

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

258 statements 243 run 15 missing 0 excluded

« prev ^ index » next coverage.py v7.11.0, created at 2025-10-31 19:00 -0500

```
1 import os
2 import re
3 import json
4 from flask import Flask, jsonify, request
5 from flask_cors import CORS
6 from werkzeug.security import check_password_hash
7 # Import Field from Pydantic
8 from pydantic import BaseModel, field_validator, ValidationError, Field
9 from typing import List, Optional
10 from datetime import datetime
11
12 # Import all 5 models and the db object from your models.py
13 from models import db, UserCredentials, UserProfile, EventDetails, VolunteerHistory, States
14
15 # App & DB Setup
16 BASE_DIR = os.path.abspath(os.path.dirname(__file__))
17 app = Flask(__name__)
18 CORS(app, origins=["http://localhost:5173", "http://127.0.0.1:5173"])
19 app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{"os.path.join(BASE_DIR, 'volunteer.db')}"}"
20 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
21
22 # Link the db object from models.py to our app
23 db.init_app(app)
24
25
26 # Pydantic Models for Data Validation
27
28 class UserRegistration(BaseModel):
29     email: str
30     password: str
31
32     @field_validator('email')
33     @classmethod
34     def email_must_be_valid(cls, value):
35         if not re.match(r"^[^@]+@^[^@]+\.[^@]+$", value):
36             raise ValueError('Email is not valid')
37         return value
38
39     @field_validator('password')
40     @classmethod
41     def password_complexity(cls, value):
42         if len(value) < 8:
43             raise ValueError('Password must be at least 8 characters long')
44         return value
45
46 class UserLogin(BaseModel):
47     email: str
48     password: str
49
50 class ProfileUpdate(BaseModel):
51     full_name: str
52     address1: Optional[str] = None
53     address2: Optional[str] = None
54     city: Optional[str] = None
55     state: Optional[str] = None
56
57     zipcode: Optional[str] = Field(default=None, validation_alias='zip_code')
58
59     skills: Optional[str] = None
60     preferences: Optional[str] = None
61     availability: Optional[str] = None
62
```

```

63 | @field_validator('skills', 'availability', 'preferences', mode='before')
64 | @classmethod
65 | def convert_list_to_string(cls, value):
66 |     if isinstance(value, list):
67 |         return ', '.join(value)
68 |     return value
69 |
70 | @field_validator('zipcode')
71 | @classmethod
72 | def validate_zipcode(cls, value):
73 |     if value is None or value == "":
74 |         return value
75 |     if not (value.isdigit() and (len(value) == 5 or len(value) == 9)):
76 |         raise ValueError('Zip code must be 5 or 9 digits')
77 |     return value
78 |
79 | class EventCreation(BaseModel):
80 |     event_name: str
81 |     city: Optional[str] = None
82 |     state: Optional[str] = None
83 |     zipcode: Optional[str] = None
84 |     skills: Optional[str] = None
85 |     preferences: Optional[str] = None
86 |     availability: Optional[str] = None
87 |
88 | @field_validator('skills', 'availability', 'preferences', mode='before')
89 | @classmethod
90 | def convert_list_to_string(cls, value):
91 |     if isinstance(value, list):
92 |         return ', '.join(value)
93 |     return value
94 |
95 |
96 | # API Endpoints (Built for your 5 tables)
97 |
98 | @app.route('/register', methods=['POST'])
99 | def register_user():
100 |     try:
101 |         user_data = UserRegistration(**request.json)
102 |
103 |         if UserCredentials.query.filter_by(email=user_data.email).first():
104 |             return jsonify({"message": "User with this email already exists"}), 409
105 |
106 |         new_user = UserCredentials(
107 |             email=user_data.email,
108 |             role="admin" if user_data.email == "admin@example.com" else "volunteer"
109 |         )
110 |         new_user.set_password(user_data.password)
111 |
112 |         db.session.add(new_user)
113 |         db.session.commit()
114 |
115 |         return jsonify({"message": "Registration successful", "user": {"email": new_user.email, "role": new_user.role}}), 201
116 |
117 |     except ValidationError as e:
118 |         return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
119 |     except Exception as e:
120 |         db.session.rollback()
121 |         return jsonify({"message": "An internal error occurred", "error": str(e)}), 500
122 |
123 | @app.route('/login', methods=['POST'])
124 | def login_user():
125 |     try:
126 |         login_data = UserLogin(**request.json)
127 |
128 |         user = UserCredentials.query.filter_by(email=login_data.email).first()
129 |
130 |         if not user or not user.check_password(login_data.password):
131 |             return jsonify({"message": "Invalid email or password"}), 401

```

```

132
133     # --- FIX: Use db.session.get() ---
134     profile_complete = db.session.get(UserProfile, user.id) is not None
135
136     return jsonify({
137         "message": "Login successful",
138         "user": {
139             "email": user.email,
140             "role": user.role,
141             "profileComplete": profile_complete
142         }
143     }), 200
144 except ValidationError as e:
145     return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
146 except Exception as e:
147     db.session.rollback()
148     return jsonify({"message": "An internal error occurred", "error": str(e)}), 500
149
150 @app.route('/profile/<string:email>', methods=['GET', 'PUT'])
151 def user_profile(email):
152     user_creds = UserCredentials.query.filter_by(email=email).first()
153     if not user_creds:
154         return jsonify({"message": "User not found"}), 404
155
156     if request.method == 'GET':
157         # --- FIX: Use db.session.get() ---
158         profile = db.session.get(UserProfile, user_creds.id)
159         if not profile:
160             # Return empty object, but with 200 OK
161             # This allows the frontend to know the user exists but has no profile
162             return jsonify({}), 200
163
164         skills_list = []
165         if profile.skills:
166             skills_list = [s.strip() for s in profile.skills.split(',')]
167
168         availability_list = []
169         if profile.availability:
170             availability_list = [a.strip() for a in profile.availability.split(',')]
171
172         return jsonify({
173             "full_name": profile.full_name,
174             "address1": profile.address1,
175             "address2": profile.address2,
176             "city": profile.city,
177             "state": profile.state,
178             "zip_code": profile.zipcode,
179             "skills": skills_list,
180             "preferences": profile.preferences,
181             "availability": availability_list
182         }), 200
183
184     if request.method == 'PUT':
185         try:
186             profile_data = ProfileUpdate(**request.json)
187
188             # --- FIX: Use db.session.get() ---
189             profile = db.session.get(UserProfile, user_creds.id)
190
191             if not profile:
192                 profile = UserProfile(id=user_creds.id, full_name=profile_data.full_name)
193                 # --- FIX: Added .session ---
194                 db.session.add(profile)
195
196                 # Update all fields from Pydantic model
197                 profile.full_name = profile_data.full_name
198                 profile.address1 = profile_data.address1
199                 profile.address2 = profile_data.address2
200                 profile.city = profile_data.city

```

```

201         profile.state = profile_data.state
202         profile.zipcode = profile_data.zipcode
203         profile.skills = profile_data.skills
204         profile.preferences = profile_data.preferences
205         profile.availability = profile_data.availability
206
207         db.session.commit()
208         return jsonify({"message": "Profile updated successfully"}), 200
209
210     except ValidationError as e:
211         return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
212     except Exception as e:
213         db.session.rollback()
214         return jsonify({"message": "An internal error occurred", "error": str(e)}), 500
215
216 @app.route('/events', methods=['GET', 'POST'])
217 def manage_events():
218     if request.method == 'GET':
219         events = EventDetails.query.all()
220         event_list = [{
221             "id": event.id,
222             "event_name": event.event_name,
223             "city": event.city,
224             "state": event.state,
225             "zipcode": event.zipcode,
226             "skills": event.skills,
227             "preferences": event.preferences,
228             "availability": event.availability
229         } for event in events]
230         return jsonify(event_list), 200
231
232     if request.method == 'POST':
233         try:
234             event_data = EventCreation(**request.json)
235
236             new_event = EventDetails(
237                 event_name=event_data.event_name,
238                 city=event_data.city,
239                 state=event_data.state,
240                 zipcode=event_data.zipcode,
241                 skills=event_data.skills,
242                 preferences=event_data.preferences,
243                 availability=event_data.availability
244             )
245             db.session.add(new_event)
246             db.session.commit()
247
248             return jsonify({"message": "Event created successfully", "event_id": new_event.id}), 201
249
250         except ValidationError as e:
251             return jsonify({"message": "Validation error", "errors": json.loads(e.json())}), 400
252         except Exception as e:
253             db.session.rollback()
254             return jsonify({"message": "An internal error occurred", "error": str(e)}), 500
255
256 @app.route('/signup', methods=['POST'])
257 def signup_for_event():
258     data = request.get_json()
259     email = data.get('email')
260     event_id = data.get('event_id')
261
262     if not email or not event_id:
263         return jsonify({"message": "Email and Event ID are required"}), 400
264
265     user_creds = UserCredentials.query.filter_by(email=email).first()
266     if not user_creds:
267         return jsonify({"message": "User not found"}), 404
268
269     # --- FIX: Use db.session.get() ---

```

```

270 |     event = db.session.get(EventDetails, event_id)
271 |     if not event:
272 |         return jsonify({"message": "Event not found"}), 404
273 |
274 |     existing_signup = VolunteerHistory.query.filter_by(user_id=user_creds.id, event_id=event_id).first()
275 |     if existing_signup:
276 |         return jsonify({"message": "Already signed up for this event"}), 409
277 |
278 |     new_signup = VolunteerHistory(user_id=user_creds.id, event_id=event_id)
279 |     db.session.add(new_signup)
280 |     db.session.commit()
281 |
282 |     return jsonify({"message": "Successfully signed up for event"}), 201
283 |
284 | @app.route('/history/<string:email>', methods=['GET'])
285 | def get_volunteer_history(email):
286 |     user_creds = UserCredentials.query.filter_by(email=email).first()
287 |     if not user_creds:
288 |         return jsonify({"message": "User not found"}), 404
289 |
290 |     history_events = db.session.query(VolunteerHistory, EventDetails)\
291 |         .join(EventDetails, VolunteerHistory.event_id == EventDetails.id)\
292 |         .filter(VolunteerHistory.user_id == user_creds.id)\
293 |         .all()
294 |
295 |     event_list = []
296 |     for record, event in history_events:
297 |         event_list.append({
298 |             "event_id": event.id,
299 |             "event_name": event.event_name,
300 |             # --- FIX: Use .isoformat() for a standard, robust date format ---
301 |             "participation_date": record.participation_date.isoformat()
302 |         })
303 |
304 |     return jsonify(event_list), 200
305 |
306 | @app.route('/matching/<int:event_id>', methods=['GET'])
307 | def get_volunteer_matches(event_id):
308 |     # --- FIX: Use db.session.get() ---
309 |     event = db.session.get(EventDetails, event_id)
310 |     if not event:
311 |         return jsonify({"message": "Event not found"}), 404
312 |
313 |     if not event.skills:
314 |         return jsonify([]), 200 # No skills to match
315 |
316 |     event_skills_set = set(skill.strip().lower() for skill in event.skills.split(','))
317 |
318 |     volunteers = db.session.query(UserProfile, UserCredentials)\
319 |         .join(UserCredentials, UserProfile.id == UserCredentials.id)\
320 |         .filter(UserCredentials.role == 'volunteer', UserProfile.skills != None)\
321 |         .all()
322 |
323 |     matches = []
324 |     for profile, user_creds in volunteers:
325 |         if not profile.skills:
326 |             continue
327 |
328 |         profile_skills_set = set(skill.strip().lower() for skill in profile.skills.split(','))
329 |
330 |         if event_skills_set.intersection(profile_skills_set):
331 |             matches.append({
332 |                 "email": user_creds.email,
333 |                 "full_name": profile.full_name,
334 |                 "skills": profile.skills
335 |             })
336 |
337 |     return jsonify(matches), 200
338 |

```

```

339 @app.route('/data/states', methods=['GET'])
340 def get_states():
341     states = States.query.all()
342     state_list = [{"code": s.code, "name": s.name} for s in states]
343     return jsonify(state_list), 200
344
345 SKILLS_LIST = [
346     "First Aid", "Logistics", "Event Setup", "Public Speaking",
347     "Registration", "Tech Support", "Catering", "Marketing",
348     "Fundraising", "Photography", "Social Media", "Team Leadership", "Translation"
349 ]
350 @app.route('/data/skills', methods=['GET'])
351 def get_skills():
352     return jsonify(SKILLS_LIST), 200
353
354 # Database Initializer (Seeder)
355 def init_db():
356     """Create all tables and populate static/seed data."""
357     db.create_all()
358
359     # Populate States
360     if States.query.count() == 0:
361         print("Populating states...")
362         states_data = [
363             States(code='AL', name='Alabama'), States(code='AK', name='Alaska'),
364             States(code='AZ', name='Arizona'), States(code='AR', name='Arkansas'),
365             States(code='CA', name='California'), States(code='CO', name='Colorado'),
366             States(code='CT', name='Connecticut'), States(code='DE', name='Delaware'),
367             States(code='FL', name='Florida'), States(code='GA', name='Georgia'),
368             States(code='HI', name='Hawaii'), States(code='ID', name='Idaho'),
369             States(code='IL', name='Illinois'), States(code='IN', name='Indiana'),
370             States(code='IA', name='Iowa'), States(code='KS', name='Kansas'),
371             States(code='KY', name='Kentucky'), States(code='LA', name='Louisiana'),
372             States(code='ME', name='Maine'), States(code='MD', name='Maryland'),
373             States(code='MA', name='Massachusetts'), States(code='MI', name='Michigan'),
374             States(code='MN', name='Minnesota'), States(code='MS', name='Mississippi'),
375             States(code='MO', name='Missouri'), States(code='MT', name='Montana'),
376             States(code='NE', name='Nebraska'), States(code='NV', name='Nevada'),
377             States(code='NH', name='New Hampshire'), States(code='NJ', name='New Jersey'),
378             # --- FIX: Corrected typo ---
379             States(code='NM', name='New Mexico'), States(code='NY', name='New York'),
380             States(code='NC', name='North Carolina'), States(code='ND', name='North Dakota'),
381             States(code='OH', name='Ohio'), States(code='OK', name='Oklahoma'),
382             States(code='OR', name='Oregon'),
383             # --- FIX: Corrected typo ---
384             States(code='PA', name='Pennsylvania'),
385             States(code='RI', name='Rhode Island'), States(code='SC', name='South Carolina'),
386             States(code='SD', name='South Dakota'), States(code='TN', name='Tennessee'),
387             States(code='TX', name='Texas'), States(code='UT', name='Utah'),
388             States(code='VT', name='Vermont'), States(code='VA', name='Virginia'),
389             States(code='WA', name='Washington'), States(code='WV', name='West Virginia'),
390             States(code='WI', name='Wisconsin'), States(code='WY', name='Wyoming')
391         ]
392         db.session.bulk_save_objects(states_data)
393         db.session.commit()
394
395     # Populate default Admin user
396     if not UserCredentials.query.filter_by(email="admin@example.com").first():
397         print("Creating admin user...")
398         admin = UserCredentials(email="admin@example.com", role="admin")
399         admin.set_password("AdminPassword1")
400         db.session.add(admin)
401         db.session.commit() # Commit admin first to get an ID
402
403     # Create profile, linking the ID
404     admin_profile = UserProfile(
405         id=admin.id,
406         full_name="Admin User",
407         address1="123 Admin Way",

```

```

408         address2=None,
409         city="Houston",
410         state="TX",
411         zipcode="77001",
412         skills="Team Leadership, Management"
413     )
414     db.session.add(admin_profile)
415     db.session.commit()
416
417     # Populate default Volunteer user
418     if not UserCredentials.query.filter_by(email="volunteer@example.com").first():
419         print("Creating volunteer user...")
420         vol = UserCredentials(email="volunteer@example.com", role="volunteer")
421         vol.set_password("Password1")
422         db.session.add(vol)
423         db.session.commit() # Commit volunteer to get ID
424
425         # Add profile for the volunteer
426         vol_profile = UserProfile(
427             id=vol.id,
428             full_name="John Doe",
429             address1="123 Main St",
430             address2=None,
431             city="Houston",
432             state="TX",
433             zipcode="77002",
434             skills="First Aid, Logistics",
435             availability="2026-12-01, 2026-12-15"
436         )
437         db.session.add(vol_profile)
438         db.session.commit()
439
440     # Populate default Events
441     if EventDetails.query.count() == 0:
442         print("Populating events...")
443         event1 = EventDetails(
444             event_name="Community Food Drive",
445             city="Houston",
446             state="TX",
447             zipcode="77002",
448             skills="Logistics, Event Setup"
449         )
450         event2 = EventDetails(
451             event_name="Park Cleanup Day",
452             city="Houston",
453             state="TX",
454             zipcode="77056",
455             skills="Event Setup, General Labor"
456         )
457         db.session.add_all([event1, event2])
458         db.session.commit() # Commit events to get IDs
459
460     # Add history for volunteer
461     vol = UserCredentials.query.filter_by(email="volunteer@example.com").first()
462     if vol:
463         # Check if history record already exists (idempotency)
464         existing_record = VolunteerHistory.query.filter_by(user_id=vol.id, event_id=event1.id).first()
465         if not existing_record:
466             history_record = VolunteerHistory(user_id=vol.id, event_id=event1.id)
467             db.session.add(history_record)
468             db.session.commit()
469
470
471     # Main Execution
472     if __name__ == '__main__':
473         with app.app_context():
474             init_db()
475             print(f"Database initialized at: {os.path.join(BASE_DIR, 'volunteer.db')}")
476         app.run(debug=True, port=5001)

```

