from django.shortcuts import render, redirect

from info.models import Info

from job.models import Job

from job.utils import match\_jobs\_to\_candidate

import openai

import logging

logging.basicConfig(level=logging.INFO)

def dashboard(request):

# Récupérer uniquement les jobs disponibles

jobs = Job.objects.filter(is\_available=True).exclude(pk=None)

return render(request, 'dashboard/dashboard.html', {'jobs': jobs})

def recruiter\_cvs(request):

if request.user.is\_recruiter:

# Récupérer tous les CVs ayant un fichier associé

cvs = Info.objects.exclude(upload\_cv="") # Exclut les CVs sans fichier

# Filtrer par recherche si nécessaire

search\_query = request.GET.get('search', '')

if search\_query:

cvs = cvs.filter(skills\_\_icontains=search\_query)

# Classer les CVs par pertinence

ranked\_cvs = sorted(cvs, key=lambda cv: cv.skills or "", reverse=True)

return render(request, 'dashboard/recruiter\_cvs.html', {'cvs': ranked\_cvs})

else:

return redirect('login')

def recommended\_jobs(request):

if request.user.is\_applicant:

info = Info.objects.get(user=request.user)

recommended\_jobs = []

for job in Job.objects.filter(is\_available=True):

match\_score = match\_jobs\_to\_candidate(info.skills, job.skills\_required)

if match\_score > 0.5: # Seuil de pertinence

recommended\_jobs.append((job, match\_score))

# Classer les offres par pertinence

recommended\_jobs = sorted(recommended\_jobs, key=lambda x: x[1], reverse=True)

return render(request, 'dashboard/recommended\_jobs.html', {'recommended\_jobs': recommended\_jobs})

else:

return redirect('login')

def generate\_cover\_letter(request, pk):

try:

# Récupérer le job correspondant

job = Job.objects.get(pk=pk)

except Job.DoesNotExist:

# Si le job n'existe pas, retourner une erreur 404

return render(request, 'dashboard/cover\_letter.html', {

'error': 'The requested job does not exist.'

})

# Vérifier si l'utilisateur est authentifié

if not request.user.is\_authenticated:

return render(request, 'dashboard/cover\_letter.html', {

'error': 'You need to log in to generate a cover letter.'

})

# Récupérer les informations du candidat

try:

info = Info.objects.get(user=request.user)

except Info.DoesNotExist:

return render(request, 'dashboard/cover\_letter.html', {

'error': 'You need to upload your profile information before generating a cover letter.'

})

# Vérifier si les informations nécessaires existent

if not job.title or not job.skills\_required or not info.skills:

return render(request, 'dashboard/cover\_letter.html', {

'error': 'Insufficient data to generate a cover letter. Please ensure job and profile information are complete.'

})

# Générer le prompt pour OpenAI

prompt = f"""

Generate a professional cover letter for the following job:

Job Title: {job.title}

Skills Required: {job.skills\_required}

Candidate Skills: {info.skills}

"""

# Appeler l'API OpenAI avec `openai.Chat.create`

try:

response = openai.Chat.create(

model="gpt-3.5-turbo",

messages=[

{"role": "user", "content": prompt}

]

)

letter = response["choices"][0]["message"]["content"]

except Exception as e: # Capturer toutes les erreurs

return render(request, 'dashboard/cover\_letter.html', {

'error': f"An error occurred while generating the cover letter: {str(e)}"

})

# Vérifier si une lettre a été générée

if not letter.strip():

return render(request, 'dashboard/cover\_letter.html', {

'error': 'The cover letter could not be generated. Please try again later.'

})

# Retourner le template avec la lettre générée

context = {'job': job, 'letter': letter}

return render(request, 'dashboard/cover\_letter.html', context)

def letters\_table(request):

# Récupérer les jobs disponibles

jobs = Job.objects.filter(is\_available=True)

return render(request, 'dashboard/letters\_table.html', {'jobs': jobs})