Coverage for **app.py**: 94%

258 statements | 243 run | 15 missing | 0 excluded

```python
import os
import re
import json
from flask import Flask, jsonify, request
from flask_cors import CORS
from werkzeug.security import check_password_hash
# Import Field from Pydantic
from pydantic import BaseModel, field_validator, ValidationError, Field
from typing import List, Optional
from datetime import datetime

# Import all 5 models and the db object from your models.py
from models import db, UserCredentials, UserProfile, EventDetails, VolunteerHistory, States

# App & DB Setup
BASE_DIR = os.path.abspath(os.path.dirname(__file__))
app = Flask(__name__)
CORS(app, origins=["http://localhost:5173", "http://127.0.0.1:5173"])
app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{os.path.join(BASE_DIR, 'volunteer.db')}"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Link the db object from models.py to our app
db.init_app(app)


# Pydantic Models for Data Validation

class UserRegistration(BaseModel):
    email: str
    password: str

    @field_validator('email')
    @classmethod
    def email_must_be_valid(cls, value):
        if not re.match(r"[^@]+@[^@]+\.[^@]+", value):
            raise ValueError('Email is not valid')
        return value

    @field_validator('password')
    @classmethod
    def password_complexity(cls, value):
        if len(value) < 8:
            raise ValueError('Password must be at least 8 characters long')
        return value

class UserLogin(BaseModel):
    email: str
    password: str

class ProfileUpdate(BaseModel):
    full_name: str
    address1: Optional[str] = None
    address2: Optional[str] = None
    city: Optional[str] = None
    state: Optional[str] = None

    zipcode: Optional[str] = Field(default=None, validation_alias='zip_code')

    skills: Optional[str] = None
    preferences: Optional[str] = None
    availability: Optional[str] = None

    @field_validator('skills', 'availability', 'preferences', mode='before')
    @classmethod
    def convert_list_to_string(cls, value):
        if isinstance(value, list):
            return ', '.join(value)
        return value

    @field_validator('zipcode')
    @classmethod
    def validate_zipcode(cls, value):
        if value is None or value == "":
            return value
        if not (value.isdigit() and (len(value) == 5 or len(value) == 9)):
            raise ValueError('Zip code must be 5 or 9 digits')
        return value

class EventCreation(BaseModel):
    event_name: str
    city: Optional[str] = None
    state: Optional[str] = None
    zipcode: Optional[str] = None
    skills: Optional[str] = None
    preferences: Optional[str] = None
    availability: Optional[str] = None

    @field_validator('skills', 'availability', 'preferences', mode='before')
    @classmethod
    def convert_list_to_string(cls, value):
        if isinstance(value, list):
            return ', '.join(value)
        return value


# API Endpoints (Built for your 5 tables)

@app.route('/register', methods=['POST'])
def register_user():
    try:
        user_data = UserRegistration(**request.json)

        if UserCredentials.query.filter_by(email=user_data.email).first():
            return jsonify({"message": "User with this email already exists"}), 409

        new_user = UserCredentials(
            email=user_data.email,
            role="admin" if user_data.email == "admin@example.com" else "volunteer"
        )
        new_user.set_password(user_data.password)

        db.session.add(new_user)
        db.session.commit()

        return jsonify({"message": "Registration successful", "user": {"email": new_user.email, "role": new_user.role}}), 20

    except ValidationError as e:
        return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
    except Exception as e:
        db.session.rollback()
        return jsonify({"message": "An internal error occurred", "error": str(e)}), 500

@app.route('/login', methods=['POST'])
def login_user():
    try:
        login_data = UserLogin(**request.json)

        user = UserCredentials.query.filter_by(email=login_data.email).first()

        if not user or not user.check_password(login_data.password):
            return jsonify({"message": "Invalid email or password"}), 401

        # --- FIX: Use db.session.get() ---
        profile_complete = db.session.get(UserProfile, user.id) is not None

        return jsonify({
            "message": "Login successful",
            "user": {
                "email": user.email,
                "role": user.role,
                "profileComplete": profile_complete
            }
        }), 200
    except ValidationError as e:
        return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
    except Exception as e:
        db.session.rollback()
        return jsonify({"message": "An internal error occurred", "error": str(e)}), 500

@app.route('/profile/<string:email>', methods=['GET', 'PUT'])
def user_profile(email):
    user_creds = UserCredentials.query.filter_by(email=email).first()
    if not user_creds:
        return jsonify({"message": "User not found"}), 404

    if request.method == 'GET':
        # --- FIX: Use db.session.get() ---
        profile = db.session.get(UserProfile, user_creds.id)
        if not profile:
            # Return empty object, but with 200 OK
            # This allows the frontend to know the user exists but has no profile
            return jsonify({}), 200

        skills_list = []
        if profile.skills:
            skills_list = [s.strip() for s in profile.skills.split(',')]

        availability_list = []
        if profile.availability:
            availability_list = [a.strip() for a in profile.availability.split(',')]

        return jsonify({
            "full_name": profile.full_name,
            "address1": profile.address1,
            "address2": profile.address2,
            "city": profile.city,
            "state": profile.state,
            "zip_code": profile.zipcode,
            "skills": skills_list,
            "preferences": profile.preferences,
            "availability": availability_list
        }), 200

    if request.method == 'PUT':
        try:
            profile_data = ProfileUpdate(**request.json)

            # --- FIX: Use db.session.get() ---
            profile = db.session.get(UserProfile, user_creds.id)

            if not profile:
                profile = UserProfile(id=user_creds.id, full_name=profile_data.full_name)
                # --- FIX: Added .session ---
                db.session.add(profile)

            # Update all fields from Pydantic model
            profile.full_name = profile_data.full_name
            profile.address1 = profile_data.address1
            profile.address2 = profile_data.address2
            profile.city = profile_data.city
            profile.state = profile_data.state
            profile.zipcode = profile_data.zipcode
            profile.skills = profile_data.skills
            profile.preferences = profile_data.preferences
            profile.availability = profile_data.availability

            db.session.commit()
            return jsonify({"message": "Profile updated successfully"}), 200

        except ValidationError as e:
            return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
        except Exception as e:
            db.session.rollback()
            return jsonify({"message": "An internal error occurred", "error": str(e)}), 500

@app.route('/events', methods=['GET', 'POST'])
def manage_events():
    if request.method == 'GET':
        events = EventDetails.query.all()
        event_list = [{
            "id": event.id,
            "event_name": event.event_name,
            "city": event.city,
            "state": event.state,
            "zipcode": event.zipcode,
            "skills": event.skills,
            "preferences": event.preferences,
            "availability": event.availability
        } for event in events]
        return jsonify(event_list), 200

    if request.method == 'POST':
        try:
            event_data = EventCreation(**request.json)

            new_event = EventDetails(
                event_name=event_data.event_name,
                city=event_data.city,
                state=event_data.state,
                zipcode=event_data.zipcode,
                skills=event_data.skills,
                preferences=event_data.preferences,
                availability=event_data.availability
            )
            db.session.add(new_event)
            db.session.commit()

            return jsonify({"message": "Event created successfully", "event_id": new_event.id}), 201

        except ValidationError as e:
            return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
        except Exception as e:
            db.session.rollback()
            return jsonify({"message": "An internal error occurred", "error": str(e)}), 500

@app.route('/signup', methods=['POST'])
def signup_for_event():
    data = request.get_json()
    email = data.get('email')
    event_id = data.get('event_id')

    if not email or not event_id:
        return jsonify({"message": "Email and Event ID are required"}), 400

    user_creds = UserCredentials.query.filter_by(email=email).first()
    if not user_creds:
        return jsonify({"message": "User not found"}), 404

    # --- FIX: Use db.session.get() ---
    event = db.session.get(EventDetails, event_id)
    if not event:
        return jsonify({"message": "Event not found"}), 404

    existing_signup = VolunteerHistory.query.filter_by(user_id=user_creds.id, event_id=event_id).first()
    if existing_signup:
        return jsonify({"message": "Already signed up for this event"}), 409

    new_signup = VolunteerHistory(user_id=user_creds.id, event_id=event_id)
    db.session.add(new_signup)
    db.session.commit()

    return jsonify({"message": "Successfully signed up for event"}), 201

@app.route('/history/<string:email>', methods=['GET'])
def get_volunteer_history(email):
    user_creds = UserCredentials.query.filter_by(email=email).first()
    if not user_creds:
        return jsonify({"message": "User not found"}), 404

    history_events = db.session.query(VolunteerHistory, EventDetails)\
                            .join(EventDetails, VolunteerHistory.event_id == EventDetails.id)\
                            .filter(VolunteerHistory.user_id == user_creds.id)\
                            .all()

    event_list = []
    for record, event in history_events:
        event_list.append({
            "event_id": event.id,
            "event_name": event.event_name,
            # --- FIX: Use .isoformat() for a standard, robust date format ---
            "participation_date": record.participation_date.isoformat()
        })

    return jsonify(event_list), 200

@app.route('/matching/<int:event_id>', methods=['GET'])
def get_volunteer_matches(event_id):
    # --- FIX: Use db.session.get() ---
    event = db.session.get(EventDetails, event_id)
    if not event:
        return jsonify({"message": "Event not found"}), 404

    if not event.skills:
         return jsonify([]), 200 # No skills to match

    event_skills_set = set(skill.strip().lower() for skill in event.skills.split(','))

    volunteers = db.session.query(UserProfile, UserCredentials)\
                            .join(UserCredentials, UserProfile.id == UserCredentials.id)\
                            .filter(UserCredentials.role == 'volunteer', UserProfile.skills != None)\
                            .all()

    matches = []
    for profile, user_creds in volunteers:
        if not profile.skills:
            continue

        profile_skills_set = set(skill.strip().lower() for skill in profile.skills.split(','))

        if event_skills_set.intersection(profile_skills_set):
            matches.append({
                "email": user_creds.email,
                "full_name": profile.full_name,
                "skills": profile.skills
            })

    return jsonify(matches), 200

```

```python
@app.route('/data/states', methods=['GET'])
def get_states():
    states = States.query.all()
    state_list = [{"code": s.code, "name": s.name} for s in states]
    return jsonify(state_list), 200

SKILLS_LIST = [
    "First Aid", "Logistics", "Event Setup", "Public Speaking",
    "Registration", "Tech Support", "Catering", "Marketing",
    "Fundraising", "Photography", "Social Media", "Team Leadership", "Translation"
]
@app.route('/data/skills', methods=['GET'])
def get_skills():
    return jsonify(SKILLS_LIST), 200

# Database Initializer (Seeder)
def init_db():
    """Create all tables and populate static/seed data."""
    db.create_all()

    # Populate States
    if States.query.count() == 0:
        print("Populating states...")
        states_data = [
            States(code='AL', name='Alabama'), States(code='AK', name='Alaska'),
            States(code='AZ', name='Arizona'), States(code='AR', name='Arkansas'),
            States(code='CA', name='California'), States(code='CO', name='Colorado'),
            States(code='CT', name='Connecticut'), States(code='DE', name='Delaware'),
            States(code='FL', name='Florida'), States(code='GA', name='Georgia'),
            States(code='HI', name='Hawaii'), States(code='ID', name='Idaho'),
            States(code='IL', name='Illinois'), States(code='IN', name='Indiana'),
            States(code='IA', name='Iowa'), States(code='KS', name='Kansas'),
            States(code='KY', name='Kentucky'), States(code='LA', name='Louisiana'),
            States(code='ME', name='Maine'), States(code='MD', name='Maryland'),
            States(code='MA', name='Massachusetts'), States(code='MI', name='Michigan'),
            States(code='MN', name='Minnesota'), States(code='MS', name='Mississippi'),
            States(code='MO', name='Missouri'), States(code='MT', name='Montana'),
            States(code='NE', name='Nebraska'), States(code='NV', name='Nevada'),
            States(code='NH', name='New Hampshire'), States(code='NJ', name='New Jersey'),
            # --- FIX: Corrected typo ---
            States(code='NM', name='New Mexico'), States(code='NY', name='New York'),
            States(code='NC', name='North Carolina'), States(code='ND', name='North Dakota'),
            States(code='OH', name='Ohio'), States(code='OK', name='Oklahoma'),
            States(code='OR', name='Oregon'),
            # --- FIX: Corrected typo ---
            States(code='PA', name='Pennsylvania'),
            States(code='RI', name='Rhode Island'), States(code='SC', name='South Carolina'),
            States(code='SD', name='South Dakota'), States(code='TN', name='Tennessee'),
            States(code='TX', name='Texas'), States(code='UT', name='Utah'),
            States(code='VT', name='Vermont'), States(code='VA', name='Virginia'),
            States(code='WA', name='Washington'), States(code='WV', name='West Virginia'),
            States(code='WI', name='Wisconsin'), States(code='WY', name='Wyoming')
        ]
        db.session.bulk_save_objects(states_data)
        db.session.commit()

    # Populate default Admin user
    if not UserCredentials.query.filter_by(email="admin@example.com").first():
        print("Creating admin user...")
        admin = UserCredentials(email="admin@example.com", role="admin")
        admin.set_password("AdminPassword1")
        db.session.add(admin)
        db.session.commit() # Commit admin first to get an ID

        # Create profile, linking the ID
        admin_profile = UserProfile(
            id=admin.id,
            full_name="Admin User",
            address1="123 Admin Way",
            address2=None,
            city="Houston",
            state="TX",
            zipcode="77001",
            skills="Team Leadership, Management"
        )
        db.session.add(admin_profile)
        db.session.commit()

    # Populate default Volunteer user
    if not UserCredentials.query.filter_by(email="volunteer@example.com").first():
        print("Creating volunteer user...")
        vol = UserCredentials(email="volunteer@example.com", role="volunteer")
        vol.set_password("Password1")
        db.session.add(vol)
        db.session.commit() # Commit volunteer to get ID

        # Add profile for the volunteer
        vol_profile = UserProfile(
            id=vol.id,
            full_name="John Doe",
            address1="123 Main St",
            address2=None,
            city="Houston",
            state="TX",
            zipcode="77002",
            skills="First Aid, Logistics",
            availability="2026-12-01, 2026-12-15"
        )
        db.session.add(vol_profile)
        db.session.commit()

    # Populate default Events
    if EventDetails.query.count() == 0:
        print("Populating events...")
        event1 = EventDetails(
            event_name="Community Food Drive",
            city="Houston",
            state="TX",
            zipcode="77002",
            skills="Logistics, Event Setup"
        )
        event2 = EventDetails(
            event_name="Park Cleanup Day",
            city="Houston",
            state="TX",
            zipcode="77056",
            skills="Event Setup, General Labor"
        )
        db.session.add_all([event1, event2])
        db.session.commit() # Commit events to get IDs

        # Add history for volunteer
        vol = UserCredentials.query.filter_by(email="volunteer@example.com").first()
        if vol:
            # Check if history record already exists (idempotency)
            existing_record = VolunteerHistory.query.filter_by(user_id=vol.id, event_id=event1.id).first()
            if not existing_record:
                history_record = VolunteerHistory(user_id=vol.id, event_id=event1.id)
                db.session.add(history_record)
                db.session.commit()


# Main Execution
if __name__ == '__main__':
    with app.app_context():
        init_db()
        print(f"Database initialized at: {os.path.join(BASE_DIR, 'volunteer.db')}")
    app.run(debug=True, port=5001)
```