spécifie les étapes pour construire l'image Docker, y compris l'installation des dépendances, la configuration des services Flask, et l'ajout des modèles de classification.

➤ Construction de l'image :

Les commandes Docker build sont utilisées pour construire l'image en se basant sur le Dockerfile.

Méthodes pour assurer la portabilité et la réalisabilité :

➤ Dépendances déclarées :

Toutes les dépendances nécessaires sont explicitement déclarées dans le Dockerfile pour assurer la reproductibilité de l'environnement.

➤ Utilisation de volumes :

Les volumes Docker peuvent être utilisés pour rendre persistants les données et assurer la portabilité de l'environnement entre différentes machines hôtes.

Conclusion

Ces détails offrent une vue approfondie de la conception et de l'implémentation de l'architecture, des services Flask, du Docker-compose et de la création d'image Docker pour garantir une compréhension complète du processus de développement et de déploiement de l'application .

Chapitre 3

Implémentation des composant de projet

Introduction

Dans cette partie on va voir en détails chaque service et son fonctionnement.

1. Le service de Frontend :

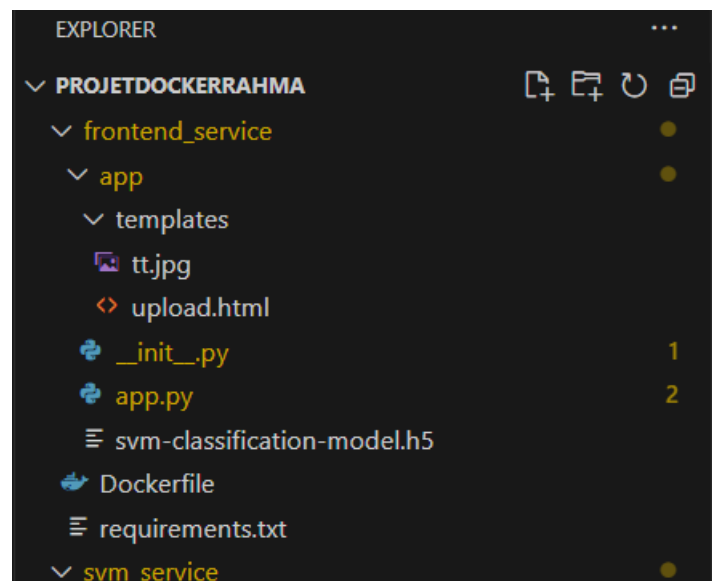


Figure 2: Architecture de la partie frontend

Le dossier service frontend contient un Dockerfile qui est destiné à la construction d'une image Docker pour un service frontend utilisant Python, probablement avec Flask pour développer des applications web.

Ce Dockerfile prépare l'environnement en copiant les fichiers du répertoire local dans le conteneur, installant les dépendances Python à partir du fichier requirements.txt, exposant le port 3000, et définissant la commande par défaut pour lancer l'application Flask via app.py

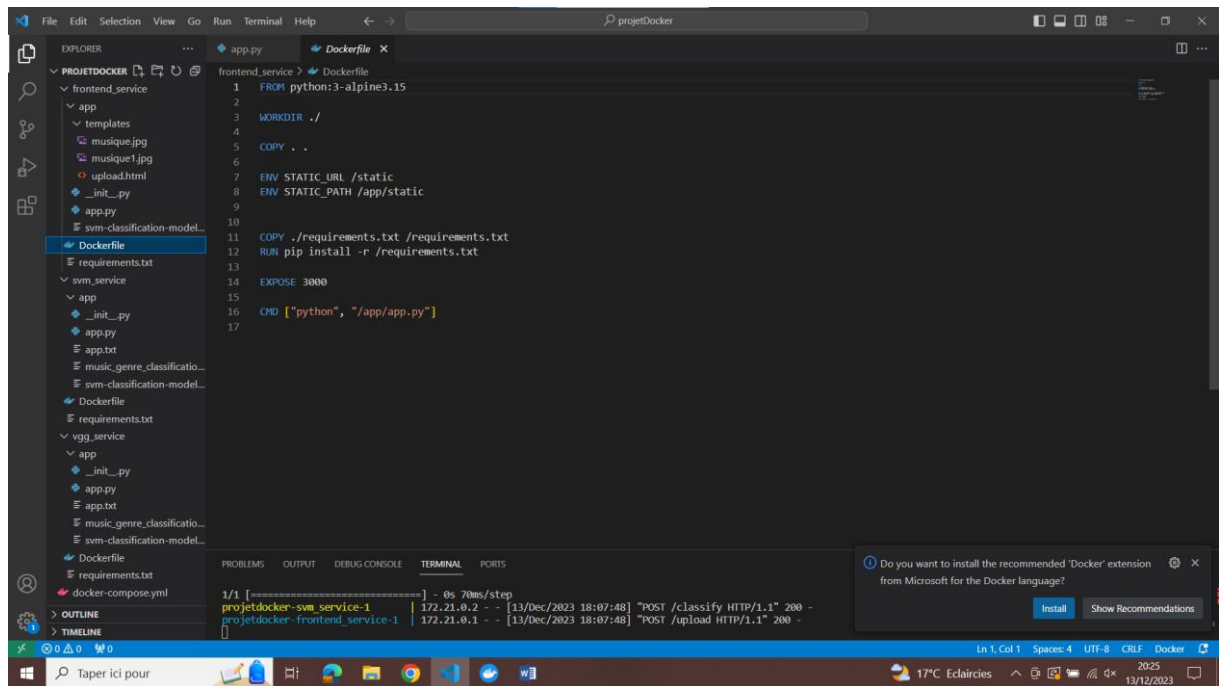


Figure 3: DockerFile du front

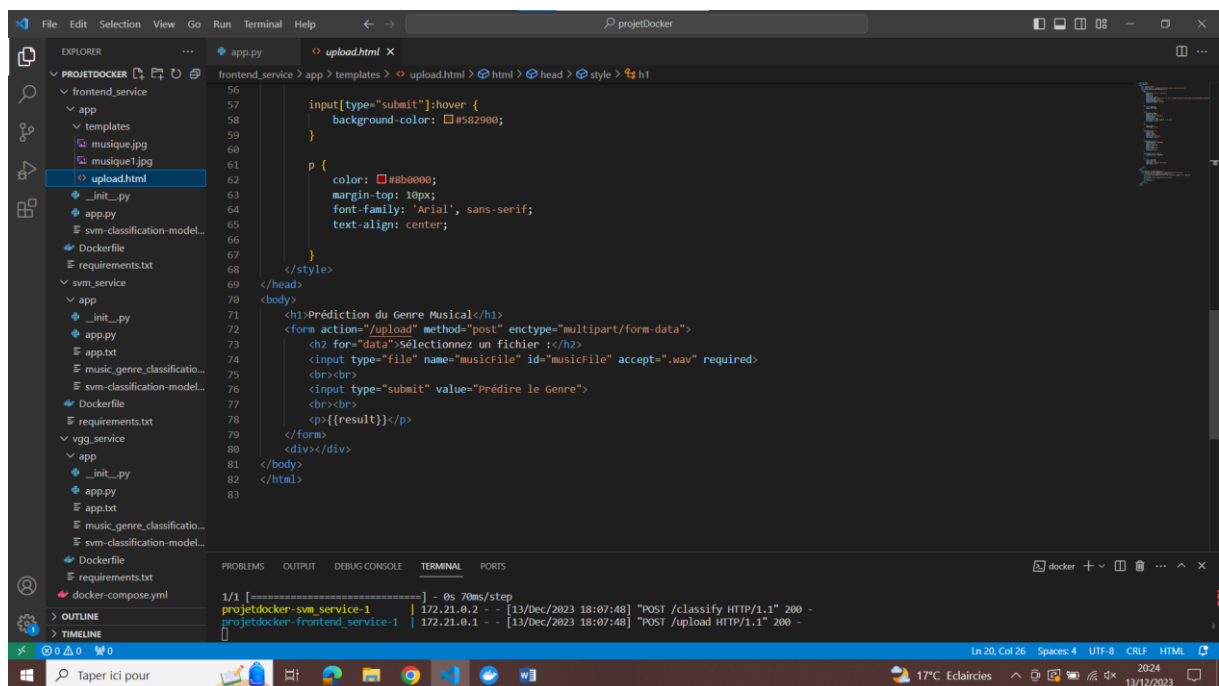


Figure 4 : Design du Front

Le fichier requirements.txt est un fichier souvent utilisé dans les projets Python pour spécifier les dépendances nécessaires à l'application. Chaque ligne de ce fichier représente un package Python spécifique et éventuellement sa version.

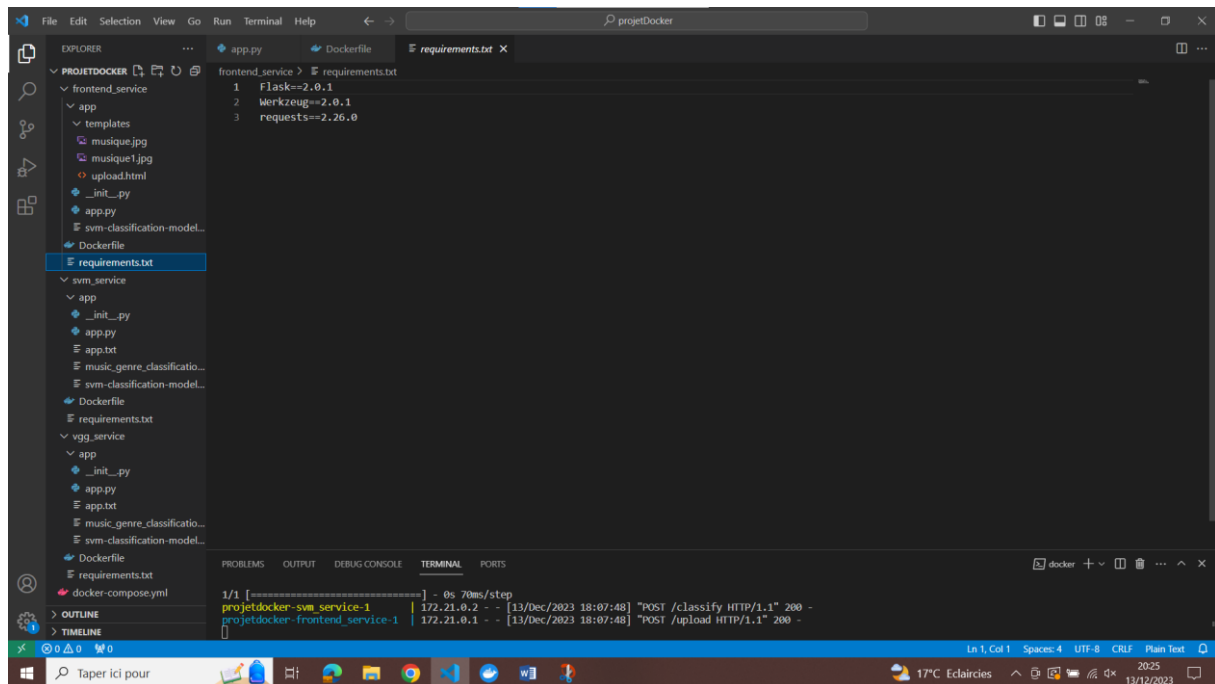


Figure 5 : Contenu de Requirement du Front

2. Le service de SVM:

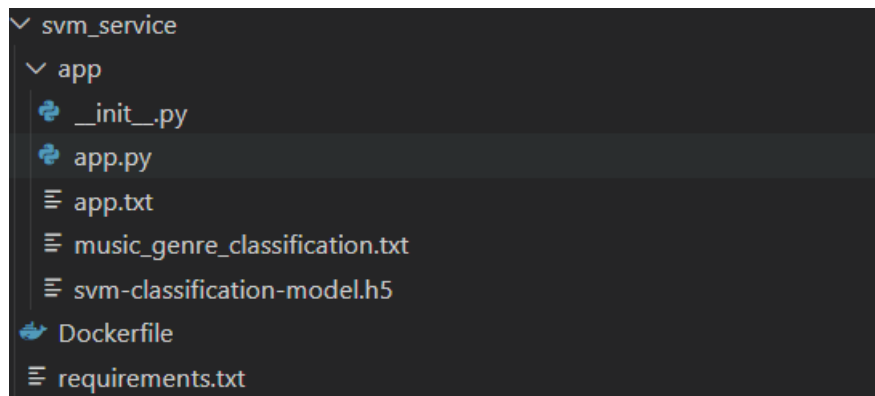


Figure 6 : Architecture de service SVM

Ce code représente un service web Flask nommé svm_service qui utilise un modèle pré-entraîné pour classer le genre musical d'un fichier audio WAV.

Ce service web Flask expose une route (/classify) qui permet de recevoir un fichier audio, de le classer à l'aide d'un modèle pré-entraîné de classification musicale SVM et de renvoyer le genre musical prédit dans une réponse JSON.

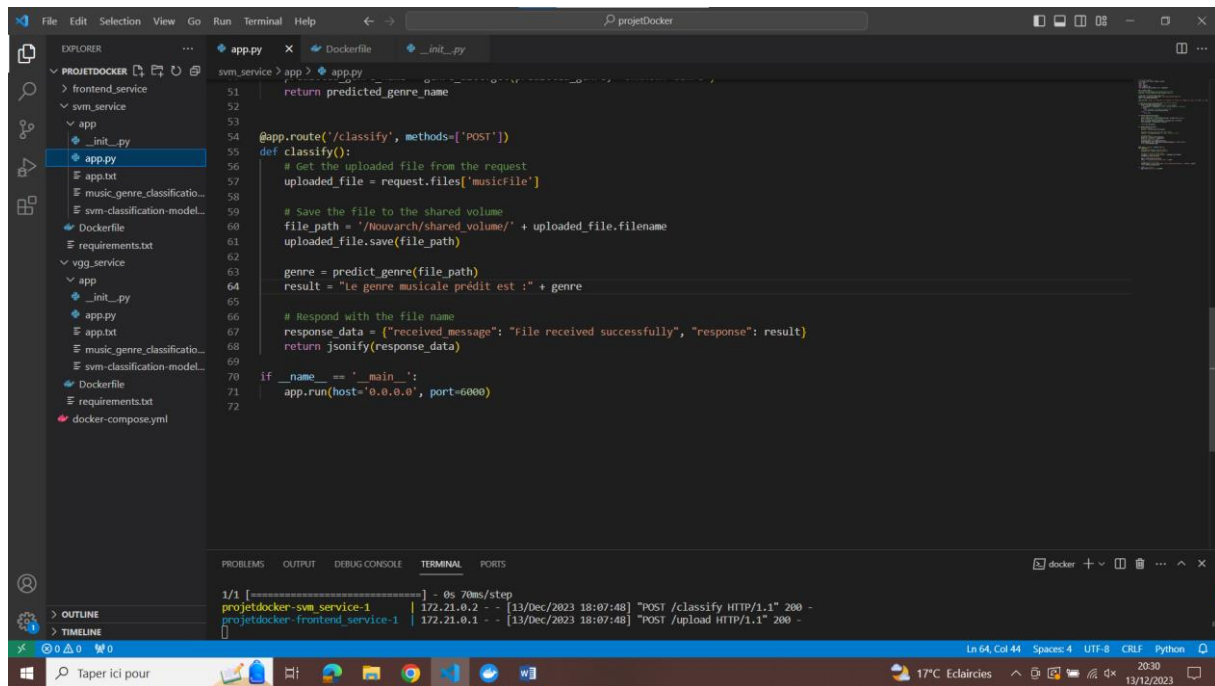


Figure 7 : Contenu de fichier app.py

Ce fichier Dockerfile est utilisé pour créer une image Docker qui va exécuter un service Python pour la classification des genres musicaux basée sur un modèle SVM (Support Vector Machine) via Flask.

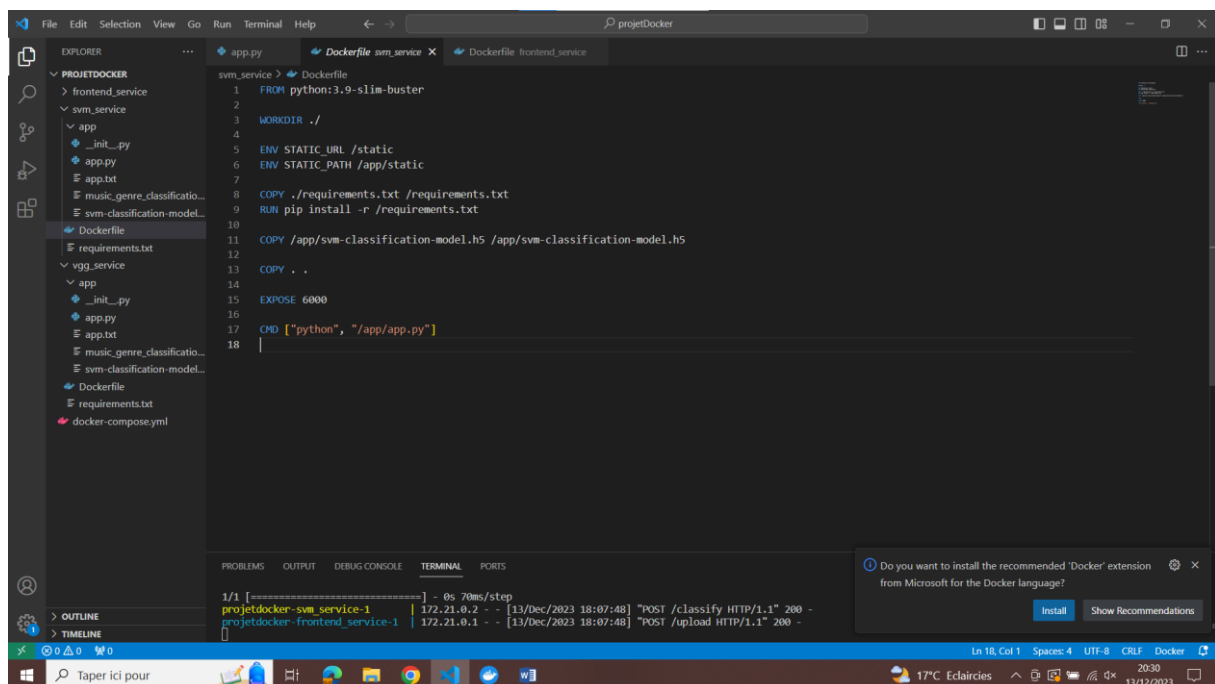


Figure 8 : Contenu de DockerFile du SVM

Ce fichier __init__.py est utilisé pour initialiser une application Flask dans un package Python appelé app. Ce fichier est utilisé pour créer une instance d'application Flask (app) et démarrer l'application Flask lorsque ce fichier est exécuté directement.

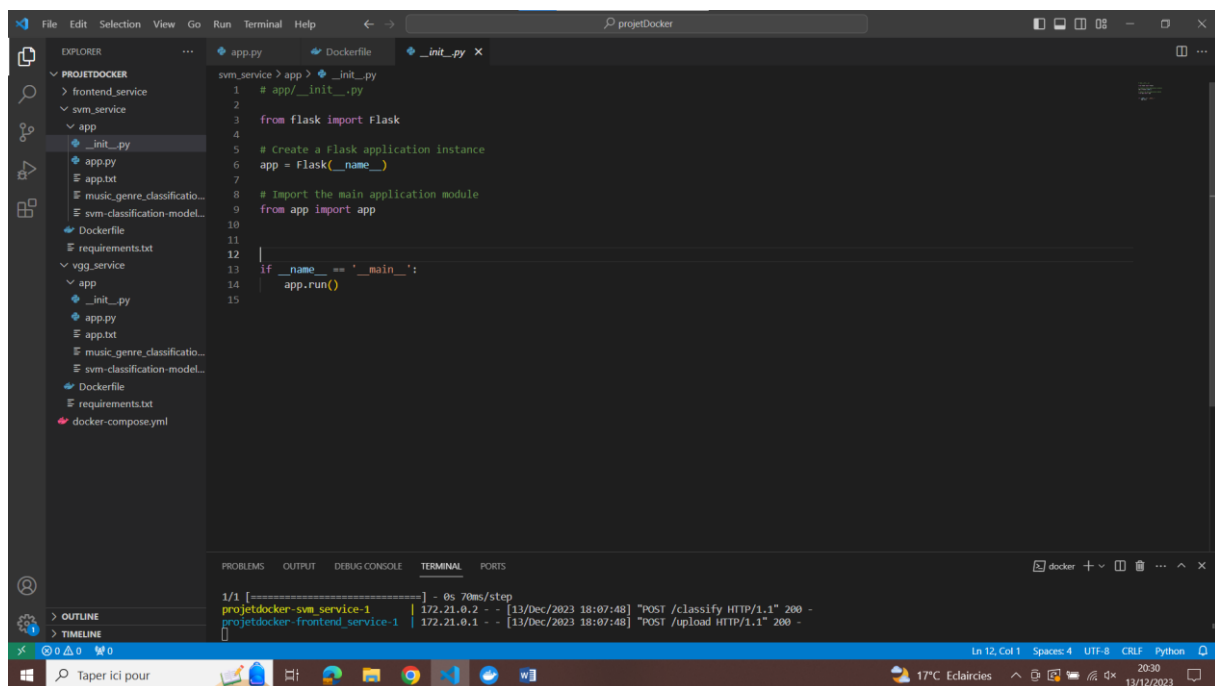


Figure 9 : Contenu de fichier init.py

Ce fichier, nommé requirements.txt, est un fichier standard utilisé dans les projets Python pour répertorier les dépendances nécessaires à l'exécution de l'application. Chaque ligne de ce fichier représente un module Python spécifique et éventuellement une version précise nécessaire pour le bon fonctionnement de l'application.

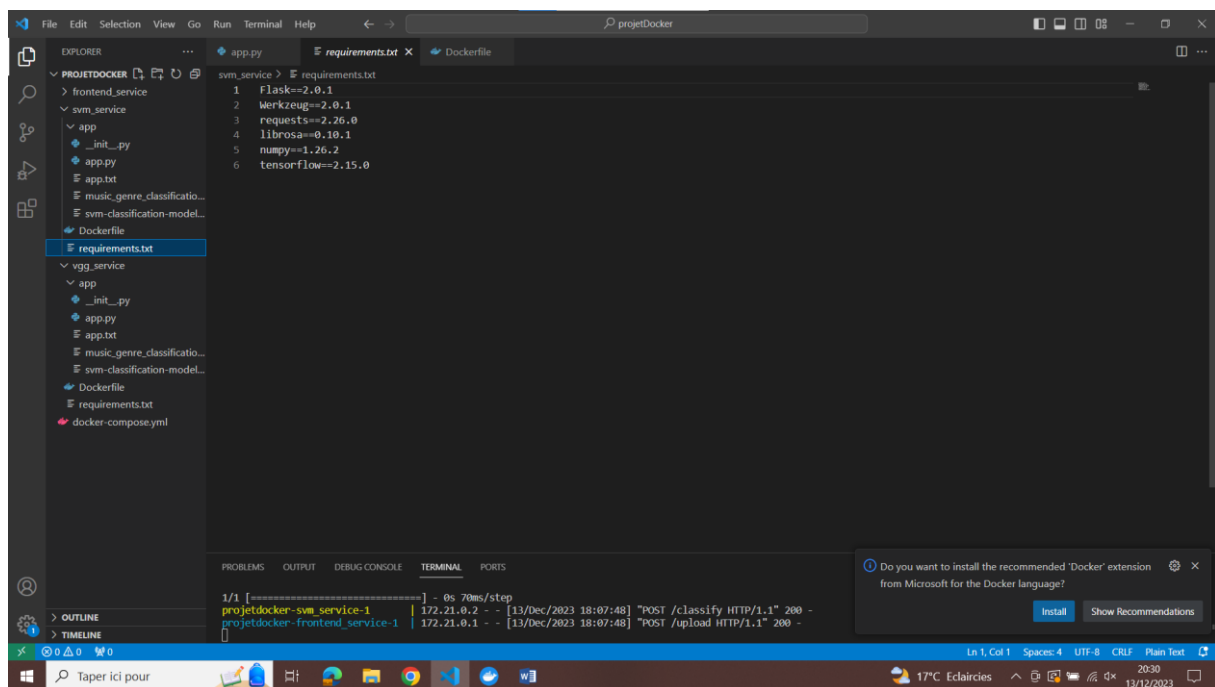


Figure 10 : Contenu de fichier requirement du SVM

3. Le service de VGG :

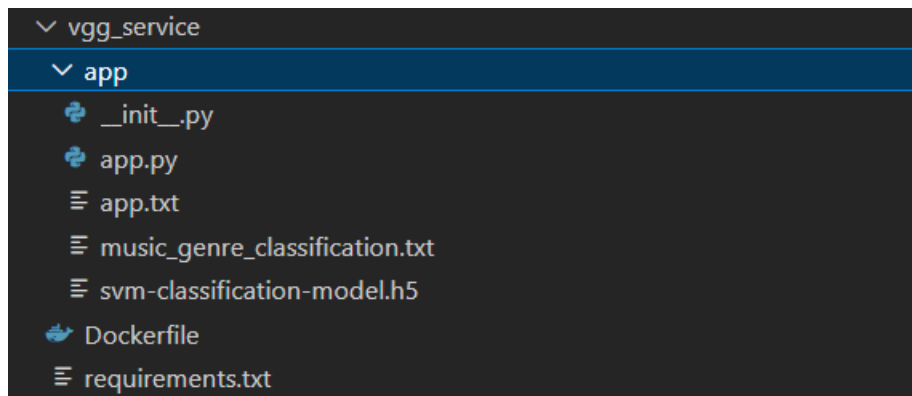


Figure 11 : Architecture de VGG_Service

Ce Dockerfile est utilisé pour construire une image Docker qui hébergera un service VGG pour la classification des genres musicaux basée sur un modèle VGG19 via Flask.

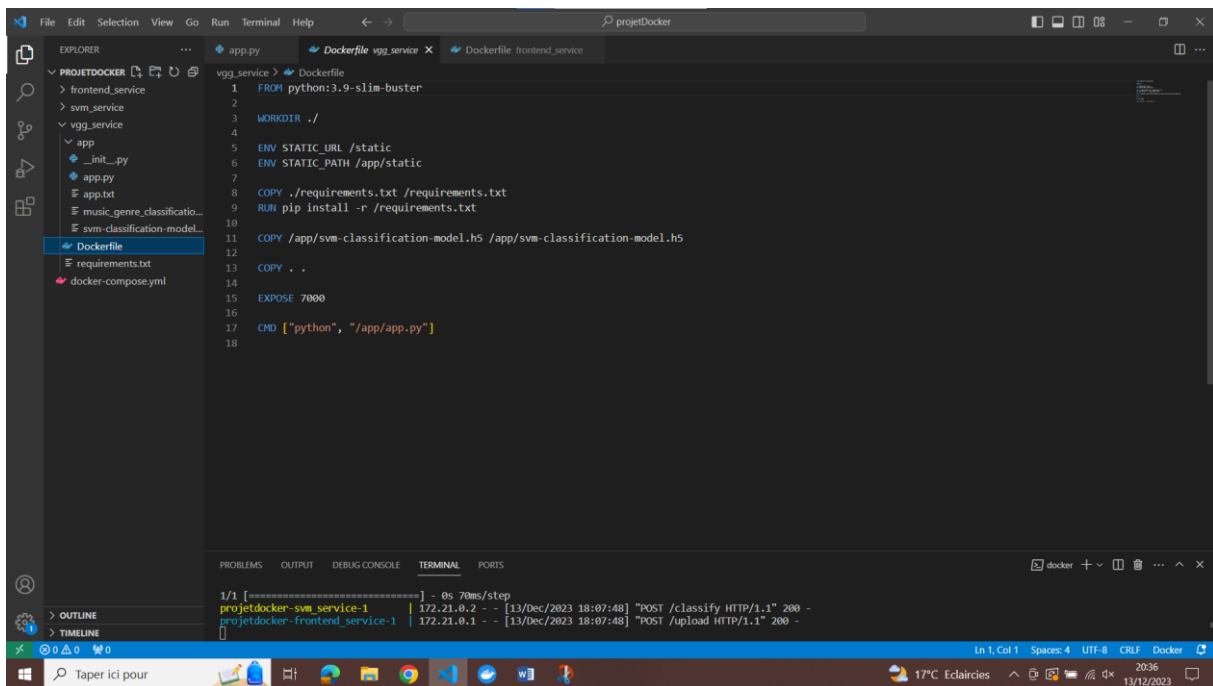


Figure 12: Continue de DockerFile du VGG

Ce fichier requirements.txt spécifie les dépendances nécessaires à l'exécution du service VGG pour la classification des genres musicaux via Flask. Chaque ligne représente une bibliothèque Python spécifique avec une version précise nécessaire pour le bon fonctionnement du service.

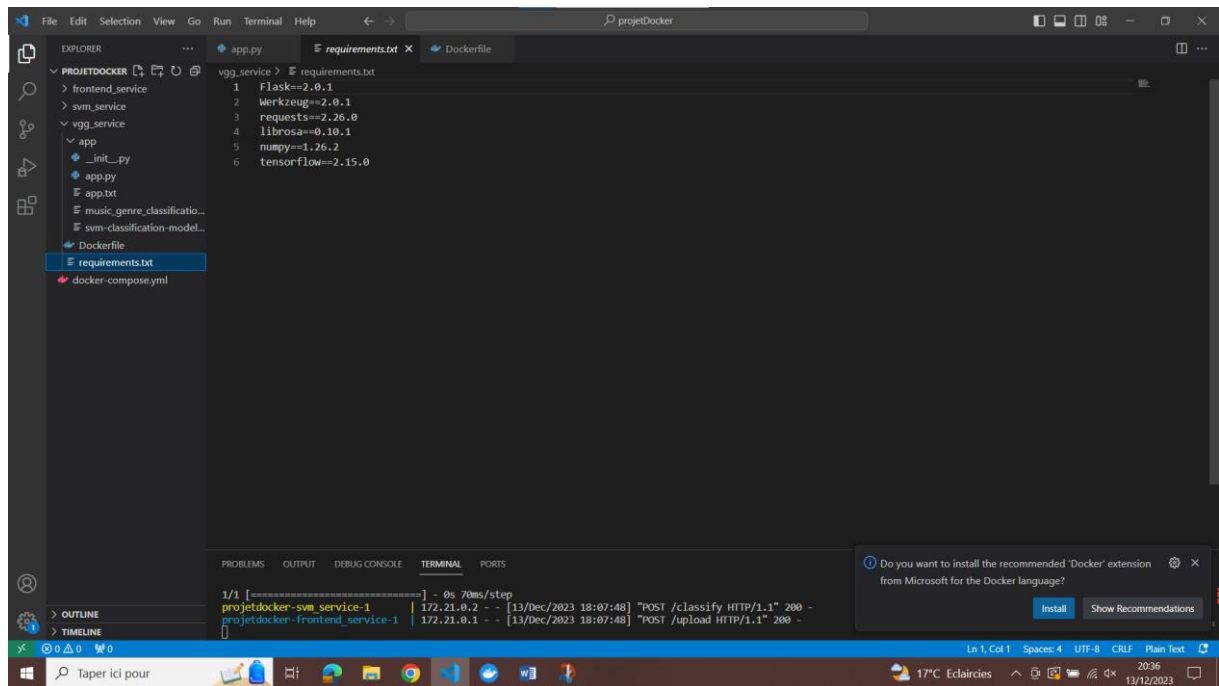


Figure 13 : Contenu de requirement du VGG

Ce fichier `__init__.py` est utilisé pour initialiser une application Flask à l'intérieur d'un package Python nommé `app`.

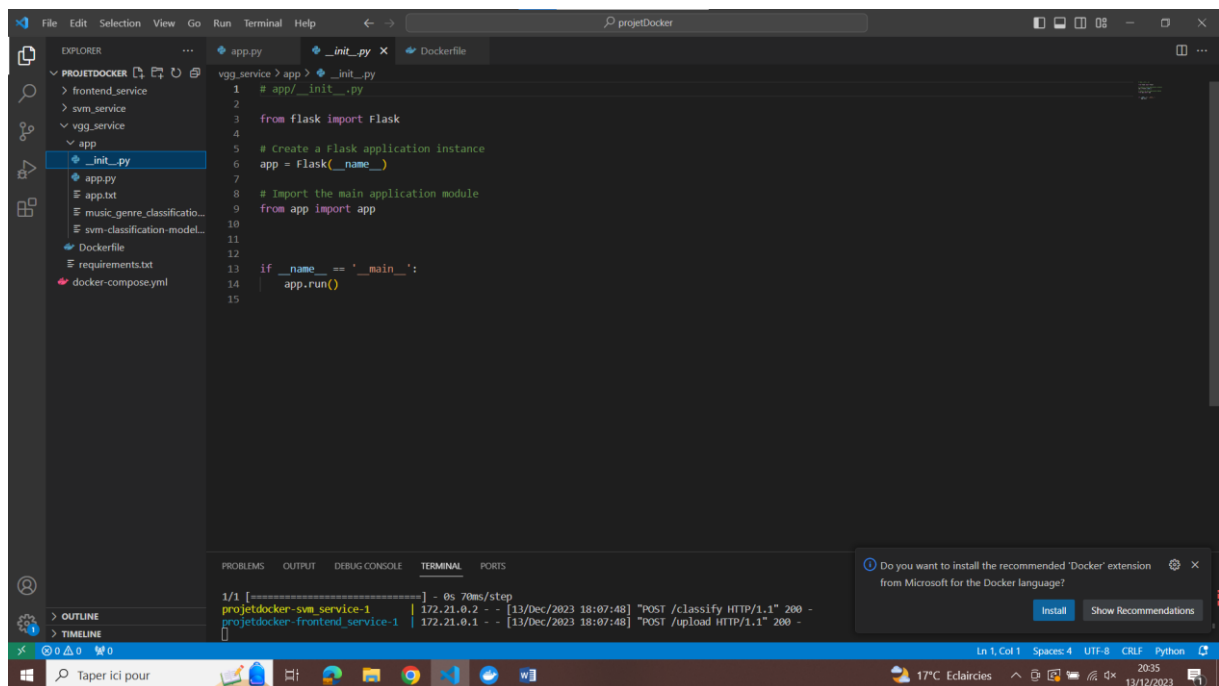
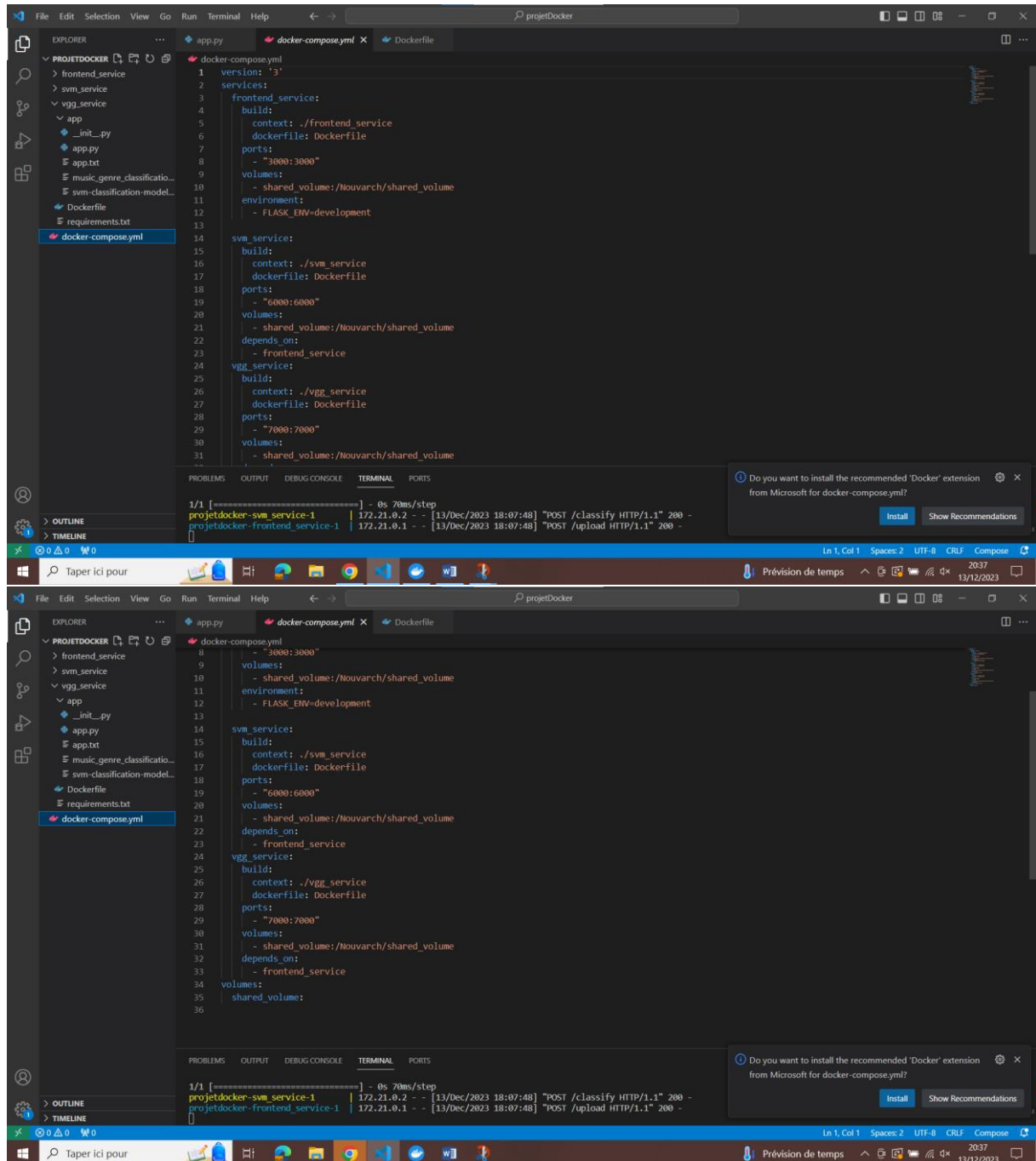


Figure 14 : Contenu de fichier init du VGG

Ce fichier DockerCompose.yml est utilisé pour décrire et configurer un ensemble de services basés sur Docker qui fonctionnent ensemble pour créer une application complète.

ce fichier Docker Compose décrit la configuration de trois services Docker (frontend_service, svm_service, vgg_service) et détaille comment construire, exposer des ports, partager des volumes, définir des variables d'environnement et spécifier les dépendances entre ces services pour créer une application distribuée et fonctionnelle.



The image shows two screenshots of the Visual Studio Code editor. The top screenshot displays the `docker-compose.yml` file with the following content:

```
1 version: '3'
2 services:
3   frontend_service:
4     build:
5       context: ../frontend_service
6       dockerfile: Dockerfile
7     ports:
8       - "3000:3000"
9     volumes:
10      - shared_volume:/Nouvarch/shared_volume
11     environment:
12      - FLASK_ENV=development
13
14   svm_service:
15     build:
16       context: ../svm_service
17       dockerfile: Dockerfile
18     ports:
19       - "6000:6000"
20     volumes:
21      - shared_volume:/Nouvarch/shared_volume
22     depends_on:
23      - frontend_service
24
25   vgg_service:
26     build:
27       context: ../vgg_service
28       dockerfile: Dockerfile
29     ports:
30       - "7000:7000"
31     volumes:
32      - shared_volume:/Nouvarch/shared_volume
```

The bottom screenshot shows the same file with the terminal output of the `docker-compose up` command:

```
1/1 [=====] - 0s 70ms/step
projctdocker-svm_service-1 | 172.21.0.2 - - [13/Dec/2023 18:07:48] "POST /classify HTTP/1.1" 200 -
projctdocker-frontend_service-1 | 172.21.0.1 - - [13/Dec/2023 18:07:48] "POST /upload HTTP/1.1" 200 -
```

Figure 15 : Conteneur de fichier DockerCompose.YML du projet

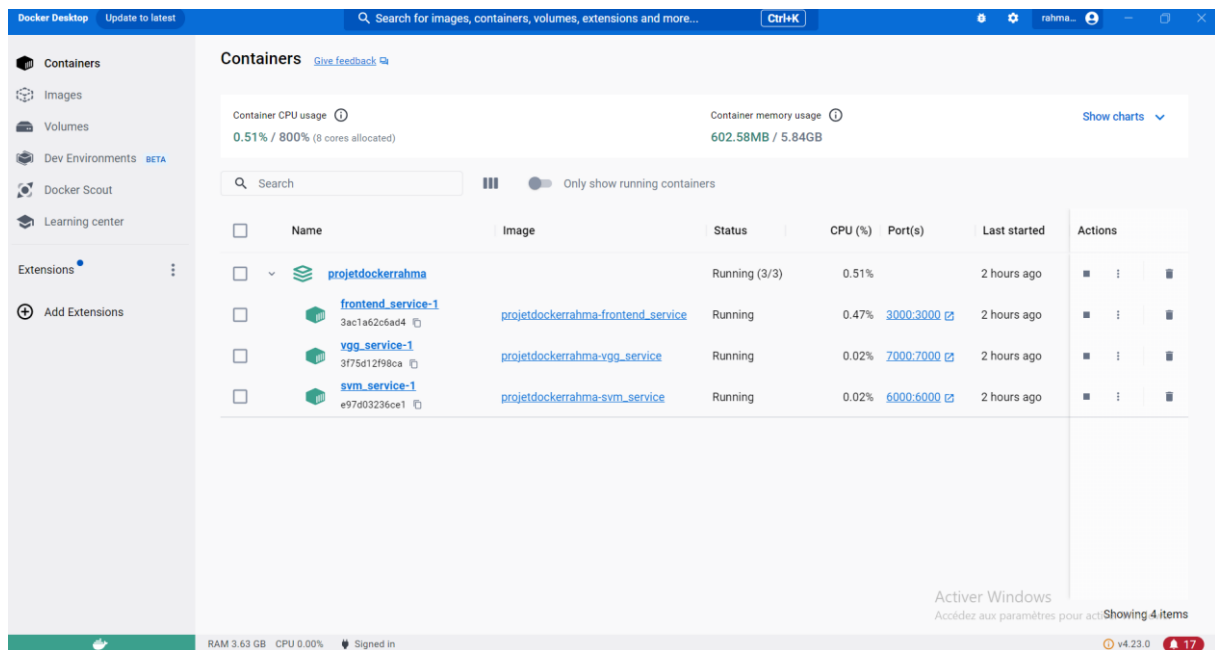


Figure 16-figure de container créer

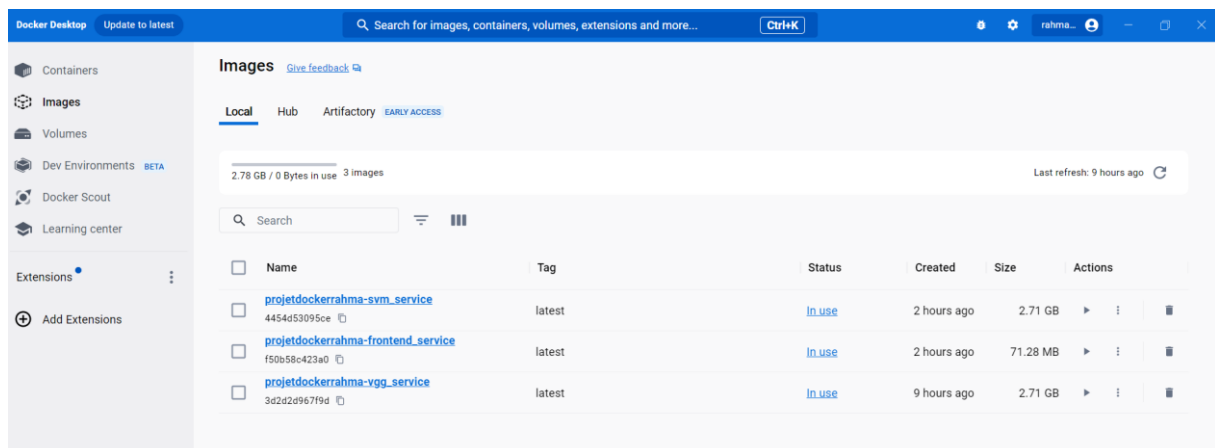


Figure 17-Figure création image

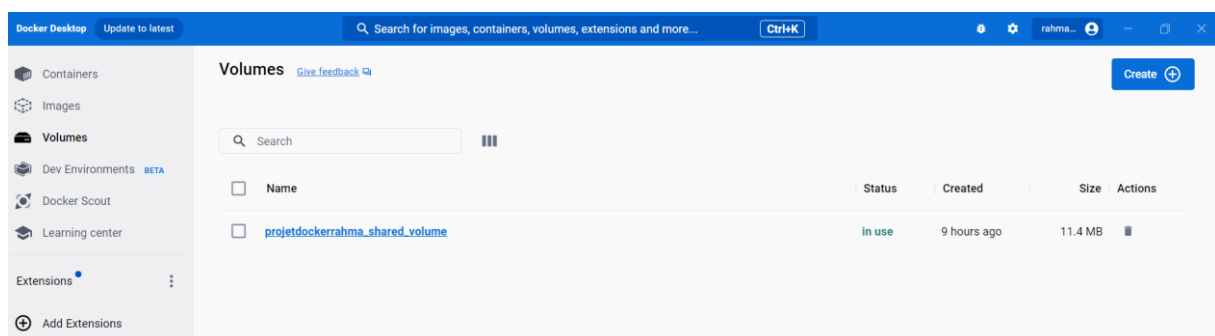


Figure 18-figure création volume

Conclusion

Ces détails offrent une vue approfondie de la conception et de l'implémentation de l'architecture, des services Flask, du Docker-compose et de la création d'image Docker pour garantir une compréhension complète du processus de développement et de déploiement de l'application

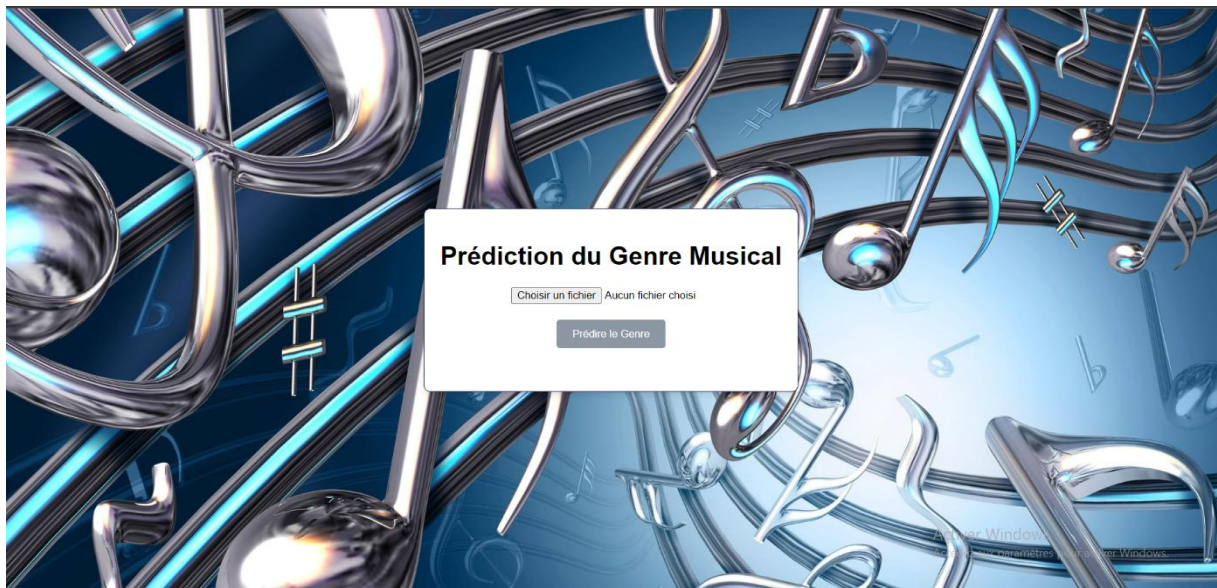
Chapitre 4

Réalisation

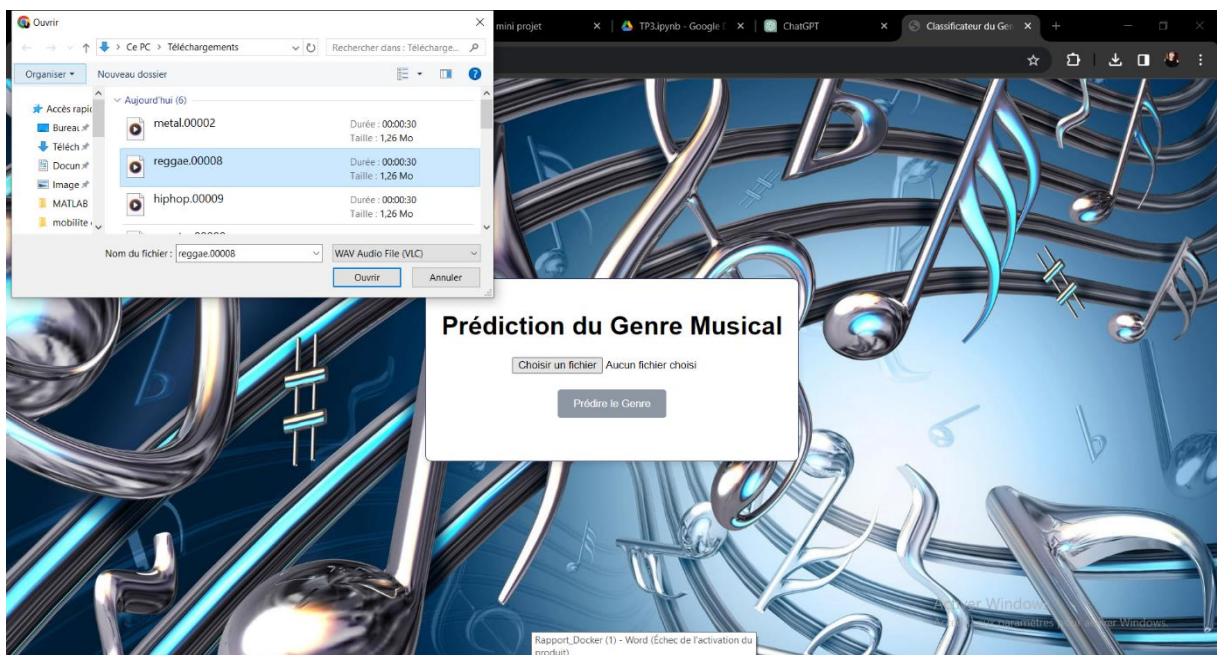
1. Scenario d'exécution :

L'ensemble des captures suivante vont mettre en évidence un cycle d'exécution d'insertion de fichier audio et de prédire son genre musical :

Voici l'interface initiale de projet



Si on clique sur le bouton « Choisir un fichier », voici ce que nous affiche pour choisir un fichier wax (musique) pour prédire son genre musical.





Après le clique sur le bouton « Prédire le genre » du fichier sélectionné, voici ce que nous affiche le résultat de la prédiction.

