Task 4

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

from sqlalchemy.exc import IntegrityError

from flask\_bcrypt import Bcrypt

from flask\_jwt\_extended import JWTManager, create\_access\_token, jwt\_required, get\_jwt\_identity

import redis

import os

import uuid

import re

from dotenv import load\_dotenv

# Load environment variables from .env file

load\_dotenv()

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

# Database configuration

app.config['SQLALCHEMY\_DATABASE\_URI'] = os.getenv('DATABASE\_URI', 'sqlite:///users.db')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

app.config['SQLALCHEMY\_POOL\_SIZE'] = int(os.getenv('POOL\_SIZE', 5))

app.config['SQLALCHEMY\_MAX\_OVERFLOW'] = int(os.getenv('MAX\_OVERFLOW', 10))

app.config['JWT\_SECRET\_KEY'] = os.getenv('JWT\_SECRET\_KEY', 'your\_jwt\_secret\_key')

db = SQLAlchemy(app)

bcrypt = Bcrypt(app)

jwt = JWTManager(app)

# Redis cache configuration

redis\_host = os.getenv('REDIS\_HOST', 'localhost')

redis\_port = int(os.getenv('REDIS\_PORT', 6379))

redis\_db = int(os.getenv('REDIS\_DB', 0))

cache = redis.Redis(host=redis\_host, port=redis\_port, db=redis\_db)

# User model

class User(db.Model):

\_\_tablename\_\_ = 'users'

id = db.Column(db.String(36), primary\_key=True, default=lambda: str(uuid.uuid4()))

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

email = db.Column(db.String(100), unique=True, nullable=False)

password = db.Column(db.String(128), nullable=False)

age = db.Column(db.Integer, nullable=False)

role = db.Column(db.String(20), default='user')

# Utility function to validate email

def is\_valid\_email(email):

email\_regex = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'

return re.match(email\_regex, email) is not None

# Create the database tables

@app.before\_first\_request

def create\_tables():

db.create\_all()

# User registration

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

def register\_user():

data = request.get\_json()

if not data:

return jsonify({"error": "No data provided"}), 400

name = data.get("name")

email = data.get("email")

password = data.get("password")

age = data.get("age")

if not name or not email or not password or not age:

return jsonify({"error": "Missing required fields: name, email, password, or age"}), 400

if not is\_valid\_email(email):

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

try:

age = int(age)

if age <= 0:

raise ValueError

except ValueError:

return jsonify({"error": "Age must be a positive integer"}), 400

hashed\_password = bcrypt.generate\_password\_hash(password).decode('utf-8')

new\_user = User(name=name, email=email, password=hashed\_password, age=age)

try:

db.session.add(new\_user)

db.session.commit()

except IntegrityError:

db.session.rollback()

return jsonify({"error": "Email already exists"}), 400

return jsonify({"message": "User registered successfully"}), 201

# User login

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

def login\_user():

data = request.get\_json()

if not data:

return jsonify({"error": "No data provided"}), 400

email = data.get("email")

password = data.get("password")

if not email or not password:

return jsonify({"error": "Missing required fields: email or password"}), 400

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

if not user or not bcrypt.check\_password\_hash(user.password, password):

return jsonify({"error": "Invalid email or password"}), 401

access\_token = create\_access\_token(identity={"id": user.id, "role": user.role})

return jsonify({"access\_token": access\_token}), 200

# Get all users (Admin only)

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

@jwt\_required()

def get\_users():

current\_user = get\_jwt\_identity()

if current\_user['role'] != 'admin':

return jsonify({"error": "Access forbidden"}), 403

cached\_users = cache.get('users')

if cached\_users:

return jsonify({"users": eval(cached\_users)}), 200

users = User.query.all()

users\_data = [{"id": user.id, "name": user.name, "email": user.email, "age": user.age} for user in users]

# Cache the result with expiration of 300 seconds

cache.set('users', str(users\_data), ex=300)

return jsonify({"users": users\_data}), 200

# Get user profile (Authenticated users)

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

@jwt\_required()

def get\_profile():

current\_user = get\_jwt\_identity()

user = User.query.get(current\_user['id'])

if not user:

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

return jsonify({"id": user.id, "name": user.name, "email": user.email, "age": user.age, "role": user.role}), 200

# Update a user by ID (Admin or Owner only)

@app.route('/users/<user\_id>', methods=['PUT'])

@jwt\_required()

def update\_user(user\_id):

current\_user = get\_jwt\_identity()

if current\_user['role'] != 'admin' and current\_user['id'] != user\_id:

return jsonify({"error": "Access forbidden"}), 403

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

if not user:

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

data = request.get\_json()

if not data:

return jsonify({"error": "No data provided"}), 400

name = data.get("name", user.name)

email = data.get("email", user.email)

age = data.get("age", user.age)

if not is\_valid\_email(email):

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

try:

age = int(age)

if age <= 0:

raise ValueError

except ValueError:

return jsonify({"error": "Age must be a positive integer"}), 400

user.name = name

user.email = email

user.age = age

try:

db.session.commit()

except IntegrityError:

db.session.rollback()

return jsonify({"error": "Email already exists"}), 400

# Invalidate cache after update

cache.delete('users')

return jsonify({"id": user.id, "name": user.name, "email": user.email, "age": user.age}), 200

# Delete a user by ID (Admin only)

@app.route('/users/<user\_id>', methods=['DELETE'])

@jwt\_required()

def delete\_user(user\_id):

current\_user = get\_jwt\_identity()

if current\_user['role'] != 'admin':

return jsonify({"error": "Access forbidden"}), 403

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

if not user:

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

db.session.delete(user)

db.session.commit()

# Invalidate cache after deletion

cache.delete('users')

return jsonify({"message": "User deleted successfully"}), 200

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

app.run(debug=True)