from flask import Flask, render\_template\_string, request, redirect, url\_for, session, flash, jsonify

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from datetime import datetime, timedelta

import praw

import json

from functools import wraps

import requests

from apscheduler.schedulers.background import BackgroundScheduler

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = os.environ.get('SECRET\_KEY', 'your-secret-key-change-this-' + str(os.urandom(24)))

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///reddit\_scraper.db'

db = SQLAlchemy(app)

# GoHighLevel Configuration

GHL\_API\_KEY = os.environ.get('GHL\_API\_KEY', 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2NhdGlvbl9pZCI6IlA0RUNGMGpEcGw0Wkl4UFZpZk9iIiwiY29tcGFueV9pZCI6Ik5WZzRLM2FDaFN6bmRkcjFHT1dKIiwidmVyc2lvbiI6MSwiaWF0IjoxNzA2NTgyNzI4NjY0LCJzdWIiOiJ1c2VyX2lkIn0.rYgfEe8ZJEwJvbIpX7BDqIgOG1kYYl\_BucRFmbPNsZU')

GHL\_LOCATION\_ID = os.environ.get('GHL\_LOCATION\_ID', 'P4ECF0jDpl4ZIxPVifOb')

# Database Models

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

email = db.Column(db.String(120), unique=True, nullable=False)

password\_hash = db.Column(db.String(200), nullable=False)

is\_admin = db.Column(db.Boolean, default=False)

email\_notifications = db.Column(db.Boolean, default=True)

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

scrapes = db.relationship('Scrape', backref='user', lazy=True)

class Scrape(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(100), nullable=False)

subreddits = db.Column(db.Text, nullable=False)

keywords = db.Column(db.Text, nullable=False)

limit = db.Column(db.Integer, default=50)

auto\_run = db.Column(db.Boolean, default=False)

run\_frequency = db.Column(db.String(20), default='daily') # daily, weekly, manual

next\_run = db.Column(db.DateTime)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

last\_run = db.Column(db.DateTime)

class ScrapeResult(db.Model):

id = db.Column(db.Integer, primary\_key=True)

scrape\_id = db.Column(db.Integer, db.ForeignKey('scrape.id'), nullable=False)

results = db.Column(db.Text)

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

emailed = db.Column(db.Boolean, default=False)

# Email Functions

def send\_email\_ghl(to\_email, subject, html\_content):

"""Send email via GoHighLevel"""

url = f"https://rest.gohighlevel.com/v1/conversations/messages/email"

headers = {

"Authorization": f"Bearer {GHL\_API\_KEY}",

"Content-Type": "application/json"

}

payload = {

"locationId": GHL\_LOCATION\_ID,

"type": "Email",

"email": to\_email,

"subject": subject,

"html": html\_content

}

try:

response = requests.post(url, headers=headers, json=payload)

return response.status\_code == 200

except Exception as e:

print(f"Email error: {e}")

return False

def send\_email\_smtp(to\_email, subject, html\_content):

"""Alternative: Send email via SMTP (if not using GHL)"""

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

smtp\_server = "smtp.gmail.com"

smtp\_port = 587

smtp\_user = "YOUR\_EMAIL@gmail.com"

smtp\_password = "YOUR\_APP\_PASSWORD"

msg = MIMEMultipart('alternative')

msg['Subject'] = subject

msg['From'] = smtp\_user

msg['To'] = to\_email

msg.attach(MIMEText(html\_content, 'html'))

try:

server = smtplib.SMTP(smtp\_server, smtp\_port)

server.starttls()

server.login(smtp\_user, smtp\_password)

server.sendmail(smtp\_user, to\_email, msg.as\_string())

server.quit()

return True

except Exception as e:

print(f"Email error: {e}")

return False

def generate\_email\_html(scrape\_name, results):

"""Generate beautiful HTML email with results"""

results\_html = ""

for i, result in enumerate(results[:20], 1): # Limit to top 20 results

keywords = ', '.join(result['keywords\_found'])

results\_html += f"""

<div style="background: #f9f9f9; padding: 20px; margin: 15px 0; border-radius: 8px; border-left: 4px solid #0079d3;">

<h3 style="margin: 0 0 10px 0; color: #0079d3;">

{i}. [{result['subreddit']}] {result['title']}

</h3>

<p style="color: #666; margin: 5px 0;">

<strong>Author:</strong> {result['author']} |

<strong>Score:</strong> {result['score']} |

<strong>Posted:</strong> {result['created']}

</p>

<p style="margin: 10px 0;">

<strong>Keywords:</strong> <span style="background: #ffeb3b; padding: 2px 8px; border-radius: 4px;">{keywords}</span>

</p>

<a href="{result['url']}" style="color: #0079d3; text-decoration: none; font-weight: bold;">

View on Reddit →

</a>

</div>

"""

html = f"""

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

</head>

<body style="font-family: Arial, sans-serif; max-width: 800px; margin: 0 auto; padding: 20px; background: #f5f5f5;">

<div style="background: white; padding: 40px; border-radius: 8px;">

<div style="text-align: center; margin-bottom: 30px;">

<h1 style="color: #1a1a1a; margin: 0;">🔍 Your Reddit Scrape Results</h1>

<p style="color: #666; font-size: 16px;">{scrape\_name}</p>

</div>

<div style="background: #e3f2fd; padding: 15px; border-radius: 6px; margin-bottom: 30px;">

<p style="margin: 0; color: #1976d2;">

<strong>Found {len(results)} matching posts!</strong> Here are the highlights:

</p>

</div>

{results\_html}

<div style="text-align: center; margin-top: 40px; padding-top: 20px; border-top: 2px solid #f0f0f0;">

<p style="color: #666;">

This is your automated daily scrape report.<br>

<a href="YOUR\_PLATFORM\_URL/dashboard" style="color: #0079d3;">View all results in dashboard</a>

</p>

</div>

</div>

</body>

</html>

"""

return html

# Reddit API Setup

import os

def get\_reddit\_instance():

return praw.Reddit(

client\_id=os.environ.get('REDDIT\_CLIENT\_ID', '**5MY2WXif65oZv7TTyiH4iQ**'),

client\_secret=os.environ.get('REDDIT\_CLIENT\_SECRET', 'X1H9Qj2rwZYKaiTU6Ck35ZClkpvyaA'),

user\_agent="howl\_klouder"

)

)

def run\_scrape\_job(scrape\_id):

"""Background job to run scrape and email results"""

with app.app\_context():

scrape = Scrape.query.get(scrape\_id)

if not scrape:

return

user = User.query.get(scrape.user\_id)

reddit = get\_reddit\_instance()

subreddits = json.loads(scrape.subreddits)

keywords = json.loads(scrape.keywords)

results = []

for subreddit\_name in subreddits:

try:

subreddit = reddit.subreddit(subreddit\_name)

for post in subreddit.new(limit=scrape.limit):

post\_text = f"{post.title} {post.selftext}".lower()

found\_keywords = [kw for kw in keywords if kw.lower() in post\_text]

if found\_keywords:

results.append({

'subreddit': subreddit\_name,

'title': post.title,

'author': str(post.author),

'score': post.score,

'url': f"https://reddit.com{post.permalink}",

'created': str(datetime.fromtimestamp(post.created\_utc)),

'keywords\_found': found\_keywords

})

except Exception as e:

print(f'Error scraping r/{subreddit\_name}: {str(e)}')

# Save results

result\_record = ScrapeResult(

scrape\_id=scrape.id,

results=json.dumps(results),

emailed=False

)

scrape.last\_run = datetime.utcnow()

# Schedule next run

if scrape.run\_frequency == 'daily':

scrape.next\_run = datetime.utcnow() + timedelta(days=1)

elif scrape.run\_frequency == 'weekly':

scrape.next\_run = datetime.utcnow() + timedelta(weeks=1)

db.session.add(result\_record)

db.session.commit()

# Send email if user wants notifications

if user.email\_notifications and len(results) > 0:

html\_content = generate\_email\_html(scrape.name, results)

subject = f"🔍 {len(results)} new Reddit posts found: {scrape.name}"

# Use GoHighLevel or SMTP

email\_sent = send\_email\_ghl(user.email, subject, html\_content)

# email\_sent = send\_email\_smtp(user.email, subject, html\_content) # Alternative

if email\_sent:

result\_record.emailed = True

db.session.commit()

def check\_scheduled\_scrapes():

"""Check and run scheduled scrapes"""

with app.app\_context():

now = datetime.utcnow()

due\_scrapes = Scrape.query.filter(

Scrape.auto\_run == True,

Scrape.next\_run <= now

).all()

for scrape in due\_scrapes:

run\_scrape\_job(scrape.id)

# Initialize scheduler

scheduler = BackgroundScheduler()

scheduler.add\_job(func=check\_scheduled\_scrapes, trigger="interval", minutes=10)

scheduler.start()

# Decorators

def login\_required(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if 'user\_id' not in session:

return redirect(url\_for('login'))

return f(\*args, \*\*kwargs)

return decorated\_function

def admin\_required(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if 'user\_id' not in session:

return redirect(url\_for('login'))

user = User.query.get(session['user\_id'])

if not user or not user.is\_admin:

flash('Admin access required')

return redirect(url\_for('dashboard'))

return f(\*args, \*\*kwargs)

return decorated\_function

# Routes

@app.route('/')

def index():

if 'user\_id' in session:

return redirect(url\_for('dashboard'))

return render\_template\_string(INDEX\_TEMPLATE)

@app.route('/register', methods=['GET', 'POST'])

def register():

if request.method == 'POST':

username = request.form['username']

email = request.form['email']

password = request.form['password']

if User.query.filter\_by(username=username).first():

flash('Username already exists')

return redirect(url\_for('register'))

if User.query.filter\_by(email=email).first():

flash('Email already exists')

return redirect(url\_for('register'))

user = User(

username=username,

email=email,

password\_hash=generate\_password\_hash(password),

email\_notifications=True

)

db.session.add(user)

db.session.commit()

flash('Account created successfully!')

return redirect(url\_for('login'))

return render\_template\_string(REGISTER\_TEMPLATE)

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = User.query.filter\_by(username=username).first()

if user and check\_password\_hash(user.password\_hash, password):

session['user\_id'] = user.id

session['is\_admin'] = user.is\_admin

session['username'] = user.username

return redirect(url\_for('dashboard'))

flash('Invalid username or password')

return render\_template\_string(LOGIN\_TEMPLATE)

@app.route('/logout')

def logout():

session.clear()

return redirect(url\_for('index'))

@app.route('/dashboard')

@login\_required

def dashboard():

user = User.query.get(session['user\_id'])

scrapes = Scrape.query.filter\_by(user\_id=user.id).all()

return render\_template\_string(DASHBOARD\_TEMPLATE, user=user, scrapes=scrapes)

@app.route('/settings', methods=['GET', 'POST'])

@login\_required

def settings():

user = User.query.get(session['user\_id'])

if request.method == 'POST':

user.email\_notifications = 'email\_notifications' in request.form

db.session.commit()

flash('Settings updated!')

return redirect(url\_for('dashboard'))

return render\_template\_string(SETTINGS\_TEMPLATE, user=user)

@app.route('/scrape/create', methods=['GET', 'POST'])

@login\_required

def create\_scrape():

if request.method == 'POST':

name = request.form['name']

subreddits = request.form['subreddits'].split(',')

keywords = request.form['keywords'].split(',')

limit = int(request.form.get('limit', 50))

auto\_run = 'auto\_run' in request.form

run\_frequency = request.form.get('run\_frequency', 'daily')

scrape = Scrape(

name=name,

subreddits=json.dumps([s.strip() for s in subreddits]),

keywords=json.dumps([k.strip() for k in keywords]),

limit=limit,

auto\_run=auto\_run,

run\_frequency=run\_frequency,

next\_run=datetime.utcnow() + timedelta(days=1) if auto\_run else None,

user\_id=session['user\_id']

)

db.session.add(scrape)

db.session.commit()

flash('Scrape configuration created!')

return redirect(url\_for('dashboard'))

return render\_template\_string(CREATE\_SCRAPE\_TEMPLATE)

@app.route('/scrape/run/<int:scrape\_id>')

@login\_required

def run\_scrape(scrape\_id):

scrape = Scrape.query.get\_or\_404(scrape\_id)

if scrape.user\_id != session['user\_id'] and not session.get('is\_admin'):

flash('Unauthorized')

return redirect(url\_for('dashboard'))

# Run scrape in background and redirect

run\_scrape\_job(scrape\_id)

flash('Scrape completed! Check your email if notifications are enabled.')

# Get latest results

latest\_result = ScrapeResult.query.filter\_by(scrape\_id=scrape\_id).order\_by(ScrapeResult.created\_at.desc()).first()

results = json.loads(latest\_result.results) if latest\_result else []

return render\_template\_string(RESULTS\_TEMPLATE, results=results, scrape=scrape)

@app.route('/admin')

@admin\_required

def admin\_panel():

users = User.query.all()

all\_scrapes = Scrape.query.all()

return render\_template\_string(ADMIN\_TEMPLATE, users=users, all\_scrapes=all\_scrapes)

@app.route('/admin/impersonate/<int:user\_id>')

@admin\_required

def impersonate\_user(user\_id):

user = User.query.get\_or\_404(user\_id)

session['original\_admin\_id'] = session['user\_id']

session['impersonating'] = True

session['user\_id'] = user.id

session['username'] = user.username

session['is\_admin'] = False

flash(f'Now viewing as {user.username}')

return redirect(url\_for('dashboard'))

@app.route('/admin/stop-impersonate')

@login\_required

def stop\_impersonate():

if 'original\_admin\_id' in session:

admin = User.query.get(session['original\_admin\_id'])

session['user\_id'] = admin.id

session['username'] = admin.username

session['is\_admin'] = True

session.pop('original\_admin\_id')

session.pop('impersonating')

flash('Returned to admin account')

return redirect(url\_for('admin\_panel'))

# Templates (keeping previous ones and adding new ones)

SETTINGS\_TEMPLATE = '''

<!DOCTYPE html>

<html>

<head>

<title>Settings</title>

<style>

body { font-family: Arial, sans-serif; max-width: 600px; margin: 50px auto; padding: 20px; }

.setting-group { background: #f9f9f9; padding: 20px; margin: 15px 0; border-radius: 8px; }

.btn { background: #0079d3; color: white; padding: 12px 24px; border: none; cursor: pointer; border-radius: 4px; }

input[type="checkbox"] { margin-right: 10px; }

label { font-weight: bold; }

</style>

</head>

<body>

<h2>Account Settings</h2>

<form method="POST">

<div class="setting-group">

<label>

<input type="checkbox" name="email\_notifications" {% if user.email\_notifications %}checked{% endif %}>

Email me daily scrape results

</label>

<p style="color: #666; margin-top: 10px; font-size: 14px;">

Receive automated emails when your scheduled scrapes find new posts

</p>

</div>

<button type="submit" class="btn">Save Settings</button>

<a href="{{ url\_for('dashboard') }}" style="margin-left: 15px;">Cancel</a>

</form>

</body>

</html>

'''

# Update CREATE\_SCRAPE\_TEMPLATE to include automation options

CREATE\_SCRAPE\_TEMPLATE = '''

<!DOCTYPE html>

<html>

<head>

<title>Create Scrape</title>

<style>

body { font-family: Arial, sans-serif; max-width: 600px; margin: 50px auto; padding: 20px; }

input, textarea, select { width: 100%; padding: 10px; margin: 10px 0; box-sizing: border-box; }

.btn { background: #0079d3; color: white; padding: 12px; border: none; width: 100%; cursor: pointer; border-radius: 4px; }

label { font-weight: bold; display: block; margin-top: 15px; }

.hint { color: #666; font-size: 14px; }

.automation-box { background: #e3f2fd; padding: 15px; border-radius: 6px; margin: 15px 0; }

input[type="checkbox"] { width: auto; margin-right: 8px; }

</style>

</head>

<body>

<h2>Create New Scrape</h2>

<form method="POST">

<label>Scrape Name</label>

<input type="text" name="name" placeholder="My Tech News Scrape" required>

<label>Subreddits (comma-separated)</label>

<input type="text" name="subreddits" placeholder="python, programming, webdev" required>

<span class="hint">Enter subreddit names without r/</span>

<label>Keywords (comma-separated)</label>

<input type="text" name="keywords" placeholder="tutorial, beginner, guide" required>

<span class="hint">Posts containing any of these words will be found</span>

<label>Posts per Subreddit</label>

<input type="number" name="limit" value="50" min="10" max="500">

<div class="automation-box">

<label>

<input type="checkbox" name="auto\_run" id="auto\_run" onchange="toggleFrequency()">

Run automatically and email me results

</label>

<div id="frequency\_options" style="display: none; margin-top: 15px;">

<label>Frequency</label>

<select name="run\_frequency">

<option value="daily">Daily</option>

<option value="weekly">Weekly</option>

</select>

</div>

</div>

<button type="submit" class="btn">Create Scrape</button>

</form>

<p><a href="{{ url\_for('dashboard') }}">Back to Dashboard</a></p>

<script>

function toggleFrequency() {

const checkbox = document.getElementById('auto\_run');

const options = document.getElementById('frequency\_options');

options.style.display = checkbox.checked ? 'block' : 'none';

}

</script>

</body>

</html>

'''

# Keep all other templates from before (INDEX, REGISTER, LOGIN, DASHBOARD, RESULTS, ADMIN)

INDEX\_TEMPLATE = '''[Previous INDEX\_TEMPLATE code]'''

REGISTER\_TEMPLATE = '''[Previous REGISTER\_TEMPLATE code]'''

LOGIN\_TEMPLATE = '''[Previous LOGIN\_TEMPLATE code]'''

DASHBOARD\_TEMPLATE = '''[Previous DASHBOARD\_TEMPLATE code with added automation indicators]'''

RESULTS\_TEMPLATE = '''[Previous RESULTS\_TEMPLATE code]'''

ADMIN\_TEMPLATE = '''[Previous ADMIN\_TEMPLATE code]'''

if \_\_name\_\_ == '\_\_main\_\_':

with app.app\_context():

db.create\_all()

if not User.query.filter\_by(username='admin').first():

admin = User(

username='admin',

email='admin@example.com',

password\_hash=generate\_password\_hash('admin123'),

is\_admin=True

)

db.session.add(admin)

db.session.commit()

print("Created admin user: admin / admin123")

app.run(debug=True, port=5000)