import ibm\_db as db

from flask import Flask, render\_template, request, redirect, session, abort

import os

import pathlib

import requests

from dotenv import load\_dotenv

from sendgrid import SendGridAPIClient

from sendgrid.helpers.mail import Mail

from google.oauth2 import id\_token

from google\_auth\_oauthlib.flow import Flow

from pip.\_vendor import cachecontrol

import google.auth.transport.requests

# Configure Flask app

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

SECRET\_KEY = os.urandom(32)

app.config['SECRET\_KEY'] = SECRET\_KEY

# Load .env file

load\_dotenv()

# Connect to the Database

HOSTNAME = os.getenv('HOSTNAME')

PORT\_NUMBER = os.getenv('PORT\_NUMBER')

DATABASE\_NAME = os.getenv('DATABASE\_NAME')

USERNAME = os.getenv('USER')

PASSWORD = os.getenv('PASSWORD')

GOOGLE\_CLIENT\_ID = os.getenv('GOOGLE\_AUTH\_CLIENT\_ID')

connection\_string = "DATABASE={0};HOSTNAME={1};PORT={2};SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;PROTOCOL=TCPIP;UID={3};PWD={4};".format(DATABASE\_NAME, HOSTNAME, PORT\_NUMBER, USERNAME, PASSWORD)

conn = db.connect(connection\_string, "", "")

# Frequently used variables

SIGN\_UP\_PAGE\_URL = '/'

LOG\_IN\_PAGE\_URL = '/login'

HOME\_PAGE\_URL = '/home'

GOOGLE\_LOGIN\_PAGE\_URL = '/google\_login'

# Google Auth Configuration

os.environ["OAUTHLIB\_INSECURE\_TRANSPORT"] = "1"

client\_secrets\_file = os.path.join(pathlib.Path(\_\_file\_\_).parent, "client\_secret.json")

flow = Flow.from\_client\_secrets\_file(

client\_secrets\_file=client\_secrets\_file,

scopes=["https://www.googleapis.com/auth/userinfo.profile", "https://www.googleapis.com/auth/userinfo.email", "openid"],

redirect\_uri="http://127.0.0.1:5000/callback"

)

# Helper Function to execute SQL queries

def execute\_sql(statement, \*\*params):

global conn

stmt = db.prepare(conn, statement)

param\_id = 1

for key, val in params.items():

db.bind\_param(stmt, param\_id, val)

param\_id += 1

result = ''

try:

db.execute(stmt)

result = db.fetch\_assoc(stmt)

except:

pass

return result

# Creates user table if not exists

create\_table = "CREATE TABLE IF NOT EXISTS user(email varchar(30), username varchar(30), password varchar(30))"

execute\_sql(statement=create\_table)

# Helper function to send confirmation mail on sign in

def send\_confirmation\_mail(user, email):

message = Mail(

from\_email="nutritionassistant854@gmail.com",

to\_emails=email,

subject="YAYY!! Your Account was created successfully!",

html\_content= "<strong>Account Created with username {0}</strong>".format(user)

)

try:

sg = SendGridAPIClient(os.environ.get('SENDGRID\_API\_KEY'))

response = sg.send(message)

print(response.status\_code)

print(response.body)

print(response.headers)

except Exception as e:

print(e)

# Sign up page

@app.route(SIGN\_UP\_PAGE\_URL, methods=['GET', 'POST'])

def signup():

msg = ''

if session.get('user'):

return redirect(HOME\_PAGE\_URL)

if request.method == 'POST':

user = request.form['user']

email = request.form['email']

password = request.form['password']

duplicate\_check = "SELECT \* FROM user WHERE username=?"

account = execute\_sql(statement=duplicate\_check, user=user)

if account:

msg = "There is already an account with this username!"

else:

insert\_query = "INSERT INTO user values(?, ?, ?)"

execute\_sql(statement=insert\_query, email=email, user=user, password=password)

send\_confirmation\_mail(user, email)

return redirect(LOG\_IN\_PAGE\_URL)

return render\_template('signup.html', msg=msg)

# Login page

@app.route(LOG\_IN\_PAGE\_URL, methods=['GET', 'POST'])

def login():

msg = ''

if session.get('user'):

return redirect(HOME\_PAGE\_URL)

if request.method == "POST":

user = request.form['user']

password = request.form['password']

duplicate\_check = "SELECT \* FROM user WHERE username=?"

account = execute\_sql(statement=duplicate\_check, user=user)

print(account)

if account and account['PASSWORD'] == password:

session['user'] = user

return redirect(HOME\_PAGE\_URL)

elif account and account['PASSWORD'] != password:

msg = 'Invalid Password!'

else:

msg = "Invalid Username!"

return render\_template('login.html', msg=msg)

# Login using Gmail

@app.route(GOOGLE\_LOGIN\_PAGE\_URL)

def google\_login():

authorization\_url, state = flow.authorization\_url()

session["state"] = state

return redirect(authorization\_url)

# Configuring user credentials after gmail login

@app.route("/callback")

def callback():

flow.fetch\_token(authorization\_response=request.url)

if session["state"] != request.args["state"]:

abort(500) # State does not match!

credentials = flow.credentials

request\_session = requests.session()

cached\_session = cachecontrol.CacheControl(request\_session)

token\_request = google.auth.transport.requests.Request(session=cached\_session)

id\_info = id\_token.verify\_oauth2\_token(

id\_token=credentials.\_id\_token,

request=token\_request,

audience=GOOGLE\_CLIENT\_ID,

clock\_skew\_in\_seconds=10

)

session["user"] = id\_info.get("email")

session["google\_id"] = id\_info.get("sub")

session["name"] = id\_info.get("name")

return redirect(HOME\_PAGE\_URL)

# Home page

@app.route(HOME\_PAGE\_URL)

def homepage():

if not session.get('user'):

return redirect(LOG\_IN\_PAGE\_URL)

return render\_template('homepage.html', user=session.get('user'))

# Logout user

@app.route('/logout')

def logout():

session['user'] = ''

return redirect(LOG\_IN\_PAGE\_URL)

# Delete user account

@app.route('/delete')

def delete():

if not session.get('user'):

return redirect(LOG\_IN\_PAGE\_URL)

user = session['user']

delete\_query = "DELETE FROM user WHERE username=?"

execute\_sql(statement=delete\_query, user=user)

session.clear()

return redirect(SIGN\_UP\_PAGE\_URL)

# Run the application

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

app.run(debug=True)