



[< Back to Full Stack Web Developer Nanodegree](#)

# Item Catalog

REVIEW

CODE REVIEW 3

HISTORY

▼ application.py 3

1 # =====LICENSE\_START

AWESOME

**Tip** ⚡

As a future reference tip, I'd strongly suggest having a look at how to structure a Flask app that allows scalability.

We can structure app into modules/Flask blueprints instead of writing all code in one file as such, which will make code hard to read and maintain wh

## Resources

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

```

2 # =====
3 # Aimee Ukasick Apache-2.0
4 # =====
5 # Copyright (C) 2018 Aimee Ukasick . All rights reserved.
6 # =====
7 # This software file is distributed by Aimee Ukasick
8 # under the Apache License, Version 2.0 (the "License");
9 # you may not use this file except in compliance with the License.
10 # You may obtain a copy of the License at
11 #
12 # http://www.apache.org/licenses/LICENSE-2.0
13 #
14 # This file is distributed on an "AS IS" BASIS,
15 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
16 # See the License for the specific language governing permissions and
17 # limitations under the License.
18 # =====LICENSE_END
19 # =====
20
21 import datetime
22 import sys
23 from flask_sqlalchemy import SQLAlchemy
24 from sqlalchemy.orm.exc import NoResultFound
25 from flask_dance.contrib.github import make_github_blueprint, github
26 from flask_dance.consumer.backend.sqla import OAuthConsumerMixin, \
27     SQLAlchemyBackend
28 from flask_dance.consumer import oauth_authorized, oauth_error
29 from flask_login import (
30     LoginManager, UserMixin, current_user,
31     login_required, login_user, logout_user
32 )
33 from flask import (
34     Flask, flash, jsonify, make_response, redirect,
35     render_template, request, url_for
36 )
37 import movie_data
38 import json
39
40 app = Flask(__name__)

```

```

app = flask(__name__)
40 app.config['SECRET_KEY'] = 'thisissupposedtobeasecret'
42
43 # load github client_id and client_secret
44 CLIENT_ID = json.loads(
45     open('client_secret.json', 'r').read())['github']['client_id']
46 CLIENT_SECRET = json.loads(
47     open('client_secret.json', 'r').read())['github']['client_secret']
48
49 # create flask-dance blueprint and register it
50 blueprint = make_github_blueprint(client_id=CLIENT_ID,
51                                   client_secret=CLIENT_SECRET)
52 app.register_blueprint(blueprint, url_prefix='/login')
53
54 # set up the database
55 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///catalog.db'
56 db = SQLAlchemy()
57
58
59 # set up data models
60
61 class User(UserMixin, db.Model):
62     """
63     Creates a database table for User, required by flask-dance.
64     Extends flask_login.UserMixin and flask_sqlalchemy.SQLAlchemy.Model
65     """
66     # this must be 'id!' do not change to user_idnt or login_user will not work
67     id = db.Column(db.Integer, primary_key=True)
68     username = db.Column(db.String(250), unique=True)
69
70     @property
71     def serialize(self):
72         """Return object data in easily serializable format"""
73         return {
74             'id': self.id,
75             'username': self.username
76         }
77
78
79 class OAuth(OAuthConsumerMixin, db.Model):
80     """
81     Creates a database table for OAuth, required by flask-dance.
82     Extends flask_dance.consumer.backend.sqla.OAuthConsumerMixin and
83     flask_sqlalchemy.SQLAlchemy.Model
84     """
85     user_id = db.Column(db.Integer, db.ForeignKey(User.id))
86     github_user_id = db.Column(db.String, nullable=False)

```

```
86     github_user_id = db.Column(db.String, nullable=False)
87     user = db.relationship(User)
88
89     @property
90     def serialize(self):
91         """Return object data in easily serializable format"""
92         return {
93             'user_id': self.user_id,
94             'github_user_id': self.github_user_id,
95             'provider': self.provider
96         }
97
98
99     class Category(db.Model):
100         """
101         Creates a database table for Category
102         Extends flask_sqlalchemy.SQLAlchemy.Model
103         """
104         category_idnt = db.Column(db.Integer, primary_key=True)
105         name = db.Column(db.String(250), nullable=False)
106         create_dt = db.Column(db.DateTime, default=datetime.datetime.now())
107         create_by = db.Column(db.Integer, db.ForeignKey(User.id))
108         modify_dt = db.Column(db.DateTime, default=datetime.datetime.now())
109         user = db.relationship(User)
110
111         @property
112         def serialize(self):
113             """Return object data in easily serializable format"""
114             return {
115                 'name': self.name,
116                 'category_idnt': self.category_idnt
117             }
118
119
120     class Movie(db.Model):
121         """
122         Creates a database table for movies
123         Extends flask_sqlalchemy.SQLAlchemy.Model
124         """
125         movie_idnt = db.Column(db.Integer, primary_key=True)
126         title = db.Column(db.String(250), nullable=False)
127         description = db.Column(db.String(1000), nullable=False)
128         poster_img_url = db.Column(db.String(500), nullable=False)
129         trailer_url = db.Column(db.String(500), nullable=False)
130         create_dt = db.Column(db.DateTime, default=datetime.datetime.now())
131         create_by = db.Column(db.Integer, db.ForeignKey(User.id))
132         modify_dt = db.Column(db.DateTime, default=datetime.datetime.now())
```

```
132     modify_dt = db.Column(db.DateTime, default=datetime.datetime.now())
133     category_idnt = db.Column(db.Integer,
134                               db.ForeignKey(Category.category_idnt))
135     user = db.relationship(User)
136     category = db.relationship(Category)
137
138     @property
139     def serialize(self):
140         """Return object data in easily serializable format"""
141         return {
142             'title': self.title,
143             'description': self.description,
144             'poster_img_url': self.poster_img_url,
145             'trailer_url': self.trailer_url,
146             'category': self.serialize_category,
147             'create_dt': self.create_dt,
148             'modify_dt': self.modify_dt
149         }
150
151     @property
152     def serialize_category(self):
153         """Return Category in serializable format"""
154         return self.category.serialize
155
156
157 # set up login manager
158 login_manager = LoginManager()
159 login_manager.login_view = 'github.login'
160
161
162 @login_manager.user_loader
163 def load_user(user_id):
164     """
165     Required by Flask-Login to be implemented
166     :param user_id:
167     :return: User
168     """
169     return User.query.get(int(user_id))
170
171
172 # set up SQLAlchemy backend
173 blueprint.backend = SQLAlchemyBackend(OAuth, db.session, user=current_user)
174
175
176 # create/login local user on successful OAuth login
177 # this was copied from the Flask-Dance docs tutorial
178 @oauth.authorized_connect_via(blueprint)
```

```
178 @oauth_authorized.connect_via(blueprint)
179 def github_logged_in(blueprint, token):
180     """
181     Required by Flask-Dance
182     Called when a user has successfully authenticated with Github
183     :param blueprint: the flask-dance blueprint
184     :param token: oauth token
185     :return: False
186     """
187     if not token:
188         flash("Failed to log in with GitHub.", category="error")
189         return False
190
191     resp = blueprint.session.get("/user")
192     if not resp.ok:
193         msg = "Failed to fetch user info from GitHub."
194         flash(msg, category="error")
195         return False
196
197     github_info = resp.json()
198     github_user_id = str(github_info["id"])
199
200     # Find this OAuth token in the database, or create it
201     query = OAuth.query.filter_by(
202         provider=blueprint.name,
203         github_user_id=github_user_id,
204     )
205     try:
206         oauth = query.one()
207     except NoResultFound:
208         oauth = OAuth(
209             provider=blueprint.name,
210             github_user_id=github_user_id,
211             token=token,
212         )
213
214     if oauth.user:
215         login_user(oauth.user)
216         flash("Successfully signed in with GitHub.")
217
218     else:
219         # Create a new local user account for this user
220         username = github_info["login"]
221         user = User(username=username)
222         # Associate the new local user account with the OAuth token
223         oauth.user = user
224         # Save and commit our database models
```

```
224         # Save and commit our database models
225         db.session.add_all([user, oauth])
226         db.session.commit()
227         # Log in the new local user account
228         login_user(user)
229         flash("Successfully signed in with GitHub.")
230
231     # Disable Flask-Dance's default behavior for saving the OAuth token
232     return False
233
234
235 # notify on OAuth provider error
236 # copied from Flask-Dance docs tutorial
237 @oauth_error.connect_via(blueprint)
238 def github_error(blueprint, error, error_description=None, error_uri=None):
239     """
240     Required by Flask-Dance for when there is a provider error
241     :param blueprint:
242     :param error:
243     :param error_description:
244     :param error_uri:
245     """
246     msg = (
247         "OAuth error from {name}! "
248         "error={error} description={description} uri={uri}"
249     ).format(
250         name=blueprint.name,
251         error=error,
252         description=error_description,
253         uri=error_uri,
254     )
255     flash(msg, category="error")
256
257
258 @app.route('/logout')
259 @login_required
260 def logout():
261     """
262     Log the user out
263     User must be logged in
264     :return: redirect to Home page
265     """
266     logout_user()
267     return redirect(url_for('index'))
268
269
270 @app.route('/')

```

```
270 @app.route('/')
271 def index():
272     """
273     Home page
274     Fetches all categories
275     Fetches 10 most recently modified movies
276     :return: index.html
277     """
278     categories = Category.query.order_by(Category.name).all()
279
280     recent_items = Movie.query.order_by(Movie.modify_dt.desc()).limit(10).all()
281
282     # Render webpage
283     placeholder_txt = "Recently Modified Movies"
284     return render_template('index.html',
285                           categories=categories,
286                           movies=recent_items,
287                           placeholder_txt=placeholder_txt)
288
289
290 @app.route('/<int:category_id>')
291 def fetch_movies_for(category_id):
292     """
293     Fetches movies for the selected category
294     Updates the placeholder text for the Movies column
295     :param category_id:
296     :return: index.html
297     """
298     categories = Category.query.order_by(Category.name).all()
299     selected_category = Category.query.filter_by(
300         category_idnt=category_id).one()
301     movies = Movie.query.filter_by(category_idnt=category_id).order_by(
302         Movie.title.desc()).all()
303
304     # Render webpage
305     placeholder_txt = "{} Movies".format(selected_category.name)
306     return render_template('index.html',
307                           categories=categories,
308                           movies=movies,
309                           placeholder_txt=placeholder_txt)
310
311
312 @app.route('/view/<int:movie_id>')
313 def view_movie(movie_id):
314     """
315     Fetches the movie for the selected movie ID
316     :param movie_id:
```



```
316     .param movie_id.  
317     :return: item_view.html  
318     """  
319     movie = Movie.query.filter_by(movie_idnt=movie_id).one()  
320     return render_template('item_view.html',  
321                           movie=movie)  
322  
323  
324 @app.route('/view/<int:movie_id>/json')
```

▲  
AWESOME

## Tip ⚡

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

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

```
325 def view_movie_json(movie_id):  
326     """  
327     Fetches a single movie and returns a JSON string of the serialized Movie  
328     :param movie_id:  
329     :return: JSON string  
330     """  
331     movie = Movie.query.filter_by(movie_idnt=movie_id).one()  
332     return jsonify(Movie=movie.serialize)  
333  
334  
335 @app.route('/movies/json')  
336 def fetch_all_movies_json():  
337     """  
338     Fetches all movies, sorted by title  
339     :return: JSON list of movies  
340     """  
341     movies = Movie.query.order_by(Movie.title.desc()).all()  
342     return jsonify(json_list=[i.serialize for i in movies])  
343  
344  
345 @app.route('/add', methods=['GET', 'POST'])  
346 @login_required  
347 def add_movie():
```

```
348 """
349 Login Required!
350 Both GET and POST
351 Add and Edit use the same HTML page
352 For GET, creates a new Movie with blank data and returns item_edit.html
353 For POST, creates a new Movie, fills it with form data, saves, creates a
354 flash message, and redirects to the Home page
355 """
356 if request.method == 'POST':
357     # Get form fields
358     movie = Movie()
359     fill_movie(request.form, movie)
360     movie.create_dt = datetime.datetime.now()
361     movie.user = current_user
362     msg = "{} added".format(movie.title)
363     db.session.add(movie)
364     db.session.commit()
365     flash(msg)
366     return redirect(url_for('index'))
367 else:
368     categories = Category.query.order_by(Category.name).all()
369     movie = Movie()
370     movie.title = ''
371     movie.description = ''
372     movie.poster_img_url = ''
373     movie.trailer_url = ''
374     title = "Add Movie"
375     return render_template('item_edit.html',
376                           form_title=title,
377                           categories=categories,
378                           movie=movie,
379                           display_audit="false")
380
381
382 @app.route('/edit/<int:movie_id>', methods=['GET', 'POST'])
383 @login_required
384 def edit_movie(movie_id):
385     """
386     Login Required!
387     Both GET and POST
388     Add and Edit use the same HTML page; so whether or not to display audit
389     fields is set with display_audit
390     For GET, fetches the movie and returns item_edit.html
391     For POST, fetches the movie, fills it with form data, saves, creates a
392     flash message, and redirects to the Home page
393     """
```

```

393
394     if request.method == 'POST':
395         # retrieve form data and store
396         movie = Movie.query.filter_by(movie_idnt=movie_id).one()
397         movie = fill_movie(request.form, movie)
398         msg = "{} updated".format(movie.title)
399         db.session.add(movie)
400         db.session.commit()
401         flash(msg)
402         return redirect(url_for('index'))
403     else:
404         categories = Category.query.order_by(Category.name).all()
405         # fetch movie
406         movie = Movie.query.filter_by(movie_idnt=movie_id).one()
407         title = "Edit Movie"
408         return render_template('item_edit.html',
409                                form_title=title,
410                                categories=categories,
411                                movie=movie,
412                                display_audit="true")
413
414
415 def fill_movie(form, movie):
416     """
417     Fills the movie with data from the form
418     Called by both add_movie and edit_movie
419     :param form:
420     :param movie:
421     :return: movie
422     """
423     movie.title = form['title']
424     movie.category_idnt = form['dd_category']
425     movie.description = form['desc']
426     movie.poster_img_url = form['poster_img_url']
427     movie.trailer_url = form['trailer_url']
428     movie.modify_dt = datetime.datetime.now()
429     return movie
430
431
432 @app.route('/delete/<int:movie_id>', methods=['GET', 'POST'])
433 @login_required
434 def delete_movie(movie_id):
435     """
436     Both GET AND POST
437     GET: fetch movie and return item_delete.html
438     POST: Delete the movie from the database, create a flash msg, redirect Home
439     :param movie_id:

```

```
439     .param movie_id.  
440     :return:  
441     """  
442     if request.method == 'POST':  
443         # retrieve form data and store  
444         movie = Movie.query.filter_by(movie_idnt=movie_id).one()  
445         msg = "{} deleted".format(movie.title)  
446         db.session.delete(movie)  
447         db.session.commit()  
448         flash(msg)  
449         return redirect(url_for('index'))  
450     else:  
451         # fetch movie  
452         movie = Movie.query.filter_by(movie_idnt=movie_id).one()  
453         return render_template('item_delete.html',  
454                               movie=movie)  
455  
456  
457 # hook up extensions to app  
458 db.init_app(app)  
459 login_manager.init_app(app)  
460  
461 # create --setup switch to create and load database  
462 if __name__ == '__main__':  
463     if "--setup" in sys.argv:  
464         with app.app_context():  
465             db.create_all()  
466             db.session.commit()  
467             print("Database tables created")  
468             print("Loading data")  
469             movie_data.load_data(db.session)  
470             print("Data loaded")  
471     else:  
472         app.run(debug=True)  
473
```

  
AWESOME

## For Future Reference ⚡

As a developer, I also strongly recommend having a look using Docker as an alternative (or even replacement) for Vagrant. Vagrant website has a quick

- [Vagrant vs Docker](#)

• [Migrate to 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/layout.html`

▶ `templates/item_view.html`

▶ `templates/item_edit.html`

▶ `templates/item_delete.html`

▶ `templates/index.html`

▶ `static/css/Footer-Basic.css`

▶ `movie_data.py`

▶ `README.rst`

RETURN TO PATH