

[Return to "Full Stack Web Developer Nanodegree" in the classroom](#)

# Item Catalog

REVIEW

CODE REVIEW 3

HISTORY

▼ app.py 3

```
1 #!/usr/bin/python
```

AWESOME

## Tip ⚡

As a future reference tip, I'd strongly suggest having a look at how to structure a FI scalability.

We can structure app into modules/Flask blueprints instead of writing all code in o will make code hard to read and maintain when the application grows.

## Resources 📖

- <http://flask.pocoo.org/docs/0.12/patterns/packages/>
- <https://www.digitalocean.com/community/tutorials/how-to-structure-large-f>

```
2 # import libraries
3 import random
4 import string
5 import httpplib2
6 import json
7 import requests
8
```

```

9 # import Flask, SQLAlchemy, and OAuth
10 from flask import Flask, render_template, request, redirect
11 from flask import url_for, jsonify, make_response
12 from flask import session as login_session
13 from sqlalchemy import create_engine
14 from sqlalchemy.orm import sessionmaker
15 from database_setup import Base, CityDB, User
16 from oauth2client.client import flow_from_clientsecrets
17 from oauth2client.client import FlowExchangeError
18 # from oauth2client.client import AccessTokenCredentials
19
20 # set up app
21 app = Flask(__name__)
22 app.secret_key = 'itsasecret'
23 secret_file = json.loads(open('client_secret.json', 'r').read())
24 CLIENT_ID = secret_file['web']['client_id']
25 APPLICATION_NAME = 'Item-Catalog'
26
27 engine = create_engine('sqlite:///Catalog.db')
28 Base.metadata.bind = engine
29 DBSession = sessionmaker(bind=engine)
30 session = DBSession()
31
32
33 def check_user():
34     email = login_session['email']
35     return session.query(User).filter_by(email=email).one_or_none()
36
37
38 def check_admin():
39     return session.query(User).filter_by(
40         email='assirims2015@gmail.com').one_or_none()
41
42
43 def create_user():
44     name = login_session['name']
45     email = login_session['email']
46     url = login_session['img']
47     provider = login_session['provider']
48     newUser = User(name=name, email=email, image=url, provider=provider)
49     session.add(newUser)
50     session.commit()
51
52
53 def new_state():
54     state = ''.join(random.choice(string.ascii_uppercase +
55         string.digits) for x in xrange(32))
56     login_session['state'] = state
57     return state
58
59
60 def queryAllCities():
61     return session.query(CityDB).all()
62
63
64 # App Routation
65 @app.route('/')
66 @app.route('/cities/')
67 def show_cities():
68     cities = queryAllCities()

```

```

68     state = new_state()
69     return render_template('main.html', cities=cities, current
70
71
72
73 # New City
74 @app.route('/city/new/', methods=['GET', 'POST'])
75 def new_city():
76     if request.method == 'POST':
77         # check if user is logged in or not
78         if 'provider' in login_session and login_session['pro
79             city_name = request.form['city_name']
80             bookAuthor = request.form['region']
81             coverUrl = request.form['coverUrl']
82             description = request.form['description']
83             description = description.replace('\n', '<br>')
84             category = request.form['category']
85             user_id = check_user().id
86
87             if city_name and bookAuthor and coverUrl and desc
88                 and category:
89                 new_city = CityDB(
90                     city_name=city_name,
91                     region=bookAuthor,
92                     coverUrl=coverUrl,
93                     description=description,
94                     category=category,
95                     user_id=user_id,
96                 )
97                 session.add(new_city)
98                 session.commit()
99                 return redirect(url_for('show_cities'))
100     else:
101         state = new_state()
102         return render_template(
103             'newItem.html',
104             currentPage='new',
105             title='Add New city',
106             errorMsg='All Fields are Required!',
107             state=state,
108             login_session=login_session,
109         )
110     else:
111         state = new_state()
112         cities = queryAllCities()
113         return render_template(
114             'main.html',
115             cities=cities,
116             currentPage='main',
117             state=state,
118             login_session=login_session,
119             errorMsg='Please Login first to Add city!',
120         )
121     elif 'provider' in login_session and login_session['provi
122         != 'null':
123         state = new_state()
124         return render_template('newItem.html', currentPage='n
125                                 title='Add New city', state=st
126                                 login_session=login_session)
127     else:
128         state = new_state()

```

```

128         state = new_state()
129         cities = queryAllCities()
130         return render_template(
131             'main.html',
132             cities=cities,
133             currentPage='main',
134             state=state,
135             login_session=login_session,
136             errorMsg='Please Login first to Add city!',
137         )
138
139
140 # To show city of different category
141
142 @app.route('/cities/category/<string:category>/')
143 def sort_cities(category):
144     cities = session.query(CityDB).filter_by(category=category)
145     state = new_state()
146     return render_template(
147         'main.html',
148         cities=cities,
149         currentPage='main',
150         error='Sorry! No city in Database With This Genre :(',
151         state=state,
152         login_session=login_session)
153
154
155 # To show book detail
156
157 @app.route('/cities/category/<string:category>/<int:cityId>/')
158 def city_detail(category, cityId):
159     city = session.query(CityDB).filter_by(id=cityId,
160                                             category=category)
161     state = new_state()
162     if city:
163         return render_template('itemDetail.html', city=city,
164                               currentPage='detail', state=state,
165                               login_session=login_session)
166     else:
167         return render_template('main.html', currentPage='main',
168                               error=""No city Found with the genre",
169                               state=state,
170                               login_session=login_session)
171
172
173 # To edit city detail
174
175 @app.route('/cities/category/<string:category>/<int:cityId>/',
176            methods=['GET', 'POST'])
177 def edit_city_details(category, cityId):
178     city = session.query(CityDB).filter_by(id=cityId,
179                                             category=category)
180     if request.method == 'POST':
181
182         # check if user is logged in or not
183
184         if 'provider' in login_session and login_session['provider']
185             != 'null':
186             city_name = request.form['city_name']
187             bookAuthor = request.form['region']

```

Rate this review



```

188         coverUrl = request.form['coverUrl']
189         description = request.form['description']
190         category = request.form['category']
191         user_id = check_user().id
192         admin_id = check_admin().id
193
194         # check if city owner is same as logged in user
195
196         if city.user_id == user_id or user_id == admin_id:
197             if city_name and bookAuthor and coverUrl and
198                 and category:
199                 city.city_name = city_name
200                 city.region = bookAuthor
201                 city.coverUrl = coverUrl
202                 description = description.replace('\n',
203                 city.description = description
204                 city.category = category
205                 session.add(city)
206                 session.commit()
207                 return redirect(url_for('city_detail',
208                                     category=city.category,
209                                     cityId=city.id))
210             else:
211                 state = new_state()
212                 return render_template(
213                     'editItem.html',
214                     currentPage='edit',
215                     title='Edit city Details',
216                     city=city,
217                     state=state,
218                     login_session=login_session,
219                     errorMsg='All Fields are Required!',
220                     )
221         else:
222             state = new_state()
223             return render_template(
224                 'itemDetail.html',
225                 city=city,
226                 currentPage='detail',
227                 state=state,
228                 login_session=login_session,
229                 errorMsg='Sorry! The Owner can only edit
230
231         else:
232             state = new_state()
233             return render_template(
234                 'itemDetail.html',
235                 city=city,
236                 currentPage='detail',
237                 state=state,
238                 login_session=login_session,
239                 errorMsg='Please Login to Edit the city Deta
240
241         elif city:
242             state = new_state()
243             return render_template(
244                 'itemDetail.html',
245                 city=city,
246                 currentPage='detail',
247                 state=state,
248                 login_session=login_session,
249                 errorMsg='Please Login to Edit the city Deta

```

```

247         return render_template(
248             'editItem.html',
249             currentPage='edit',
250             title='Edit city Details',
251             city=city,
252             state=state,
253             login_session=login_session,
254         )
255     else:
256         return render_template(
257             'itemDetail.html',
258             city=city,
259             currentPage='detail',
260             state=state,
261             login_session=login_session,
262             errorMsg='Sorry! The Owner can only edit
263         else:
264         return render_template(
265             'itemDetail.html',
266             city=city,
267             currentPage='detail',
268             state=state,
269             login_session=login_session,
270             errorMsg='Please Login to Edit the city Detai
271         )
272     else:
273         state = new_state()
274         return render_template('main.html', currentPage='main
275             error=""Error Editing city! N
276             with this Category and city Id
277             state=state,
278             login_session=login_session)
279
280
281
282 @app.route('/cities/category/<string:category>/<int:cityId>/c
283 def delete_city(category, cityId):
284     city = session.query(CityDB).filter_by(category=category,
285     state = new_state()
286     if city:
287         # check if user is logged in or not
288         if 'provider' in login_session and login_session['pro
289             user_id = check_user().id
290             admin_id = check_admin().id
291             if user_id == city.user_id or user_id == admin_id
292                 session.delete(city)
293                 session.commit()
294                 return redirect(url_for('showcitys'))
295         else:
296             return render_template(
297                 'itemDetail.html',
298                 city=city,
299                 currentPage='detail',
300                 state=state,
301                 login_session=login_session,
302                 errorMsg='Sorry! Only the Owner Can delet
303             )
304     else:
305         return render_template(
306             'itemDetail.html',
307             city=city

```

```

307         city=city,
308         currentPage='detail',
309         state=state,
310         login_session=login_session,
311         errorMsg='Please Login to Delete the city!',
312     )
313     else:
314         return render_template('main.html', currentPage='main',
315                                error=""Error Deleting city!
316                                with this Category and city Id
317                                state=state,
318                                login_session=login_session)
319
320
321 # JSON Endpoints
322 @app.route('/cities.json/')
323 def citiesJSON():
324     cities = session.query(CityDB).all()
325     return jsonify(Cities=[city.serialize for city in cities])
326
327
328 @app.route('/cities/category/<string:category>.json/')

```

AWESOME

## Tip ⚡

If you want to build API endpoints with Flask, a scalable method is to use the `flask-restful` package.

- <https://flask-restful.readthedocs.io/en/latest/quickstart.html>

```

329 def categoryJSON(category):
330     cities = session.query(CityDB).filter_by(category=category)
331     return jsonify(Cities=[city.serialize for city in cities])
332
333
334 @app.route('/cities/category/<string:category>/<int:cityId>.')
335 def bookJSON(category, cityId):
336     city = session.query(CityDB).filter_by(category=category,
337                                             id=cityId).first()
338     return jsonify(city=city.serialize)
339
340
341 # google signin function
342 @app.route('/gconnect', methods=['POST'])
343 def gConnect():
344     if request.args.get('state') != login_session['state']:
345         response.make_response(json.dumps('Invalid State parameter'))
346         response.headers['Content-Type'] = 'application/json'
347         return response
348
349     # Obtain authorization code
350     code = request.data
351     try:

```

```

352         # Upgrade the authorization code into a credentials
353         oauth_flow = flow_from_clientsecrets('client_secret.json')
354         oauth_flow.redirect_uri = 'postmessage'
355         credentials = oauth_flow.step2_exchange(code)
356     except FlowExchangeError:
357         response = make_response(json.dumps("""Failed to upgrade
358         response.headers['Content-Type'] = 'application/json'
359         return response
360
361     # Check that the access token is valid.
362     access_token = credentials.access_token
363     url = \
364         'https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=
365         % access_token
366     header = httplib2.Http()
367     result = json.loads(header.request(url, 'GET')[1])
368
369     # If there was an error in the access token info, abort.
370     if result.get('error') is not None:
371         response = make_response(json.dumps(result.get('error')
372         response.headers['Content-Type'] = 'application/json'
373         return response
374
375     # Verify that the access token is used for the intended user.
376     gplus_id = credentials.id_token['sub']
377     if result['user_id'] != gplus_id:
378         response = make_response(json.dumps(
379             """Token's user ID does not
380             match given user ID.""",
381             401)
382         response.headers['Content-Type'] = 'application/json'
383         return response
384
385     # Verify that the access token is valid for this app.
386
387     if result['issued_to'] != CLIENT_ID:
388         response = make_response(json.dumps(
389             """Token's client ID
390             does not match app's.""",
391             401)
392         response.headers['Content-Type'] = 'application/json'
393         return response
394
395     # Store the access token in the session for later use.
396
397     stored_credentials = login_session.get('credentials')
398     stored_gplus_id = login_session.get('gplus_id')
399     if stored_credentials is not None and gplus_id == stored_gplus_id:
400         response = \
401             make_response(json.dumps('Current user is already logged in',
402             200)
403         response.headers['Content-Type'] = 'application/json'
404         return response
405
406     login_session['credentials'] = access_token
407     login_session['id'] = gplus_id
408
409     # Get user info
410
411     userinfo_url = 'https://www.googleapis.com/oauth2/v1/userinfo'

```



```

412     params = {'access_token': access_token, 'alt': 'json'}
413     answer = requests.get(userinfo_url, params=params)
414
415     data = answer.json()
416
417     # ADD PROVIDER TO LOGIN SESSION
418
419     login_session['name'] = data['name']
420     login_session['img'] = data['picture']
421     login_session['email'] = data['email']
422     login_session['provider'] = 'google'
423     if not check_user():
424         create_user()
425     return jsonify(name=login_session['name'],
426                   email=login_session['email'],
427                   img=login_session['img'])
428
429
430 # logout user
431
432 @app.route('/logout', methods=['post'])
433 def logout():
434
435     # Disconnect based on provider
436
437     if login_session.get('provider') == 'google':
438         return gdisconnect()
439     else:
440         response = make_response(json.dumps({'state': 'notConnected'}),
441                                   200)
442         response.headers['Content-Type'] = 'application/json'
443         return response
444
445
446 @app.route('/gdisconnect')
447 def gdisconnect():
448     access_token = login_session['credentials']
449
450     # Only disconnect a connected user.
451
452     if access_token is None:
453         response = make_response(json.dumps({'state': 'notConnected'}),
454                                   200)
455         response.headers['Content-Type'] = 'application/json'
456         return response
457     url = 'https://accounts.google.com/o/oauth2/revoke?token='
458         % access_token
459     header = httplib2.Http()
460     result = header.request(url, 'GET')[0]
461
462     if result['status'] == '200':
463
464         # Reset the user's session.
465
466         del login_session['credentials']
467         del login_session['id']
468         del login_session['name']
469         del login_session['email']
470         del login_session['img']
471         login_session['provider'] = 'null'

```

```

471         response = make_response(json.dumps({'state': 'logged
472                                     200)
473         response.headers['Content-Type'] = 'application/json
474         return response
475     else:
476         # if given token is invalid, unable to revoke token
477
478         response = make_response(json.dumps({'state': 'error
479                                     200)
480         response.headers['Content-Type'] = 'application/json
481         return response
482
483 if __name__ == '__main__':
484     app.debug = True
485     app.run(host='', port=5000)
486
487
488

```

AWESOME

## For Future Reference ⚡

As a developer, I also strongly recommend having a look using Docker as an altern (replacement) for Vagrant. Vagrant website has a quick comparison between the 2

- [Vagrant vs. Docker](#)

Docker containers are generally more light-weight and are much faster to start.

For a Flask tutorial, you can try this beginner-friendly tutorial:

- [Docker Development WorkFlow—a guide with Flask and Postgres](#)

► [templates/editItem.html](#)

► [templates/main.html](#)

► [templates/itemDetail.html](#)

► [templates/newItem.html](#)

► [templates/base.html](#)

► [static/js/app.js](#)

▶ `static/css/style.css`

▶ `static/mdl/material.indigo-red.min.css`

▶ `static/mdl/material.js`

▶ `README.md`

▶ `database_setup.py`

▶ `dummybooks.py`

[RETURN TO PATH](#)

---