Fyyur: Artist Booking Site

REVIEW | CODE REVIEW 7 | HISTORY

app.py 4

```
#----------------------------------------------------------------------------#
# Imports
#----------------------------------------------------------------------------#

import json
import dateutil.parser
import babel
from flask import Flask, render_template, request, Response, flash, redirect, url_for
import logging
from logging import Formatter, FileHandler
from flask_wtf import Form
from forms import *
from Models import *
```

AWESOME

The code organization implements some separation of concerns by splitting your endpoint and model processing. This strat will always exercise; implementing modules help you document, reuse & test your functionalities.

If you need to implement a new model or change an existing one, it will be easier having them on a dedicated codebase rath single file.

To see more details and insights around this practice, you can check this Python handbook.

```

#----------------------------------------------------------------------------#
# correcting a BUG that occurs on dateutil parser collection
#----------------------------------------------------------------------------#
import collections
collections.Callable = collections.abc.Callable
#----------------------------------------------------------------------------#
# End correcting a BUG
#----------------------------------------------------------------------------#

#----------------------------------------------------------------------------#
# App Config.
#----------------------------------------------------------------------------#




#----------------------------------------------------------------------------#
# Filters.
#----------------------------------------------------------------------------#

def format_datetime(value, format='medium'):
  print("++++++++++++++")
  print("value "+value)
  date = dateutil.parser.parse(value)
  print("Date "+str(date))
  if format == 'full':
      format="EEEE MMMM, d, y 'at' h:mma"
  elif format == 'medium':
      format="EE MM, dd, y h:mma"
  return babel.dates.format_datetime(date, format, locale='en')
```

Rate this review

START

```
45
46 app.jinja_env.filters['datetime'] = format_datetime
47
48 #----------------------------------------------------------------------------#
49 # Controllers.
50 #----------------------------------------------------------------------------#
51
52 @app.route('/')
53 def index():
54   return render_template('pages/home.html')
55
56
57 #  Venues
58 #  ----------------------------------------------------------------
59 # Looking at the data provided with the question it appears that we need to
60 # pull all venues and then pull all shows and then rearrange the venue based on city,
61 # and do a count on the number of upcoming shows.
62 #
63 #
64 @app.route('/venues')
65 def venues():
66     data =[]
67     current_date = datetime.now()
68     venues = Venue.query.all()
69     unique_venues = set()
70     for venue in venues:
71         unique_venues.add((venue.city, venue.state))
72     for everylocation in unique_venues:
73         data.append({"city": everylocation[0],"state": everylocation[1],"venues": []})
74     for venue in venues:
75         num_upcoming_shows = 0
76         shows = Show.query.filter_by(venue_id=venue.id).all()
```

REQUIRED

⚠️ Your code does not make usage of JOIN statement

One of the challenges of this project is using `JOIN` statements to fetch related data based on a foreign key strategy; This is complex and efficient data models. Sure, you can acquire the same data by performing finds and iterating over the results, this will prevent your application from scaling when your dataset or number of users grows.

There are several types of join to associate tables based on the goals you need to achieve. I recommend the following articl subject:

1. SQL JOIN initial documentation
2. Different JOIN strategies explained with visuals

```
77         for show in shows:
78             if show.start_time > current_date:
79                 num_upcoming_shows += 1
80         for dataitem in data:
81             if venue.state == dataitem['state'] and venue.city == dataitem['city']:
82                 dataitem['venues'].append({"id": venue.id,"name": venue.name,"num_upcoming_shows": nu
83     return render_template('pages/venues.html', areas=data);
84
85 @app.route('/venues/search', methods=['POST'])
86 def search_venues():
87     search_term = request.form.get('search_term', '')
88     search_termlocal = '%{0}%'.format(search_term)
89     result = Venue.query.filter(Venue.name.ilike(search_termlocal))
90     response={ "count": result.count(),"data": result }
91     return render_template('pages/search_venues.html', results=response, search_term=search_term)
92
93
94 @app.route('/venues/<int:venue_id>')
95 def show_venue(venue_id):
96     venue = Venue.query.get(venue_id)
97     listofshows=Show.query.filter_by(venue_id=venue_id).all()
98     nextshowlist=[]
99     prevshowlist=[]
100    current_date = datetime.now()
101    for show_listitem in listofshows:
102        artist = Artist.query.get(show_listitem.artist_id)
103        showDictionaryItem ={
104            "artist_id": show_listitem.artist_id,
105            "artist_name": artist.name,
106            "artist_image_link": artist.image_link,
107            "start_time": format_datetime(str(show_listitem.start_time))
        }
108        if(show_listitem.start_time>current_date):
110            nextshowlist.append(showDictionaryItem)
```

```
        else:
            prevshowlist.append(showDictionaryItem)

    venuedictionary={
        "id": venue.id,
        "name": venue.name,
        "genres": venue.genres,
        "address": venue.address,
        "city": venue.city,
        "state": venue.state,
        "phone": venue.phone,
        "website_link": venue.website_link,
        "facebook_link": venue.facebook_link,
        "seeking_talent": venue.seeking_talent,
        "seeking_description":venue.seeking_description,
        "image_link": venue.image_link,
        "past_shows": prevshowlist,
        "upcoming_shows": nextshowlist,
        "past_shows_count": len(prevshowlist),
        "upcoming_shows_count": len(nextshowlist)
    }
    return render_template('pages/show_venue.html', venue=venuedictionary)

#  Create Venue
#  ----------------------------------------------------------------

@app.route('/venues/create', methods=['GET'])
def create_venue_form():
  form = VenueForm()
  return render_template('forms/new_venue.html', form=form)

@app.route('/venues/create', methods=['POST'])
def create_venue_submission():
    error = False
    try:
        form = VenueForm()
        venue = Venue(name=form.name.data, city=form.city.data, state=form.state.data,
                    address=form.address.data,phone=form.phone.data, image_link=form.image_link.data,
                    genres=form.genres.data, facebook_link=form.facebook_link.data, seeking_descripti
                    website_link=form.website_link.data, seeking_talent=form.seeking_talent.data)
        db.session.add(venue)
        db.session.commit()
    except:
        db.session.rollback()
        print(sys.exc_info())
        error=True
    finally:
        db.session.close()
```

AWESOME

The session is always closed after manipulation; this is similar to turning off the lights/equipment after you stop using them

Obviously, if we think about one or two connections laying around, the impact wouldn't be huge. Still, considering scalable a open hundreds or thousands of connections, it is crucial to control that. A careful engineer can save money and collaborate application runtime by doing so.

Some technologies are somewhat more intelligent and can close the connections automatically, mitigating that issue. Still, I analyzing if that's the case. Check out below some cool articles:

1. AWS - Postgres memory leak investigation
2. Google - MySQL connection management
3. Three-tier application connection management architecture

```
        if not error:
              # on successful db insert, flash success
              flash('Venue ' + request.form['name'] + ' was successfully listed!')
              return render_template('pages/home.html')
        else:
            flash('An error occurred. Venue ' + request.form['name'] + ' could not be listed.')
            abort (400)



@app.route('/venues/<venue_id>', methods=['DELETE'])
def delete_venue(venue_id):
  try:
    venue = Venue.query.get(venue_id)
    venue_name = venue.name
    db.session.delete(venue)
    db.session.commit()
    flash('Venue ' + venue_name + ' deleted successfully')
  except:
```

```
    flash('an error occured while deleting Venue ' + venue_name)
    db.session.rollback()
  finally:
    db.session.close()
  return None

#  Artists
#  ----------------------------------------------------------------
@app.route('/artists')
def artists():
  artists = Artist.query.all()
  data=[]
  for artist in artists:
      data.append({
      "id": artist.id,
      "name": artist.name
      })
  return render_template('pages/artists.html', artists=data)

@app.route('/artists/search', methods=['POST'])
def search_artists():
    search_term = request.form.get('search_term', '')
    search_termlocal = '%{0}%'.format(search_term)
    artists = Artist.query.filter(Artist.name.ilike(search_termlocal)).all()
```

AWESOME

Using `ilike` allows the query to be case-insensitive; kudos for that!

It seems such a small thing, but it is deeply connected to the User Experience of your application.

```
    current_date = datetime.now()
    artistdata =[]
    for artist in artists:
        listofshows=Show.query.filter_by(artist_id=artist.id).all()
        num_upcoming_shows = 0
        for show in listofshows:
            if show.start_time > current_date:
                num_upcoming_shows+=1
        artistdata.append({"id":artist.id, "name":artist.name, "num_upcoming_shows":num_upcoming_show
    countofartists=0
    if len(artists):
        countofartists = len(artists)
    response={"count":countofartists, "data":artistdata}
    return render_template('pages/search_artists.html', results=response, search_term=search_term)


@app.route('/artists/<int:artist_id>')
def show_artist(artist_id):
    artist = Artist.query.get(artist_id)
    listofshows=Show.query.filter_by(artist_id=artist_id).all()
    nextshowlist=[]
    prevshowlist=[]
    current_date = datetime.now()
    for show_listitem in listofshows:
        venue = Venue.query.get(show_listitem.venue_id)
        showDictionaryItem ={
            "venue_id": show_listitem.venue_id,
            "venue_name": venue.name,
            "venue_image_link": venue.image_link,
            "start_time": format_datetime(str(show_listitem.start_time))
        }
        if(show_listitem.start_time>current_date):
            nextshowlist.append(showDictionaryItem)
        else:
            prevshowlist.append(showDictionaryItem)
    artistdictionary={
        "id": artist.id,
        "name": artist.name,
        "genres": artist.genres,
        "city": artist.city,
        "state": artist.state,
        "phone": artist.phone,
        "facebook_link": artist.facebook_link,
        "seeking_venue": artist.seeking_venue,
        "seeking_description": artist.seeking_description,
        "website_link": artist.website_link,
        "image_link": artist.image_link,
        "past_shows": prevshowlist,
        "upcoming_shows": nextshowlist,
        "past_shows_count": len(prevshowlist),
        "upcoming_shows_count": len(nextshowlist)
    }
    return render_template('pages/show_artist.html', artist=artistdictionary)

#  Update
```

Rate this review

START

```
#  ----------------------------------------------------------------
@app.route('/artists/<int:artist_id>/edit', methods=['GET'])
def edit_artist(artist_id):
  form = ArtistForm()
  artist = Artist.query.get(artist_id)
  artist_data = {
        "id": artist.id,
        "name": artist.name,
        "genres": artist.genres,
        "city": artist.city,
        "state": artist.state,
        "phone": artist.phone,
        "facebook_link": artist.facebook_link,
        "website_link":artist.website_link,
        "seeking_venue":artist.seeking_venue,
        "seeking_description": artist.seeking_description,
        "image_link": artist.image_link
    }
  return render_template('forms/edit_artist.html', form=form, artist=artist_data)

@app.route('/artists/<int:artist_id>/edit', methods=['POST'])
def edit_artist_submission(artist_id):
  try:
      form = ArtistForm()
      artist = Artist.query.get(artist_id)
      artist.name = form.name.data
      artist.phone = form.phone.data
      artist.state = form.state.data
      artist.city = form.city.data
      artist.genres = form.genres.data
      artist.facebook_link = form.facebook_link.data
      artist.website_link = form.website_link.data
      artist.seeking_venue = form.seeking_venue.data
      artist.seeking_description = form.seeking_description.data
      artist.image_link = form.image_link.data
      db.session.commit()
      flash('Artist ' + request.form['name'] + ' is updated')
  except:
      db.session.rolback()
      flash('update failed for artist ' + request.form['name'] )
  finally:
      db.session.close()
  return redirect(url_for('show_artist', artist_id=artist_id))

@app.route('/venues/<int:venue_id>/edit', methods=['GET'])
def edit_venue(venue_id):
  form = VenueForm()
  venue = Venue.query.get(venue_id)
  venue_data={
    "id": venue.id,
    "name": venue.name,
    "genres": venue.genres,
    "address": venue.address,
    "city": venue.city,
    "state": venue.state,
    "phone": venue.phone,
    "website_link": venue.website_link,
    "facebook_link": venue.facebook_link,
    "seeking_talent": venue.seeking_talent,
    "seeking_description": venue.seeking_description,
    "image_link": venue.image_link
  }
  return render_template('forms/edit_venue.html', form=form, venue=venue_data)

@app.route('/venues/<int:venue_id>/edit', methods=['POST'])
def edit_venue_submission(venue_id):
  name=""
  try:
      form = VenueForm()
      venue = Venue.query.get(venue_id)
      name = form.name.data
      venue.name = form.name.data
      venue.genres = form.genres.data
      venue.city = form.city.data
      venue.state = form.state.data
      venue.address = form.address.data
      venue.phone = form.phone.data
      venue.facebook_link = form.facebook_link.data
      venue.website_link = form.website_link.data
      venue.image_link = form.image_link.data
      venue.seeking_talent = form.seeking_talent.data
      venue.seeking_description = form.seeking_description.data
      db.session.commit()
      flash('Venue ' + name + ' is updated')
  except:
      db.session.rollback()

      flash('update failed for venue ' + name )
  finally:
```

```
      db.session.close()
  return redirect(url_for('show_venue', venue_id=venue_id))

#  Create Artist
#  ----------------------------------------------------------------

@app.route('/artists/create', methods=['GET'])
def create_artist_form():
  form = ArtistForm()
  return render_template('forms/new_artist.html', form=form)

@app.route('/artists/create', methods=['POST'])
def create_artist_submission():
  try:
      form = ArtistForm()
      artist = Artist(
                    name=form.name.data,
                    city=form.city.data,
                    state=form.city.data,
                    phone=form.phone.data,
                    genres=form.genres.data,
                    image_link=form.image_link.data,
                    seeking_venue = form.seeking_venue.data,
                    seeking_description=form.seeking_description.data,
                    website_link= form.website_link.data,
                    facebook_link=form.facebook_link.data
                    )
      db.session.add(artist)
      db.session.commit()
      # on successful db insert, flash success
      flash('Artist ' + request.form['name'] + ' was successfully listed!')
  except:
    db.session.rollback()
    flash('An error occurred. Artist ' + request.form['name'] + ' could not be listed.')
  finally:
    db.session.close()
  return render_template('pages/home.html')


#  Shows
#  ----------------------------------------------------------------

@app.route('/shows')
def shows():
    listofshows = []
    shows = Show.query.order_by(db.desc(Show.start_time))
    for show in shows:
      listofshows.append({
        "venue_id": show.venue_id,
        "venue_name": show.venue.name,
        "artist_id": show.artist_id,
        "artist_name": show.artist.name,
        "artist_image_link": show.artist.image_link,
        "start_time": format_datetime(str(show.start_time))
    })
    return render_template('pages/shows.html', shows=listofshows)

@app.route('/shows/create')
def create_shows():
  # renders form. do not touch.
  form = ShowForm()
  return render_template('forms/new_show.html', form=form)

@app.route('/shows/create', methods=['POST'])
def create_show_submission():
    try:
        show = Show(artist_id=request.form['artist_id'], venue_id=request.form['venue_id'],start_time
        db.session.add(show)
        db.session.commit()
        flash('Show was successfully listed!')
    except:
        db.session.rollback()
        flash('An error occurred. Show could not be listed.')
    finally:
        db.session.close()
    return render_template('pages/home.html')

@app.errorhandler(404)
def not_found_error(error):
    return render_template('errors/404.html'), 404

@app.errorhandler(500)
def server_error(error):
    return render_template('errors/500.html'), 500


if not app.debug:
    file_handler = FileHandler('error.log')
```

```
    file_handler.setFormatter(
        Formatter('%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]')
    )
    app.logger.setLevel(logging.INFO)
    file_handler.setLevel(logging.INFO)
    app.logger.addHandler(file_handler)
    app.logger.info('errors')

#---------------------------------------------------------------------------#
# Launch.
#---------------------------------------------------------------------------#

# Default port:
if __name__ == '__main__':
    app.run()

# Or specify port manually:
'''
if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    app.run(host='0.0.0.0', port=port)
'''
```

▸ Models.py 2

▸ config.py 1

▸ fabfile.py

▸ forms.py

▸ migrations/env.py

▸ migrations/README

▸ migrations/versions/092c81e0ae06_initial_database_creation.py

▸ migrations/versions/c7d504feae69_added_all_models.py

▸ migrations/versions/dc594c07605c_added_website_to_artists.py

▸ migrations/versions/e4a166de4cb9_added_fields_to_artist.py

▸ README.md

▸ requirements.txt

▸ static/css/bootstrap.css

▸ static/css/bootstrap.min.css

▸ static/css/bootstrap.min.js

▸ static/css/bootstrap-theme.css

▸ static/css/bootstrap-theme.min.css

▸ static/css/layout.forms.css

▸ static/css/layout.main.css

- static/css/main.css
- static/css/main.quickfix.css
- static/css/main.responsive.css
- static/js/libs/bootstrap-3.1.1.min.js
- static/js/libs/jquery-1.11.1.min.js
- static/js/libs/modernizr-2.8.2.min.js
- static/js/libs/moment.min.js
- static/js/libs/respond-1.4.2.min.js
- static/js/plugins.js
- static/js/script.js
- templates/errors/404.html
- templates/errors/500.html
- templates/forms/edit_artist.html
- templates/forms/edit_venue.html
- templates/forms/new_artist.html
- templates/forms/new_show.html
- templates/forms/new_venue.html
- templates/layouts/form.html
- templates/layouts/main.html
- templates/pages/artists.html
- templates/pages/home.css
- templates/pages/home.html
- templates/pages/search_artists.html
- templates/pages/search_venues.html
- templates/pages/show.html
- templates/pages/show_artist.html
- templates/pages/show_venue.html
- templates/pages/shows.html

templates/pages/venues.html

RETURN TO PATH

Rate this review

START