from cryptography.fernet import Fernet

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.backends import default\_backend

from cryptography.hazmat.primitives import padding

from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

import os

import base64

from flask import Flask, request, jsonify, session

import mysql.connector

import bcrypt

from flask\_cors import CORS

import smtplib

import random # Correct import for random module

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

app.secret\_key = 'your\_secret\_key\_here' # Necessary for session management

CORS(app, resources={r"/\*": {"origins": "\*"}}, supports\_credentials=True) # Allow all origins (or specify your frontend domain)

# Static keys for AES-CBC and Fernet

AES\_CBC\_KEY = b'your\_32\_byte\_aes\_key\_here' # 32 bytes for AES-CBC (e.g., 256-bit)

FERNET\_KEY = b'your\_fernet\_key\_here' # 32 bytes for Fernet (Fernet keys are base64-encoded)

SALT\_LENGTH = 16

# MySQL connection details

db\_config = {

'user': 'dlsudercproject', # Your MySQL username

'password': 'CompScieBCS42', # Your MySQL password

'host': 'dlsudercproject.mysql.pythonanywhere-services.com', # Your database host address

'database': 'dlsudercproject$decoy' # Your database name

}

# Function to connect to MySQL

def get\_db\_connection():

connection = mysql.connector.connect(\*\*db\_config)

return connection

# Function to get user by email

def get\_user\_by\_email(email):

connection = get\_db\_connection()

cursor = connection.cursor(dictionary=True) # Ensure results are returned as dictionaries

query = "SELECT \* FROM users WHERE email = %s"

try:

cursor.execute(query, (email,))

user = cursor.fetchone() # Fetch one result

if user:

print(f"User found: {user}") # Log user data to debug

else:

print(f"User not found for email: {email}") # Log if user is not found

return user

except mysql.connector.Error as err:

print(f"Error: {err}")

return None

finally:

cursor.close()

connection.close()

def create\_protocol(email):

"""Function to create a new protocol with just the user's email."""

conn = mysql.connector.connect(\*\*db\_config)

cursor = conn.cursor()

query = "INSERT INTO Protocol (email) VALUES (%s)"

cursor.execute(query, (email,))

conn.commit()

cursor.close()

conn.close()

return "Protocol created successfully"

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

def signup():

# Extract data from the request

name = request.form.get('name')

email = request.form.get('email')

password = request.form.get('password')

account\_type = request.form.get('account-type')

# Check if any required field is missing

if not name or not email or not password or not account\_type:

return jsonify({"error": "Missing required fields."}), 400

# Hash the password

hashed\_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

# Insert into the database

connection = get\_db\_connection()

cursor = connection.cursor()

query = """

INSERT INTO users (name, email, password, account\_type)

VALUES (%s, %s, %s, %s)

"""

try:

cursor.execute(query, (name, email, hashed\_password, account\_type))

connection.commit()

return jsonify({"message": "Sign up successful!"}), 200

except mysql.connector.Error as err:

return jsonify({"error": f"Database Error: {err}"}), 500

finally:

cursor.close()

connection.close()

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

def login():

email = request.form.get('email')

password = request.form.get('password')

# Retrieve the user by email

user = get\_user\_by\_email(email)

if user:

hashed\_password = user['password']

# Check if the password matches

if bcrypt.checkpw(password.encode('utf-8'), hashed\_password.encode('utf-8')):

# Check if email is confirmed

if not user['confirmation\_status']:

return jsonify({"message": "Please confirm your email.", "redirect": "confirmation.html"}), 403

else:

# Store user info in session

session['user\_email'] = email

session['account\_type'] = user['account\_type'] # Store account-type in session

# Return the email and account-type to the frontend

return jsonify({

"message": "Login successful!",

"redirect": "dashboard.html",

"userEmail": email,

"userName": user['name'],

"accountType": user['account\_type']

}), 200

else:

return jsonify({"error": "Invalid email or password"}), 400

else:

return jsonify({"error": "User not found"}), 404

def send\_email(recipient\_email, confirmation\_code):

try:

sender\_email = "loyscedlsudercproject@gmail.com"

sender\_password = "wanq hmib dsjn nbbv" # This is your app-specific password

subject = "Your Confirmation Code"

body = f"Your confirmation code is: {confirmation\_code}"

message = f"Subject: {subject}\n\n{body}"

# Send the email through Gmail's SMTP server

with smtplib.SMTP\_SSL("smtp.gmail.com", 465) as server:

server.login(sender\_email, sender\_password)

server.sendmail(sender\_email, recipient\_email, message)

return True

except Exception as e:

print(f"Error sending email: {e}")

return False

# API endpoint to handle sending the confirmation code

@app.route('/send-code', methods=['POST'])

def send\_code():

data = request.get\_json()

email = data.get('email')

if not email:

return jsonify({"error": "Email is required"}), 400

# Generate a random 6-digit confirmation code

confirmation\_code = random.randint(100000, 999999)

# Store the confirmation code in session

session['confirmation\_code'] = confirmation\_code

# Send the email

if send\_email(email, confirmation\_code):

return jsonify({"message": "Code sent successfully", "confirmationCode": confirmation\_code})

else:

return jsonify({"error": "Failed to send email. Please try again later."}), 500

@app.route('/verify-code', methods=['POST'])

def verify\_code():

data = request.get\_json()

email = data.get('email')

if not email:

return jsonify({"error": "Email is required."}), 400

# Update confirmation status in the database based on the email

connection = get\_db\_connection()

cursor = connection.cursor()

query = "UPDATE users SET confirmation\_status = TRUE WHERE email = %s"

try:

cursor.execute(query, (email,))

connection.commit()

return jsonify({"message": "Email confirmed! Redirecting to Log in page."}), 200

except mysql.connector.Error as err:

return jsonify({"error": f"Error: {err}"}), 500

finally:

cursor.close()

connection.close()

@app.route('/get-user-role', methods=['GET'])

def get\_user\_role():

email = request.args.get('email') # Get email from the query parameters

if not email:

return jsonify({"error": "Email is required"}), 400

try:

connection = get\_db\_connection()

cursor = connection.cursor(dictionary=True)

# Query to fetch the role based on email

query = "SELECT role FROM users WHERE email = %s"

cursor.execute(query, (email,))

# Fetch the result

user = cursor.fetchone()

if user:

return jsonify({"role": user['role']}), 200 # Return the user's role

else:

return jsonify({"error": "User not found"}), 404

except Error as e:

return jsonify({"error": str(e)}), 500

finally:

if connection:

connection.close()

@app.route('/change\_password', methods=['POST'])

def change\_password():

data = request.get\_json()

email = data.get('email')

current\_password = data.get('currentPassword')

new\_password = data.get('newPassword')

if not email or not current\_password or not new\_password:

return jsonify({"error": "All fields are required."}), 400

# Establish a database connection

try:

conn = mysql.connector.connect(\*\*db\_config)

cursor = conn.cursor(dictionary=True)

# Fetch user by email

cursor.execute("SELECT \* FROM users WHERE email = %s", (email,))

user = cursor.fetchone()

if not user:

return jsonify({"error": "User not found."}), 404

# Verify the current password

if not bcrypt.checkpw(current\_password.encode('utf-8'), user['password'].encode('utf-8')):

return jsonify({"error": "Current password is incorrect."}), 400

# Hash the new password

hashed\_new\_password = bcrypt.hashpw(new\_password.encode('utf-8'), bcrypt.gensalt())

# Update the password in the database

cursor.execute("UPDATE users SET password = %s WHERE email = %s", (hashed\_new\_password, email))

conn.commit()

cursor.close()

conn.close()

return jsonify({"message": "Password updated successfully!"}), 200

except mysql.connector.Error as err:

return jsonify({"error": f"Database error: {err}"}), 500

@app.route('/submit-protocol', methods=['POST'])

def submit\_protocol():

try:

# Debug: Log form data

print(f"Form Data: {request.form}")

print(f"Files: {request.files}")

# Common fields

research\_title = request.form['research-title']

proponent1 = request.form['proponent1']

proponent2 = request.form.get('proponent2', None)

proponent3 = request.form.get('proponent3', None)

college = request.form['college']

acad\_year = request.form['acad-year']

review\_type = request.form['review-type']

email = request.form.get('user-email')

category = request.form.get('category')

experiment\_type = request.form.get('experiment-type', None)

comments = " " # Default value for comments (empty string)

# Debug: Log the values of common fields

print(f"Research Title: {research\_title}")

print(f"Proponent1: {proponent1}, Proponent2: {proponent2}, Proponent3: {proponent3}")

print(f"College: {college}, Acad Year: {acad\_year}")

print(f"Review Type: {review\_type}")

print(f"Email: {email}, Category: {category}")

# File handling

uploaded\_files = {}

for file\_key in request.files:

file = request.files[file\_key]

if file.filename:

# Extract file extension

file\_extension = f".{file.filename.rsplit('.', 1)[-1]}" if '.' in file.filename else ''

uploaded\_files[file\_key] = {

"data": file.read(),

"name": file.filename.rsplit('.', 1)[0], # File name without extension

"extension": file\_extension

}

# Debug: Log uploaded file details

print(f"Uploaded Files: {uploaded\_files}")

# Connect to the MySQL database

connection = mysql.connector.connect(\*\*db\_config)

cursor = connection.cursor()

# Insert the protocol data into the database

cursor.execute("""

INSERT INTO Protocol (

ResearchTitle, Email, College, ReviewType, Proponent1, Proponent2, Proponent3,

AcadYear, Category, Comments, ExperimentType

)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

""", (

research\_title, email, college, review\_type, proponent1, proponent2, proponent3,

acad\_year, category, comments, experiment\_type

))

# Get the last inserted protocol ID

protocol\_id = cursor.lastrowid

# Insert file data (LONGBLOB) into the database

for file\_key, file\_info in uploaded\_files.items():

cursor.execute("""

INSERT INTO Files (Protoid, FileType, Filename, FileExtension, FileData)

VALUES (%s, %s, %s, %s, %s)

""", (

protocol\_id, file\_key, file\_info["name"], file\_info["extension"], file\_info["data"]

))

# Commit the transaction

connection.commit()

# Close the cursor and connection

cursor.close()

connection.close()

return jsonify({'message': 'Protocol submitted successfully.'}), 200

except Exception as e:

# In case of any error, log the error and return the error message

print(f"Error: {str(e)}")

return jsonify({'error': str(e)}), 400

@app.route('/get-protocols', methods=['GET'])

def get\_protocols():

try:

# Get the email from the query parameters

user\_email = request.args.get('email')

if not user\_email:

return jsonify({'status': 'error', 'message': 'Email parameter is missing'}), 400

# Connect to the MySQL database

connection = mysql.connector.connect(\*\*db\_config)

cursor = connection.cursor()

# Query to get protocols for the provided email

query = """

SELECT Protoid, ResearchTitle, EthicsStatus

FROM Protocol

WHERE Email = %s

"""

cursor.execute(query, (user\_email,))

# Fetch all protocols for the user

protocols = cursor.fetchall()

# Format the response

protocols\_list = []

for protocol in protocols:

protocols\_list.append({

'Protoid': protocol[0],

'ResearchTitle': protocol[1],

'EthicsStatus': protocol[2] if protocol[2] else 'No Ethics Status Yet',

'ViewProtocolLink': f'/view-protocol/{protocol[0]}'

})

# Return the formatted protocols

return jsonify({'status': 'success', 'protocols': protocols\_list})

except mysql.connector.Error as e:

# Log and return database-specific errors

print(f"Database Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'Database error occurred', 'details': str(e)}), 500

except Exception as e:

# Log and return general errors

print(f"Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'An error occurred', 'details': str(e)}), 500

finally:

# Ensure the cursor and connection are closed

if 'cursor' in locals() and cursor is not None:

cursor.close()

if 'connection' in locals() and connection.is\_connected():

connection.close()

@app.route('/get-protocol-details', methods=['GET'])

def get\_protocol\_details():

try:

protoid = request.args.get('protoid')

if not protoid:

return jsonify({'status': 'error', 'message': 'Protoid is required'}), 400

# Connect to the MySQL database

connection = mysql.connector.connect(\*\*db\_config)

cursor = connection.cursor()

# Query to get protocol details (without files)

query\_protocol = """

SELECT Protoid, ResearchTitle, Email, College, Category, ReviewType,

Proponent1, Proponent2, Proponent3, Comments, AcadYear, EthicsStatus

FROM Protocol

WHERE Protoid = %s

"""

cursor.execute(query\_protocol, (protoid,))

protocol = cursor.fetchone()

if not protocol:

return jsonify({'status': 'error', 'message': 'Protocol not found'}), 404

# Format the response with protocol details (without files)

protocol\_data = {

'Protoid': protocol[0],

'ResearchTitle': protocol[1],

'Email': protocol[2],

'College': protocol[3],

'Category': protocol[4],

'ReviewType': protocol[5],

'Proponent1': protocol[6],

'Proponent2': protocol[7] if protocol[7] else '',

'Proponent3': protocol[8] if protocol[8] else '',

'Comments': protocol[9],

'AcadYear': protocol[10],

'EthicsStatus': protocol[11] if protocol[11] else 'Pending',

}

return jsonify({'status': 'success', 'protocol': protocol\_data})

except mysql.connector.Error as e:

# Log and return database-specific errors

print(f"Database Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'Database error occurred', 'details': str(e)}), 500

except Exception as e:

# Log and return general errors

print(f"Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'An error occurred', 'details': str(e)}), 500

finally:

# Ensure the cursor and connection are closed

if 'cursor' in locals() and cursor is not None:

cursor.close()

if 'connection' in locals() and connection.is\_connected():

connection.close()

@app.route('/get-protocol-files', methods=['GET'])

def get\_protocol\_files():

try:

protoid = request.args.get('protoid')

if not protoid:

return jsonify({'status': 'error', 'message': 'Protoid is required'}), 400

# Connect to the MySQL database

connection = mysql.connector.connect(\*\*db\_config)

cursor = connection.cursor()

# Query to fetch associated files

query\_files = """

SELECT FileID, FileType, Filename, FileExtension

FROM Files

WHERE Protoid = %s

"""

cursor.execute(query\_files, (protoid,))

files = cursor.fetchall()

if not files:

return jsonify({'status': 'success', 'files': []}) # No files found

# Format the file data

file\_data = [

{

'FileID': file[0],

'FileType': file[1],

'Filename': file[2],

'FileExtension': file[3],

}

for file in files

]

return jsonify({'status': 'success', 'files': file\_data})

except mysql.connector.Error as e:

print(f"Database Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'Database error occurred', 'details': str(e)}), 500

except Exception as e:

print(f"Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'An error occurred', 'details': str(e)}), 500

finally:

if 'cursor' in locals() and cursor is not None:

cursor.close()

if 'connection' in locals() and connection.is\_connected():

connection.close()

@app.route('/download-protocol-files', methods=['GET'])

def download\_protocol\_files():

try:

protoid = request.args.get('protoid')

if not protoid:

return jsonify({'status': 'error', 'message': 'Protoid is required'}), 400

# Connect to the MySQL database

connection = mysql.connector.connect(\*\*db\_config)

cursor = connection.cursor()

# Query to fetch associated files

query\_files = """

SELECT Filename, FileType, FileData

FROM Files

WHERE Protoid = %s

"""

cursor.execute(query\_files, (protoid,))

files = cursor.fetchall()

if not files:

return jsonify({'status': 'error', 'message': 'No files found for this protocol'}), 404

# For simplicity, handle one file at a time for downloading

if len(files) == 1:

file = files[0]

filename, file\_type, file\_data = file

# Stream the file as an attachment

return Response(

file\_data,

mimetype=file\_type,

headers={"Content-Disposition": f"attachment; filename={filename}"}

)

else:

return jsonify({'status': 'error', 'message': 'Multiple files detected. Provide additional logic to handle multiple files.'}), 500

except mysql.connector.Error as e:

print(f"Database Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'Database error occurred', 'details': str(e)}), 500

except Exception as e:

print(f"Error: {str(e)}")

return jsonify({'status': 'error', 'message': 'An error occurred', 'details': str(e)}), 500

finally:

if 'cursor' in locals() and cursor is not None:

cursor.close()

if 'connection' in locals() and connection.is\_connected():

connection.close()

# Run the Flask app

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

app.run(debug=True)