from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

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

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

app.config["SQLALCHEMY\_DATABASE\_URI"] = "sqlite:///database.db"

db = SQLAlchemy(app)

# User model

class User(db.Model):

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

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

phone = db.Column(db.String(10), unique=True, nullable=False)

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

first\_name = db.Column(db.String(30), nullable=False)

last\_name = db.Column(db.String(30), nullable=False)

role = db.Column(db.String(10), nullable=False)

# Post model

class Post(db.Model):

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

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

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

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

user = db.relationship("User", backref=db.backref("posts", lazy=True))

# Comment model

class Comment(db.Model):

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

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

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

user = db.relationship("User", backref=db.backref("comments", lazy=True))

post\_id = db.Column(db.Integer, db.ForeignKey("post.id"), nullable=False)

post = db.relationship("Post", backref=db.backref("comments", lazy=True))

# User registration endpoint

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

def register():

data = request.get\_json()

email = data.get("email")

phone = data.get("phone")

password = data.get("password")

confirm\_password = data.get("confirm\_password")

# Validate input

if not email or not phone or not password or not confirm\_password:

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

if password != confirm\_password:

return jsonify({"error": "Passwords do not match"}), 400

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

return jsonify({"error": "Email or phone already registered"}), 400

# Hash password and create new user

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

new\_user = User(email=email, phone=phone, password\_hash=password\_hash, first\_name=data.get("first\_name"), last\_name=data.get("last\_name"), role=data.get("role"))

db.session.add(new\_user)

db.session.commit()

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

# User profile endpoint

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

def profile():

if request.method == "GET":

# Return current user's profile

user = User.query.filter\_by(id=1).first() # Replace 1 with the ID of the current user

return jsonify({

"email": user.email,

"phone": user.phone,

"first\_name": user.first\_name,

"last\_name": user.last\_name,

"role": user.role

}), 200

elif request.method == "POST":

# Update current user's profile

data = request.get\_json

@app.route("/profile", methods=["POST"])

def update\_profile():

data = request.get\_json()

# Update current user's profile

user = User.query.filter\_by(id=1).first() # Replace 1 with the ID of the current user

user.first\_name = data.get("first\_name")

user.last\_name = data.get("last\_name")

user.role = data.get("role")

db.session.commit()

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

Write a REST API with following features

. User with writer role can create posts and read, update and delete only their posts

Article fields

-title

-post

User with writer role can read comments on his posts.

User with editor role can read all posts and comment on posts.

# Posts endpoint

@app.route("/posts", methods=["GET", "POST"])

def posts():

if request.method == "POST":

# Create new post

data = request.get\_json()

title = data.get("title")

content = data.get("content")

# Validate input

if not title or not content:

return jsonify({"error": "Title and content are required"}), 400

# Check user's role

user = User.query.filter\_by(id=1).first() # Replace 1 with the ID of the current user

if user.role != "writer":

return jsonify({"error": "Only users with the 'writer' role can create posts"}), 403

# Create new post

new\_post = Post(title=title, content=content, user\_id=user.id)

db.session.add(new\_post)

db.session.commit()

return jsonify({"message": "Post created successfully"}), 201

elif request.method == "GET":

# Return posts for current user

user = User.query.filter\_by(id=1).first() # Replace 1 with the ID of the current user

posts = Post.query.filter\_by(user\_id=user.id).all()

return jsonify([{

"id": post.id,

"title": post.title,

"content": post.content,

"author": f"{post.user.first\_name} {post.user.last\_name}"

} for post in posts]), 200

# Single post endpoint

@app.route("/posts/<int:post\_id>", methods=["GET", "PUT", "DELETE"])

def post(post\_id):

# Retrieve post

post = Post.query.get(post\_id)

if not post:

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

# Check user's role

user = User.query.filter\_by(id=1).first() # Replace 1 with the ID of the current user

if user.role == "writer" and post.user\_id != user.id:

return jsonify({"error": "Only the author of the post can perform this action"}), 403

if request.method == "GET":

# Return post

return jsonify({

"id": post.id,

"title": post.title,

"content": post.content,

"author": f"{post.user.first\_name} {post.user.last\_name}"

}), 200

elif request.method == "PUT":

# Update post

data = request.get\_json()

post.title = data.get("title")

post.content = data.get("content")

db.session.commit()

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

elif request.method == "DELETE":

# Delete post

db.session.delete(post)

db.session.commit()

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

# Comments endpoint

@app.route("/posts/<int:post\_id>/comments", methods=["GET", "POST"])

def comments(post\_id):

# Retrieve post

post = Post.query.get(post\_id)

if not post:

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

if request.method == "POST":

# Create new comment

data = request.get\_json()

content = data.get("content")

# Validate input

if not content:

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

# Check user's role

user = User.query.filter\_by(id=1).first() # Replace 1 with the ID of the current user

if user.role == "writer" and post.user\_id != user.id:

return jsonify({"error": "Only users with the 'writer' role can comment on their own posts"}), 403

# Create new comment

new\_comment = Comment(content=content, user\_id=user.id, post\_id=post.id)

db.session.add(new\_comment)

db.session.commit()

return jsonify({"message": "Comment created successfully"}), 201

elif request.method == "GET":

# Return comments for post

comments = Comment.query.filter\_by(post\_id=post.id).all()

return jsonify([{

"id": comment.id,

"content": comment.content,

"author": f"{comment.user.first\_name} {comment.user.last\_name}"

} for comment in comments]), 200

# Comments endpoint

@app.route("/posts/<int:post\_id>/comments", methods=["GET", "POST"])

def comments(post\_id):

# Retrieve post

post = Post.query.get(post\_id)

if not post:

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

# Check user's role

user = User.query.filter\_by(id=1).first() # Replace 1 with the ID of the current user

if user.role != "editor":

return jsonify({"error": "Only users with the 'editor' role can access this endpoint"}), 403

if request.method == "POST":

# Create new comment

data = request.get\_json()

content = data.get("content")

# Validate input

if not content:

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

# Create new comment

new\_comment = Comment(content=content, user\_id=user.id, post\_id=post.id)

db.session.add(new\_comment)

db.session.commit()

return jsonify({"message": "Comment created successfully"}), 201

elif request.method == "GET":

# Return comments for post

comments = Comment.query.filter\_by(post\_id=post.id).all()

return jsonify([{

"id": comment.id,

"content": comment.content,

"author": f"{comment.user.first\_name} {comment.user.last\_name}"

} for comment in comments]), 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)