# run using celery -A celery\_chroma\_ingest worker --loglevel=INFO --pool=solo

# C:\\Users\\shrimantas\\Downloads\\project\_redis\\uploads

# celery -A celery\_chroma\_ingest worker --beat --loglevel=INFO --pool=solo\

#https://celery.school/celery-on-windows

#https://stackoverflow.com/questions/45744992/celery-raises-valueerror-not-enough-values-to-unpack

import os

import datetime

from celery import Celery

from celery.schedules import crontab

import chromadb

from chromadb.config import Settings

from langchain\_community.embeddings import HuggingFaceEmbeddings

from langchain.vectorstores.chroma import Chroma

from langchain\_core.documents.base import Document

import PyPDF2

from redis import Redis

import logging

# Initialize text splitter and embeddings

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain\_community.embeddings import HuggingFaceEmbeddings

logger = logging.getLogger(\_\_name\_\_)

# Celery setup

# redis = Redis(host="20.41.249.147", port=6379, username="default", password="admin", db=0)

app = Celery('celery\_chroma\_ingest', broker='redis://127.0.0.1:6379/0', backend="redis://127.0.0.1:6379/0")

# print(app)

# app.conf.enable\_utc = False

# app = Celery('tasks')

# app.config\_from\_object('tasks.config')

# Initialize Chroma DB client

chroma\_client = chromadb.PersistentClient(path="vector\_storage")

collection = chroma\_client.get\_or\_create\_collection(name="pdf\_database")

# print(collection)

# PDF to text

def pdf\_to\_text(file\_path):

    pdf\_file = open(file\_path, 'rb')

    pdf\_reader = PyPDF2.PdfReader(pdf\_file)

    text = ""

    for page\_num in range(len(pdf\_reader.pages)):

        text += pdf\_reader.pages[page\_num].extract\_text()

    pdf\_file.close()

    return text

# Initialize text splitter and embeddings

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=100)

embeddings = HuggingFaceEmbeddings(model\_name="BAAI/bge-large-en-v1.5")

# print(embeddings)

# Process each PDF in the ./uploads directory

# @app.task(name="pdf\_chroma\_ingest.process\_files")

@app.task

def process\_files():

    uploads\_dir = 'uploads'

    for filename in os.listdir(uploads\_dir):

        if filename.endswith('.pdf'):

            file\_path = os.path.join(uploads\_dir, filename)

            # Split text into chunks

            text = pdf\_to\_text(file\_path)

            chunks = text\_splitter.split\_text(text)

            # Convert chunks to vector representations and store in Chroma DB

            documents\_list = []

            embeddings\_list = []

            ids\_list = []

            for i, chunk in enumerate(chunks):

                vector = embeddings.embed\_query(chunk)

                documents\_list.append(chunk)

                embeddings\_list.append(vector)

                ids\_list.append(f"{filename}\_{i}")

            collection.add(embeddings=embeddings\_list,

                           documents=documents\_list,

                           ids=ids\_list)

            logger.info(f"Stored {filename} in vector database")

            print("------")

            print(collection)

# Register the task with the Celery app

app.tasks.register(process\_files)

if \_\_name\_\_ == "\_\_main\_\_":

    app.start()

# if \_\_name\_\_ == "\_\_main\_\_":

#     process\_files.delay()

# run using celery -A celery\_chroma\_ingest worker --loglevel=INFO --pool=solo

# C:\\Users\\shrimantas\\Downloads\\project\_redis\\uploads

# celery -A celery\_chroma\_ingest worker --beat --loglevel=INFO --pool=solo

from fastapi import FastAPI, File, UploadFile

import os

from celery\_chroma\_ingest import process\_files, collection

from fastapi.middleware.cors import CORSMiddleware

from celery.result import AsyncResult

app = FastAPI()

# Configure CORS

origins = [

    "http://localhost",

    "http://localhost:3000",

    "http://localhost:8000"

]

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=origins,

    allow\_credentials=True,

    allow\_methods=["GET", "POST", "OPTIONS"],

    allow\_headers=["\*"],

)

app = FastAPI()

@app.post("/upload")

async def upload\_files(files: list[UploadFile] = File(...)):

    uploads\_dir = 'uploads'

    uploads\_dir\_path = os.path.abspath(uploads\_dir)

    # Check if the uploads directory exists, create it if not

    if not os.path.exists(uploads\_dir):

      os.makedirs(uploads\_dir)

    for file in files:

        file\_path = os.path.join(uploads\_dir, file.filename)

        with open(file\_path, "wb") as buffer:

            buffer.write(file.file.read())

    # Trigger the Celery task to process the new files

    # task = process\_files.delay()

    # task\_id = task.id

    # async\_result = AsyncResult(task\_id, app=process\_files)

    # task\_status = async\_result.status

    # task\_eta = async\_result.date\_done

    # return {"message": "Files uploaded and processing started",

    #         "task\_id": task\_id,

    #         "eta": str(task\_eta),

    #         "status": task\_status

    #         }

    task = process\_files.delay()

    task\_id = task.id

    async\_result = AsyncResult(task\_id, app=process\_files)

    task\_status = async\_result.status

    task\_time = async\_result.date\_done

    if task\_status == 'SUCCESS':

        return {

            "message": "Files uploaded and processing started",

            "task\_id": task\_id,

            "status": task\_status,

            "time": task\_time.isoformat() if task\_time else None,

            "collection\_name": collection.name,

            "uploads\_dir": uploads\_dir\_path

        }

    else:

        return {

            "message": "Files uploaded and processing started",

            "task\_id": task\_id,

            "status": task\_status,

            "time": None,

            "collection\_name": collection.name,

            "uploads\_dir": uploads\_dir\_path

        }

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run(app, host="0.0.0.0", port=8000)

# from fastapi import FastAPI, File, UploadFile

# import os

# from celery\_chroma\_ingest import process\_files, collection

# from fastapi.middleware.cors import CORSMiddleware

# from celery.result import AsyncResult

# app = FastAPI()

# # Configure CORS

# origins = [

#     "http://localhost",

#     "http://localhost:3000",

#     "http://localhost:8000"

# ]

# app.add\_middleware(

#     CORSMiddleware,

#     allow\_origins=origins,

#     allow\_credentials=True,

#     allow\_methods=["GET", "POST", "OPTIONS"],

#     allow\_headers=["\*"],

# )

# @app.post("/upload")

# async def upload\_files(files: list[UploadFile] = File(...)):

#     uploads\_dir = 'uploads'

#     uploads\_dir\_path = os.path.abspath(uploads\_dir)

#     # Create uploads dir if not exists

#     if not os.path.exists(uploads\_dir):

#         os.makedirs(uploads\_dir)

#     responses = []

#     for file in files:

#         file\_path = os.path.join(uploads\_dir, file.filename)

#         with open(file\_path, "wb") as buffer:

#             buffer.write(file.file.read())

#         # Trigger the Celery task to process the new file

#         task = process\_files.delay()

#         task\_id = task.id

#         async\_result = AsyncResult(task\_id, app=process\_files)

#         # Wait for previous task completion before triggering the next one

#         async\_result.get()

#         task\_status = async\_result.status

#         task\_time = async\_result.date\_done

#         response = {

#             # "message": "File uploaded and processing started",

#             "task\_id": task\_id,

#             "status": task\_status,

#             # "time": task\_time.isoformat() if task\_time else None,

#             "time": task\_time if task\_time else None,

#             "collection\_name": collection.name,

#             "uploads\_dir": uploads\_dir\_path,

#             "file\_name": file.filename,

#             "result": async\_result.result,

#             "successful": async\_result.successful(),

#         }

#         responses.append(response)

#     return responses

# if \_\_name\_\_ == "\_\_main\_\_":

#     import uvicorn

#     uvicorn.run(app, host="0.0.0.0", port=8000)

# from fastapi import FastAPI, File, UploadFile

# import os

# from celery\_chroma\_ingest import process\_files, collection

# from fastapi.middleware.cors import CORSMiddleware

# from celery.result import AsyncResult

# from sqlalchemy import create\_engine, Column, Integer, String, Text, DateTime

# from sqlalchemy.orm import declarative\_base, sessionmaker

# from datetime import datetime

# Base = declarative\_base()

# class Response(Base):

#     \_\_tablename\_\_ = 'responses'

#     id = Column(Integer, primary\_key=True)

#     task\_id = Column(String)

#     status = Column(String)

#     time = Column(DateTime)

#     collection\_name = Column(String)

#     uploads\_dir = Column(String)

#     file\_name = Column(String)

#     result = Column(Text)

#     successful = Column(Integer)

# engine = create\_engine('sqlite:///responses.db', echo=True)

# Base.metadata.create\_all(bind=engine)

# Session = sessionmaker(bind=engine)

# app = FastAPI()

# # Configure CORS

# origins = [

#     "http://localhost",

#     "http://localhost:3000",

#     "http://localhost:8000"

# ]

# app.add\_middleware(

#     CORSMiddleware,

#     allow\_origins=origins,

#     allow\_credentials=True,

#     allow\_methods=["GET", "POST", "OPTIONS"],

#     allow\_headers=["\*"],

# )

# @app.post("/data\_ingestion")

# async def upload\_files(files: list[UploadFile] = File(...)):

#     uploads\_dir = 'uploads'

#     uploads\_dir\_path = os.path.abspath(uploads\_dir)

#     # Create uploads dir if not exists

#     if not os.path.exists(uploads\_dir):

#         os.makedirs(uploads\_dir)

#     responses = []

#     for file in files:

#         file\_path = os.path.join(uploads\_dir, file.filename)

#         with open(file\_path, "wb") as buffer:

#             buffer.write(file.file.read())

#         # Trigger the Celery task to process the new file

#         task = process\_files.delay()

#         task\_id = task.id

#         async\_result = AsyncResult(task\_id, app=process\_files)

#         # Wait for previous task completion before triggering the next one

#         async\_result.get()

#         task\_status = async\_result.status

#         task\_time = async\_result.date\_done

#         response = {

#             "task\_id": task\_id,

#             "status": task\_status,

#             "time": task\_time if task\_time else None,

#             "collection\_name": collection.name,

#             "uploads\_dir": uploads\_dir\_path,

#             "file\_name": file.filename,

#             "result": async\_result.result,

#             "successful": async\_result.successful(),

#         }

#         responses.append(response)

#     session = Session()

#     for response in responses:

#         db\_response = Response(

#             task\_id=response['task\_id'],

#             status=response['status'],

#             time=response['time'],

#             collection\_name=response['collection\_name'],

#             uploads\_dir=response['uploads\_dir'],

#             file\_name=response['file\_name'],

#             result=str(response['result']),

#             successful=int(response['successful'])

#         )

#         session.add(db\_response)

#     session.commit()

#     session.close()

#     return responses

# @app.get("/responses")

# def get\_responses():

#     session = Session()

#     responses = session.query(Response).all()

#     session.close()

#     return [

#         {

#             'id': response.id,

#             'task\_id': response.task\_id,

#             'status': response.status,

#             'time': response.time.isoformat() if response.time else None,

#             'collection\_name': response.collection\_name,

#             'uploads\_dir': response.uploads\_dir,

#             'file\_name': response.file\_name,

#             'result': response.result,

#             'successful': bool(response.successful)

#         }

#         for response in responses

#     ]

# if \_\_name\_\_ == "\_\_main\_\_":

#     import uvicorn

#     uvicorn.run(app, host="0.0.0.0", port=8000)

# # https://kpmgindia365-my.sharepoint.com/personal/shrimantas\_kpmg\_com/\_layouts/15/onedrive.aspx?e=5%3A6676caee2ee04fe6a4cf75a2375de76c&sharingv2=true&fromShare=true&at=9&cid=5a8dd1ee%2D9172%2D4020%2D9b90%2D48bb329c6742&FolderCTID=0x01200026AA47FA986B9A4AA04737D4BACEAB9E&id=%2Fpersonal%2Fshrimantas%5Fkpmg%5Fcom%2FDocuments%2Fcelery%5Ffiles&view=0

import os

import datetime

from celery import Celery

from celery.schedules import crontab

import chromadb

from chromadb.config import Settings

from langchain\_community.embeddings import HuggingFaceEmbeddings

from langchain.vectorstores.chroma import Chroma

from langchain\_core.documents.base import Document

import PyPDF2

from redis import Redis

import logging

from celery.result import AsyncResult

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain\_community.embeddings import HuggingFaceEmbeddings

import docx

logger = logging.getLogger(\_\_name\_\_)

# Celery setup

redis = Redis(host="20.41.249.147", port=6379, username="default", password="admin", db=0)

# app = Celery('celery\_chroma\_ingest', broker='redis://127.0.0.1:6379/0', backend="redis://127.0.0.1:6379/0")

app = Celery('celery\_chroma\_ingest', broker='redis://20.41.249.147:6379/0', backend="redis://20.41.249.147:6379/0")

# Initialize Chroma DB client

chroma\_client = chromadb.PersistentClient(path="chromadb")

collection = chroma\_client.get\_or\_create\_collection(name="pdf\_database")

def get\_text(file\_path):

    if file\_path.endswith('.pdf'):

        pdf\_file = open(file\_path, 'rb')

        pdf\_reader = PyPDF2.PdfReader(pdf\_file)

        text = ""

        for page\_num in range(len(pdf\_reader.pages)):

            text += pdf\_reader.pages[page\_num].extract\_text()

        pdf\_file.close()

        text = text

    elif file\_path.endswith('.txt'):

        with open(file\_path, 'r', encoding='utf-8') as file:

            text = file.read()

    elif file\_path.endswith('.docx'):

        doc = docx.Document(file\_path)

        text = ""

        for para in doc.paragraphs:

            text += para.text + "\n"

    else:

        text = ""

    return text

# Initialize text splitter and embeddings

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=100)

embeddings = HuggingFaceEmbeddings(model\_name="BAAI/bge-large-en-v1.5")

# embeddings = HuggingFaceEmbeddings(model\_name="intfloat/e5-base-v2")

@app.task

def process\_files():

    uploads\_dir = 'uploads'

    #tracking already processed files

    files\_processed = set()

    for filename in os.listdir(uploads\_dir):

        if filename.endswith('.pdf') or filename.endswith(".txt") or filename.endswith(".docx"):

            file\_path = os.path.join(uploads\_dir, filename)

            if file\_path not in files\_processed:

                # Split text into chunks

                text = get\_text(file\_path)

                chunks = text\_splitter.split\_text(text)

                # Convert chunks to vector representations and store in Chroma DB

                documents\_list = []

                embeddings\_list = []

                ids\_list = []

                for i, chunk in enumerate(chunks):

                    vector = embeddings.embed\_query(chunk)

                    documents\_list.append(chunk)

                    embeddings\_list.append(vector)

                    ids\_list.append(f"{filename}\_{i}")

                try:

                    collection.add(embeddings=embeddings\_list,

                                   documents=documents\_list,

                                   ids=ids\_list)

                except chromadb.errors.ChromaDBError as e:

                    logger.warning(f"Error adding embeddings for {filename}: {e}")

                logger.info(f"Stored {filename} in vector database")

                files\_processed.add(file\_path)  # Add processed file to the set

                print("------")

                print(collection)

    task\_id = process\_files.request.id

    return task\_id

# Register the task with the Celery app

app.tasks.register(process\_files)

if \_\_name\_\_ == "\_\_main\_\_":

    app.start()

# import os

# import datetime

# from celery import Celery

# from celery.schedules import crontab

# import chromadb

# from chromadb.config import Settings

# from langchain\_community.embeddings import HuggingFaceEmbeddings

# from langchain.vectorstores.chroma import Chroma

# from langchain\_core.documents.base import Document

# import PyPDF2

# from redis import Redis

# import logging

# from celery.result import AsyncResult

# from langchain.text\_splitter import RecursiveCharacterTextSplitter

# from langchain\_community.embeddings import HuggingFaceEmbeddings

# import docx

# import sqlite3

# from typing import List

# from pdf\_utils import check\_if\_scanned\_full\_doc, convert\_to\_doc\_intell\_pdf\_format, process

# logger = logging.getLogger(\_\_name\_\_)

# # Celery setup

# redis = Redis(host="20.41.249.147", port=6379, username="default", password="admin", db=0)

# app = Celery('celery\_chroma\_ingest', broker='redis://20.41.249.147:6379/0', backend="redis://20.41.249.147:6379/0")

# # Initialize Chroma DB client

# chroma\_client = chromadb.PersistentClient(path="chromadb")

# collection = chroma\_client.get\_or\_create\_collection(name="pdf\_database")

# def get\_text(file\_path):

#     if file\_path.endswith('.pdf'):

#         scanned\_flag, \_ = check\_if\_scanned\_full\_doc(file\_path)

#         if scanned\_flag:

#             file\_names = convert\_to\_doc\_intell\_pdf\_format(file\_path)

#             text = ""

#             for file\_name in file\_names:

#                 docs = process(file\_name)

#                 for doc in docs:

#                     text += doc['block'] + "\n"

#         else:

#             pdf\_file = open(file\_path, 'rb')

#             pdf\_reader = PyPDF2.PdfReader(pdf\_file)

#             text = ""

#             for page\_num in range(len(pdf\_reader.pages)):

#                 text += pdf\_reader.pages[page\_num].extract\_text()

#             pdf\_file.close()

#     elif file\_path.endswith('.txt'):

#         with open(file\_path, 'r', encoding='utf-8') as file:

#             text = file.read()

#     elif file\_path.endswith('.docx'):

#         doc = docx.Document(file\_path)

#         text = ""

#         for para in doc.paragraphs:

#             text += para.text + "\n"

#     else:

#         text = ""

#     return text

# def get\_text\_from\_database(file\_path):

#     conn = sqlite3.connect(file\_path)

#     cursor = conn.cursor()

#     cursor.execute("SELECT name FROM sqlite\_master WHERE type='table'") # fetching list of tables from sqlite3 database

#     tables = [row[0] for row in cursor.fetchall()]

#     text = ""

#     for table in tables:

#         cursor.execute(f"PRAGMA table\_info({table})") # fetching list of column names, data types

#         column\_info = cursor.fetchall()

#         column\_names = [info[1] for info in column\_info]  # names of all columns

#         # Fetch data from all columns

#         rows = []

#         for row in cursor.execute(f"SELECT \* FROM {table}"):

#             row\_text = " ".join(str(value) for value in row)

#             rows.append(row\_text)

#         text += "\n".join(rows) + "\n"

#     conn.close()

#     return text

# # Initialize text splitter and embeddings

# text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=100)

# embeddings = HuggingFaceEmbeddings(model\_name="BAAI/bge-large-en-v1.5")

# # # embeddings = HuggingFaceEmbeddings(model\_name="intfloat/e5-base-v2")

# @app.task

# def process\_files():

#     uploads\_dir = 'uploads'

#     files\_processed = set()

#     for filename in os.listdir(uploads\_dir):

#         if filename.endswith('.pdf') or filename.endswith(".txt") or filename.endswith(".docx") or filename.endswith('.db'):

#             file\_path = os.path.join(uploads\_dir, filename)

#             if file\_path not in files\_processed:

#                 if filename.endswith('.db'):

#                     # Get text from the database

#                     text = get\_text\_from\_database(file\_path)

#                 else:

#                     text = get\_text(file\_path)

#                 chunks = text\_splitter.split\_text(text)

#                 # Convert chunks to vector representations and store in Chroma DB

#                 documents\_list = []

#                 embeddings\_list = []

#                 ids\_list = []

#                 for i, chunk in enumerate(chunks):

#                     vector = embeddings.embed\_query(chunk)

#                     documents\_list.append(chunk)

#                     embeddings\_list.append(vector)

#                     ids\_list.append(f"{filename}\_{i}")

#                 try:

#                     collection.add(embeddings=embeddings\_list, documents=documents\_list, ids=ids\_list)

#                 except chromadb.errors.ChromaDBError as e:

#                     logger.warning(f"Error adding embeddings for {filename}: {e}")

#                 logger.info(f"Stored {filename} in vector database")

#                 files\_processed.add(file\_path)

#     print("------")

#     print(collection)

#     task\_id = process\_files.request.id

#     return task\_id

# # Register the task with the Celery app

# app.tasks.register(process\_files)

# if \_\_name\_\_ == "\_\_main\_\_":

#     app.start()

import os

import datetime

from celery import Celery

from celery.schedules import crontab

import chromadb

from chromadb.config import Settings

from langchain\_community.embeddings import HuggingFaceEmbeddings

from langchain.vectorstores.chroma import Chroma

from langchain\_core.documents.base import Document

import PyPDF2

from redis import Redis

import logging

from celery.result import AsyncResult

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain\_community.embeddings import HuggingFaceEmbeddings

import docx

import sqlite3

from typing import List

logger = logging.getLogger(\_\_name\_\_)

# Celery setup

redis = Redis(host="20.41.249.147", port=6379, username="default", password="admin", db=0)

app = Celery('celery\_chroma\_ingest', broker='redis://20.41.249.147:6379/0', backend="redis://20.41.249.147:6379/0")

# Initialize Chroma DB client

# chroma\_client = chromadb.PersistentClient(path="chromadb")

chroma\_client = chromadb.HttpClient(host="20.41.249.147", port=6062)

collection = chroma\_client.get\_or\_create\_collection(name="pdf\_database")

def get\_text(file\_path):

    if file\_path.endswith('.pdf'):

        pdf\_file = open(file\_path, 'rb')

        pdf\_reader = PyPDF2.PdfReader(pdf\_file)

        text = ""

        for page\_num in range(len(pdf\_reader.pages)):

            text += pdf\_reader.pages[page\_num].extract\_text()

        pdf\_file.close()

        text = text

    elif file\_path.endswith('.txt'):

        with open(file\_path, 'r', encoding='utf-8') as file:

            text = file.read()

    elif file\_path.endswith('.docx'):

        doc = docx.Document(file\_path)

        text = ""

        for para in doc.paragraphs:

            text += para.text + "\n"

    else:

        text = ""

    return text

def get\_text\_from\_database(file\_path):

    conn = sqlite3.connect(file\_path)

    cursor = conn.cursor()

    cursor.execute("SELECT name FROM sqlite\_master WHERE type='table'") # fetching list of tables from sqlite3 database

    tables = [row[0] for row in cursor.fetchall()]

    text = ""

    for table in tables:

        cursor.execute(f"PRAGMA table\_info({table})") # fetching list of column names, data types

        # Fetch data from all columns

        rows = []

        for row in cursor.execute(f"SELECT \* FROM {table}"):

            row\_text = " ".join(str(value) for value in row)

            rows.append(row\_text)

        text += "\n".join(rows) + "\n"

    conn.close()

    return text

# Initialize text splitter and embeddings

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=100)

embeddings = HuggingFaceEmbeddings(model\_name="BAAI/bge-large-en-v1.5")

# embeddings = HuggingFaceEmbeddings(model\_name="intfloat/e5-base-v2")

@app.task

def process\_files():

    uploads\_dir = 'uploads'

    files\_processed = set()

    for filename in os.listdir(uploads\_dir):

        if filename.endswith('.pdf') or filename.endswith(".txt") or filename.endswith(".docx") or filename.endswith('.db'):

            file\_path = os.path.join(uploads\_dir, filename)

            if file\_path not in files\_processed:

                if filename.endswith('.db'):

                    # Get text from the database

                    text = get\_text\_from\_database(file\_path)

                else:

                    text = get\_text(file\_path)

                chunks = text\_splitter.split\_text(text)

                # Convert chunks to vector representations and store in Chroma DB

                documents\_list = []

                embeddings\_list = []

                ids\_list = []

                for i, chunk in enumerate(chunks):

                    vector = embeddings.embed\_query(chunk)

                    documents\_list.append(chunk)

                    embeddings\_list.append(vector)

                    ids\_list.append(f"{filename}\_{i}")

                try:

                    collection.add(embeddings=embeddings\_list, documents=documents\_list, ids=ids\_list)

                except chromadb.errors.ChromaDBError as e:

                    logger.warning(f"Error adding embeddings for {filename}: {e}")

                logger.info(f"Stored {filename} in vector database")

                files\_processed.add(file\_path)

    print("------")

    print(collection)

    task\_id = process\_files.request.id

    return task\_id

# Register the task with the Celery app

app.tasks.register(process\_files)

if \_\_name\_\_ == "\_\_main\_\_":

    app.start()