Task 2

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

from sqlalchemy.exc import IntegrityError

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))

db = SQLAlchemy(app)

# 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)

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

# 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()

# Create a new user

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

def create\_user():

data = request.get\_json()

if not data:

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

name = data.get("name")

email = data.get("email")

age = data.get("age")

if not name or not email or not age:

return jsonify({"error": "Missing required fields: name, email, 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

new\_user = User(name=name, email=email, 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({"id": new\_user.id, "name": new\_user.name, "email": new\_user.email, "age": new\_user.age}), 201

# Get all users

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

def get\_users():

users = User.query.all()

return jsonify([{ "id": user.id, "name": user.name, "email": user.email, "age": user.age } for user in users]), 200

# Get a single user by ID

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

def get\_user(user\_id):

user = User.query.get(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}), 200

# Update a user by ID

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

def update\_user(user\_id):

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

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

# Delete a user by ID

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

def delete\_user(user\_id):

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

if not user:

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

db.session.delete(user)

db.session.commit()

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

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

app.run(debug=True)