UDACITY

‹ 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

```
# =======================================================
# Aimee Ukasick Apache-2.0
# ===================================================================================
# Copyright (C) 2018 Aimee Ukasick . All rights reserved.
# ===================================================================================
# This software file is distributed by Aimee Ukasick
# under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# This file is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ===============LICENSE_END
# =========================================================

import datetime
import sys
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm.exc import NoResultFound
from flask_dance.contrib.github import make_github_blueprint, github
from flask_dance.consumer.backend.sqla import OAuthConsumerMixin, \
    SQLAlchemyBackend
from flask_dance.consumer import oauth_authorized, oauth_error
from flask_login import (
    LoginManager, UserMixin, current_user,
    login_required, login_user, logout_user
)
from flask import (
    Flask, flash, jsonify, make_response, redirect,
    render_template, request, url_for
)
import movie_data
import json

app = Flask(__name__)
app = Flask(__name__)
app.config['SECRET_KEY'] = 'thisissupposedtobeasecret'

# load github client_id and client_secret
CLIENT_ID = json.loads(
    open('client_secret.json', 'r').read())['github']['client_id']
CLIENT_SECRET = json.loads(
    open('client_secret.json', 'r').read())['github']['client_secret']

# create flask-dance blueprint and register it
blueprint = make_github_blueprint(client_id=CLIENT_ID,
                                  client_secret=CLIENT_SECRET)
app.register_blueprint(blueprint, url_prefix='/login')

# set up the database
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///catalog.db'
db = SQLAlchemy()


# set up data models

class User(UserMixin, db.Model):
    """
    Creates a database table for User, required by flask-dance.
    Extends flask_login.UserMixin and flask_sqlalchemy.SQLAlchemy.Model
    """
    # this must be 'id'! do not change to user_idnt or login_user will not work
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(250), unique=True)

    @property
    def serialize(self):
        """Return object data in easily serializable format"""
        return {
            'id': self.id,
            'username': self.username
        }


class OAuth(OAuthConsumerMixin, db.Model):
    """
    Creates a database table for OAuth, required by flask-dance.
    Extends flask_dance.consumer.backend.sqla.OAuthConsumerMixin and
    flask_sqlalchemy.SQLAlchemy.Model
    """
    user_id = db.Column(db.Integer, db.ForeignKey(User.id))
    github_user_id = db.Column(db.String, nullable=False)
    github_user_id = db.Column(db.String, nullable=False)
    user = db.relationship(User)

    @property
    def serialize(self):
        """Return object data in easily serializable format"""
        return {
            'user_id': self.user_id,
            'github_user_id': self.github_user_id,
            'provider': self.provider
        }


class Category(db.Model):
    """
    Creates a database table for Category
    Extends flask_sqlalchemy.SQLAlchemy.Model
    """
    category_idnt = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(250), nullable=False)
    create_dt = db.Column(db.DateTime, default=datetime.datetime.now())
    create_by = db.Column(db.Integer, db.ForeignKey(User.id))
    modify_dt = db.Column(db.DateTime, default=datetime.datetime.now())
    user = db.relationship(User)

    @property
    def serialize(self):
        """Return object data in easily serializable format"""
        return {
            'name': self.name,
            'category_idnt': self.category_idnt
        }


class Movie(db.Model):
    """
    Creates a database table for movies
    Extends flask_sqlalchemy.SQLAlchemy.Model
    """
    movie_idnt = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(250), nullable=False)
    description = db.Column(db.String(1000), nullable=False)
    poster_img_url = db.Column(db.String(500), nullable=False)
    trailer_url = db.Column(db.String(500), nullable=False)
    create_dt = db.Column(db.DateTime, default=datetime.datetime.now())
    create_by = db.Column(db.Integer, db.ForeignKey(User.id))
    modify_dt = db.Column(db.DateTime, default=datetime.datetime.now())
    modify_dt = db.Column(db.DateTime, default=datetime.datetime.now())
    category_idnt = db.Column(db.Integer,
                              db.ForeignKey(Category.category_idnt))
    user = db.relationship(User)
    category = db.relationship(Category)

    @property
    def serialize(self):
        """Return object data in easily serializable format"""
        return {
            'title': self.title,
            'description': self.description,
            'poster_img_url': self.poster_img_url,
            'trailer_url': self.trailer_url,
            'category': self.serialize_category,
            'create_dt': self.create_dt,
            'modify_dt': self.modify_dt
        }

    @property
    def serialize_category(self):
        """Return Category in serializable format"""
        return self.category.serialize


# set up login manager
login_manager = LoginManager()
login_manager.login_view = 'github.login'


@login_manager.user_loader
def load_user(user_id):
    """
    Required by Flask-Login to be implemented
    :param user_id:
    :return: User
    """
    return User.query.get(int(user_id))


# set up SQLAlchemy backend
blueprint.backend = SQLAlchemyBackend(OAuth, db.session, user=current_user)


# create/login local user on successful OAuth login
# this was copied from the Flask-Dance docs tutorial
@oauth_authorized.connect_via(blueprint)
```

```
@oauth_authorized.connect_via(blueprint)
def github_logged_in(blueprint, token):
    """
    Required by Flask-Dance
    Called when a user has successfully authenticated with Github
    :param blueprint: the flask-dance blueprint
    :param token: oauth token
    :return: False
    """
    if not token:
        flash("Failed to log in with GitHub.", category="error")
        return False

    resp = blueprint.session.get("/user")
    if not resp.ok:
        msg = "Failed to fetch user info from GitHub."
        flash(msg, category="error")
        return False

    github_info = resp.json()
    github_user_id = str(github_info["id"])

    # Find this OAuth token in the database, or create it
    query = OAuth.query.filter_by(
        provider=blueprint.name,
        github_user_id=github_user_id,
    )
    try:
        oauth = query.one()
    except NoResultFound:
        oauth = OAuth(
            provider=blueprint.name,
            github_user_id=github_user_id,
            token=token,
        )

    if oauth.user:
        login_user(oauth.user)
        flash("Successfully signed in with GitHub.")

    else:
        # Create a new local user account for this user
        username = github_info["login"]
        user = User(username=username)
        # Associate the new local user account with the OAuth token
        oauth.user = user
        # Save and commit our database models
        # Save and commit our database models
        db.session.add_all([user, oauth])
        db.session.commit()
        # Log in the new local user account
        login_user(user)
        flash("Successfully signed in with GitHub.")

    # Disable Flask-Dance's default behavior for saving the OAuth token
    return False


# notify on OAuth provider error
# copied from Flask-Dance docs tutorial
@oauth_error.connect_via(blueprint)
def github_error(blueprint, error, error_description=None, error_uri=None):
    """
    Required by Flask-Dance for when there is a provider error
    :param blueprint:
    :param error:
    :param error_description:
    :param error_uri:
    """
    msg = (
        "OAuth error from {name}! "
        "error={error} description={description} uri={uri}"
    ).format(
        name=blueprint.name,
        error=error,
        description=error_description,
        uri=error_uri,
    )
    flash(msg, category="error")


@app.route('/logout')
@login_required
def logout():
    """
    Log the user out
    User must be logged in
    :return: redirect to Home page
    """
    logout_user()
    return redirect(url_for('index'))


@app.route('/')
```

```
@app.route('/')
def index():
    """
    Home page
    Fetches all categories
    Fetches 10  most recently modified movies
    :return: index.html
    """
    categories = Category.query.order_by(Category.name).all()

    recent_items = Movie.query.order_by(Movie.modify_dt.desc()).limit(10).all()

    # Render webpage
    placeholder_txt = "Recently Modified Movies"
    return render_template('index.html',
                           categories=categories,
                           movies=recent_items,
                           placeholder_txt=placeholder_txt)


@app.route('/<int:category_id>')
def fetch_movies_for(category_id):
    """
    Fetches movies for the selected category
    Updates the placeholder text for the Movies column
    :param category_id:
    :return: index.html
    """
    categories = Category.query.order_by(Category.name).all()
    selected_category = Category.query.filter_by(
        category_idnt=category_id).one()
    movies = Movie.query.filter_by(category_idnt=category_id).order_by(
        Movie.title.desc()).all()

    # Render webpage
    placeholder_txt = "{} Movies".format(selected_category.name)
    return render_template('index.html',
                           categories=categories,
                           movies=movies,
                           placeholder_txt=placeholder_txt)


@app.route('/view/<int:movie_id>')
def view_movie(movie_id):
    """
    Fetches the movie for the selected movie ID
    :param movie_id:
```

```
    :param movie_id:
    :return: item_view.html
    """
    movie = Movie.query.filter_by(movie_idnt=movie_id).one()
    return render_template('item_view.html',
                           movie=movie)


@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

```
def view_movie_json(movie_id):
    """
    Fetches a single movie and returns a JSON string of the serialized Movie
    :param movie_id:
    :return: JSON string
    """
    movie = Movie.query.filter_by(movie_idnt=movie_id).one()
    return jsonify(Movie=movie.serialize)


@app.route('/movies/json')
def fetch_all_movies_json():
    """
    Fetches all movies, sorted by title
    :return: JSON list of movies
    """
    movies = Movie.query.order_by(Movie.title.desc()).all()
    return jsonify(json_list=[i.serialize for i in movies])


@app.route('/add', methods=['GET', 'POST'])
@login_required
def add_movie():
    """
    Login Required!
    Both GET and POST
    Add and Edit use the same HTML page
    For GET, creates a new Movie with blank data and returns item_edit.html
    For POST, creates a new Movie, fills it with form data, saves, creates a
    flash message, and redirects to the Home page
    """
    if request.method == 'POST':
        # Get form fields
        movie = Movie()
        fill_movie(request.form, movie)
        movie.create_dt = datetime.datetime.now()
        movie.user = current_user
        msg = "{} added".format(movie.title)
        db.session.add(movie)
        db.session.commit()
        flash(msg)
        return redirect(url_for('index'))
    else:
        categories = Category.query.order_by(Category.name).all()
        movie = Movie()
        movie.title = ''
        movie.description = ''
        movie.poster_img_url = ''
        movie.trailer_url = ''
        title = "Add Movie"
        return render_template('item_edit.html',
                               form_title=title,
                               categories=categories,
                               movie=movie,
                               display_audit="false")


@app.route('/edit/<int:movie_id>', methods=['GET', 'POST'])
@login_required
def edit_movie(movie_id):
    """
    Login Required!
    Both GET and POST
    Add and Edit use the same HTML page; so whether or not to display audit
    fields is set with display_audit
    For GET, fetches the movie and returns item_edit.html
    For POST, fetches the movie, fills it with form data, saves, creates a
    flash message, and redirects to the Home page
    """
    if request.method == 'POST':
        # retrieve form data and store
        movie = Movie.query.filter_by(movie_idnt=movie_id).one()
        movie = fill_movie(request.form, movie)
        msg = "{} updated".format(movie.title)
        db.session.add(movie)
        db.session.commit()
        flash(msg)
        return redirect(url_for('index'))
    else:
        categories = Category.query.order_by(Category.name).all()
        # fetch movie
        movie = Movie.query.filter_by(movie_idnt=movie_id).one()
        title = "Edit Movie"
        return render_template('item_edit.html',
                               form_title=title,
                               categories=categories,
                               movie=movie,
                               display_audit="true")


def fill_movie(form, movie):
    """
    Fills the movie with data from the form
    Called by both add_movie and edit_movie
    :param form:
    :param movie:
    :return: movie
    """
    movie.title = form['title']
    movie.category_idnt = form['dd_category']
    movie.description = form['desc']
    movie.poster_img_url = form['poster_img_url']
    movie.trailer_url = form['trailer_url']
    movie.modify_dt = datetime.datetime.now()
    return movie


@app.route('/delete/<int:movie_id>', methods=['GET', 'POST'])
@login_required
def delete_movie(movie_id):
    """
    Both GET AND POST
    GET: fetch movie and return item_delete.html
    POST: Delete the movie from the database, create a flash msg, redirect Home
    :param movie_id:
```

```
    :param movie_id:
    :return:
    """
    if request.method == 'POST':
        # retrieve form data and store
        movie = Movie.query.filter_by(movie_idnt=movie_id).one()
        msg = "{} deleted".format(movie.title)
        db.session.delete(movie)
        db.session.commit()
        flash(msg)
        return redirect(url_for('index'))
    else:
        # fetch movie
        movie = Movie.query.filter_by(movie_idnt=movie_id).one()
        return render_template('item_delete.html',
                               movie=movie)


# hook up extensions to app
db.init_app(app)
login_manager.init_app(app)

# create --setup switch to create and load database
if __name__ == '__main__':
    if "--setup" in sys.argv:
        with app.app_context():
            db.create_all()
            db.session.commit()
            print("Database tables created")
            print("Loading data")
            movie_data.load_data(db.session)
            print("Data loaded")
    else:
        app.run(debug=True)

```

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 quic

- Vagrant vs. Docker

- 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/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