UDACITY Logout

‹ Return to "Full Stack Web Developer Nanodegree" in the classroom

Item Catalog

REVIEW | CODE REVIEW 3 | HISTORY

▾ app.py 3

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

AWESOME

Tip ⚡

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

```
# import libraries
import random
import string
import httplib2
import json
import requests

```

```python
# import Flask, SqlAlchemy, and Oauth
from flask import Flask, render_template, request, redirect
from flask import url_for, jsonify, make_response
from flask import session as login_session
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from database_setup import Base, CityDB, User
from oauth2client.client import flow_from_clientsecrets
from oauth2client.client import FlowExchangeError
# from oauth2client.client import AccessTokenCredentials

# set up app
app = Flask(__name__)
app.secret_key = 'itsasecret'
secret_file = json.loads(open('client_secret.json', 'r').reac
CLIENT_ID = secret_file['web']['client_id']
APPLICATION_NAME = 'Item-Catalog'

engine = create_engine('sqlite:///Catalog.db')
Base.metadata.bind = engine
DBSession = sessionmaker(bind=engine)
session = DBSession()


def check_user():
    email = login_session['email']
    return session.query(User).filter_by(email=email).one_or_


def check_admin():
    return session.query(User).filter_by(
        email='assirims2015@gmail.com').one_or_none()


def create_user():
    name = login_session['name']
    email = login_session['email']
    url = login_session['img']
    provider = login_session['provider']
    newUser = User(name=name, email=email, image=url, provide
    session.add(newUser)
    session.commit()


def new_state():
    state = ''.join(random.choice(string.ascii_uppercase +
                    string.digits) for x in xrange(32))
    login_session['state'] = state
    return state


def queryAllCities():
    return session.query(CityDB).all()


# App Routation
@app.route('/')
@app.route('/cities/')
def show_cities():
    cities = queryAllCities()
    state = new_state()
    return render_template('main.html', cities=cities, curren


# New City
@app.route('/city/new/', methods=['GET', 'POST'])
def new_city():
    if request.method == 'POST':
        # check if user is logged in or not
        if 'provider' in login_session and login_session['pro
            city_name = request.form['city_name']
            bookAuthor = request.form['region']
            coverUrl = request.form['coverUrl']
            description = request.form['description']
            description = description.replace('\n', '<br>')
            category = request.form['category']
            user_id = check_user().id

            if city_name and bookAuthor and coverUrl and desc
                    and category:
                new_city = CityDB(
                    city_name=city_name,
                    region=bookAuthor,
                    coverUrl=coverUrl,
                    description=description,
                    category=category,
                    user_id=user_id,
                    )
                session.add(new_city)
                session.commit()
                return redirect(url_for('show_cities'))
            else:
                state = new_state()
                return render_template(
                    'newItem.html',
                    currentPage='new',
                    title='Add New city',
                    errorMsg='All Fields are Required!',
                    state=state,
                    login_session=login_session,
                    )
        else:
            state = new_state()
            cities = queryAllCities()
            return render_template(
                'main.html',
                cities=cities,
                currentPage='main',
                state=state,
                login_session=login_session,
                errorMsg='Please Login first to Add city!',
                )
    elif 'provider' in login_session and login_session['prov
            != 'null':
        state = new_state()
        return render_template('newItem.html', currentPage='r
                               title='Add New city', state=st
                               login_session=login_session)
    else:
        state = new_state()
        state = new_state()
        cities = queryAllCities()
        return render_template(
            'main.html',
            cities=cities,
            currentPage='main',
            state=state,
            login_session=login_session,
            errorMsg='Please Login first to Add city!',
            )


# To show city of different category

@app.route('/cities/category/<string:category>/')
def sort_cities(category):
    cities = session.query(CityDB).filter_by(category=categor
    state = new_state()
    return render_template(
        'main.html',
        cities=cities,
        currentPage='main',
        error='Sorry! No city in Database With This Genre :(
        state=state,
        login_session=login_session)


# To show book detail

@app.route('/cities/category/<string:category>/<int:cityId>/
def city_detail(category, cityId):
    city = session.query(CityDB).filter_by(id=cityId,
                                           category=category
    state = new_state()
    if city:
        return render_template('itemDetail.html', city=city,
                               currentPage='detail', state=st
                               login_session=login_session)
    else:
        return render_template('main.html', currentPage='mair
                               error="""No city Found with th
                               state=state,
                               login_session=login_session)


# To edit city detail

@app.route('/cities/category/<string:category>/<int:cityId>/
           methods=['GET', 'POST'])
def edit_city_details(category, cityId):
    city = session.query(CityDB).filter_by(id=cityId,
                                           category=category
    if request.method == 'POST':

        # check if user is logged in or not

        if 'provider' in login_session and login_session['pro
                'null':
            city_name = request.form['city_name']
            bookAuthor = request.form['region']
```

Rate this review

```
            coverUrl = request.form['coverUrl']
            description = request.form['description']
            category = request.form['category']
            user_id = check_user().id
            admin_id = check_admin().id

            # check if city owner is same as logged in user o

            if city.user_id == user_id or user_id == admin_ic
                if city_name and bookAuthor and coverUrl and
                        and category:
                    city.city_name = city_name
                    city.region = bookAuthor
                    city.coverUrl = coverUrl
                    description = description.replace('\n',
                    city.description = description
                    city.category = category
                    session.add(city)
                    session.commit()
                    return redirect(url_for('city_detail',
                                    category=city.category,
                                    cityId=city.id))
                else:
                    state = new_state()
                    return render_template(
                        'editItem.html',
                        currentPage='edit',
                        title='Edit city Details',
                        city=city,
                        state=state,
                        login_session=login_session,
                        errorMsg='All Fields are Required!',
                        )
            else:
                state = new_state()
                return render_template(
                    'itemDetail.html',
                    city=city,
                    currentPage='detail',
                    state=state,
                    login_session=login_session,
                    errorMsg='Sorry! The Owner can only edit
        else:
            state = new_state()
            return render_template(
                'itemDetail.html',
                city=city,
                currentPage='detail',
                state=state,
                login_session=login_session,
                errorMsg='Please Login to Edit the city Detai
                )
    elif city:
        state = new_state()
        if 'provider' in login_session and login_session['pro
                != 'null':
            user_id = check_user().id
            admin_id = check_admin().id
            if user_id == city.user_id or user_id == admin_ic
                city.description = city.description.replace(
                return render_template(
                    'editItem.html',
                    currentPage='edit',
                    title='Edit city Details',
                    city=city,
                    state=state,
                    login_session=login_session,
                    )
            else:
                return render_template(
                    'itemDetail.html',
                    city=city,
                    currentPage='detail',
                    state=state,
                    login_session=login_session,
                    errorMsg='Sorry! The Owner can only edit
        else:
            return render_template(
                'itemDetail.html',
                city=city,
                currentPage='detail',
                state=state,
                login_session=login_session,
                errorMsg='Please Login to Edit the city Deta
                )
    else:
        state = new_state()
        return render_template('main.html', currentPage='mai
                               error="""Error Editing city! N
                               with this Category and city Id
                               state=state,
                               login_session=login_session)


@app.route('/cities/category/<string:category>/<int:cityId>/d
def delete_city(category, cityId):
    city = session.query(CityDB).filter_by(category=category,
    state = new_state()
    if city:
        # check if user is logged in or not
        if 'provider' in login_session and login_session['pro
            user_id = check_user().id
            admin_id = check_admin().id
            if user_id == city.user_id or user_id == admin_id
                session.delete(city)
                session.commit()
                return redirect(url_for('showcitys'))
            else:
                return render_template(
                    'itemDetail.html',
                    city=city,
                    currentPage='detail',
                    state=state,
                    login_session=login_session,
                    errorMsg='Sorry! Only the Owner Can delet
                    )
        else:
            return render_template(
                'itemDetail.html',
                city=city
                city=city,
                currentPage='detail',
                state=state,
                login_session=login_session,
                errorMsg='Please Login to Delete the city!',
                )
    else:
        return render_template('main.html', currentPage='mair
                               error="""Error Deleting city!
                               with this Category and city Ic
                               state=state,
                               login_session=login_session)


# JSON Endpoints
@app.route('/cities.json/')
def citiesJSON():
    cities = session.query(CityDB).all()
    return jsonify(Cities=[city.serialize for city in cities


@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 `flas`

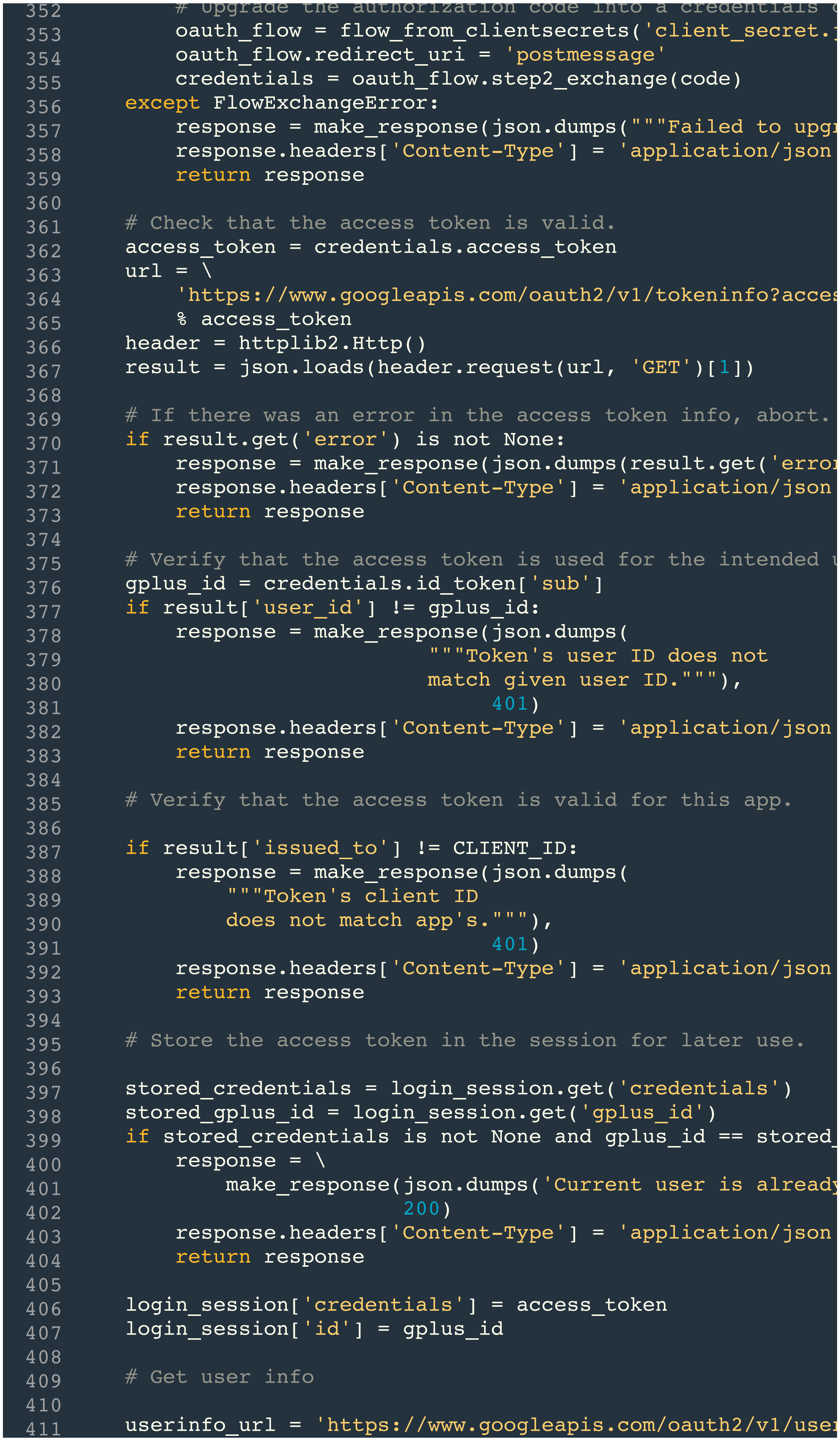
- https://flask-restful.readthedocs.io/en/latest/quickstart.html

```
def categoryJSON(category):
    cities = session.query(CityDB).filter_by(category=categor
    return jsonify(Cities=[city.serialize for city in cities


@app.route('/cities/category/<string:category>/<int:cityId>.
def bookJSON(category, cityId):
    city = session.query(CityDB).filter_by(category=category,
                                           id=cityId).first(
    return jsonify(city=city.serialize)


# google signin function
@app.route('/gconnect', methods=['POST'])
def gConnect():
    if request.args.get('state') != login_session['state']:
        response.make_response(json.dumps('Invalid State para
        response.headers['Content-Type'] = 'application/json
        return response

    # Obtain authorization code
    code = request.data
    try:
```



```
        # Upgrade the authorization code into a credentials
        oauth_flow = flow_from_clientsecrets('client_secret.
        oauth_flow.redirect_uri = 'postmessage'
        credentials = oauth_flow.step2_exchange(code)
    except FlowExchangeError:
        response = make_response(json.dumps("""Failed to upg
        response.headers['Content-Type'] = 'application/json
        return response

    # Check that the access token is valid.
    access_token = credentials.access_token
    url = \
        'https://www.googleapis.com/oauth2/v1/tokeninfo?acces
        % access_token
    header = httplib2.Http()
    result = json.loads(header.request(url, 'GET')[1])

    # If there was an error in the access token info, abort.
    if result.get('error') is not None:
        response = make_response(json.dumps(result.get('erro
        response.headers['Content-Type'] = 'application/json
        return response

    # Verify that the access token is used for the intended
    gplus_id = credentials.id_token['sub']
    if result['user_id'] != gplus_id:
        response = make_response(json.dumps(
                            """Token's user ID does not
                            match given user ID."""),
                                 401)
        response.headers['Content-Type'] = 'application/json
        return response

    # Verify that the access token is valid for this app.

    if result['issued_to'] != CLIENT_ID:
        response = make_response(json.dumps(
            """Token's client ID
            does not match app's."""),
                                 401)
        response.headers['Content-Type'] = 'application/json
        return response

    # Store the access token in the session for later use.

    stored_credentials = login_session.get('credentials')
    stored_gplus_id = login_session.get('gplus_id')
    if stored_credentials is not None and gplus_id == stored_
        response = \
            make_response(json.dumps('Current user is already
                          200)
        response.headers['Content-Type'] = 'application/json
        return response

    login_session['credentials'] = access_token
    login_session['id'] = gplus_id

    # Get user info

    userinfo_url = 'https://www.googleapis.com/oauth2/v1/user
    params = {'access_token': access_token, 'alt': 'json'}
    answer = requests.get(userinfo_url, params=params)

    data = answer.json()

    # ADD PROVIDER TO LOGIN SESSION

    login_session['name'] = data['name']
    login_session['img'] = data['picture']
    login_session['email'] = data['email']
    login_session['provider'] = 'google'
    if not check_user():
        create_user()
    return jsonify(name=login_session['name'],
                   email=login_session['email'],
                   img=login_session['img'])


# logout user

@app.route('/logout', methods=['post'])
def logout():

    # Disconnect based on provider

    if login_session.get('provider') == 'google':
        return gdisconnect()
    else:
        response = make_response(json.dumps({'state': 'notCo
                                 200)
        response.headers['Content-Type'] = 'application/json
        return response


@app.route('/gdisconnect')
def gdisconnect():
    access_token = login_session['credentials']

    # Only disconnect a connected user.

    if access_token is None:
        response = make_response(json.dumps({'state': 'notCo
                                 200)
        response.headers['Content-Type'] = 'application/json
        return response
    url = 'https://accounts.google.com/o/oauth2/revoke?token
        % access_token
    header = httplib2.Http()
    result = header.request(url, 'GET')[0]

    if result['status'] == '200':

        # Reset the user's session.

        del login_session['credentials']
        del login_session['id']
        del login_session['name']
        del login_session['email']
        del login_session['img']
        login_session['provider'] = 'null'
        response = make_response(json.dumps({'state': 'logge
                                 200)
        response.headers['Content-Type'] = 'application/json
        return response
    else:

        # if given token is invalid, unable to revoke token

        response = make_response(json.dumps({'state': 'error
                                 200)
        response.headers['Content-Type'] = 'application/json
        return response

if __name__ == '__main__':
    app.debug = True
    app.run(host='', port=5000)

```

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