```python
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:///' + os.path.join(basedir, 'app.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    POSTS_PER_PAGE = 30

```

```python
from app import app, socketio


if __name__ == '__main__':
    socketio.run(app, debug=True)

```

```python
from app import models, db
from sqlalchemy import *
from datetime import datetime

def generate_random_postings():
    result = models.Posting.query.join(models.User).with_entities(
        models.Posting.postid, models.Posting.userid, models.User.username,
        models.Posting.title, models.Posting.price, models.Posting.description,
        models.User.rating)
    return result

def add_new_post(form_input, current_user):
    db.session.add(models.Posting(
        userid = current_user.userid,
        date = datetime.now(),
        title = form_input['title'],
        description = form_input['description'],
        price = form_input['price'],
        category = form_input['category'],
        contactmethod = form_input['contactmethod'],
        tags = form_input['tags']))
    db.session.commit()
    return True

def add_user(new_user_info):
    try:
        newUser = models.User(
            username        = new_user_info['username'],
            email           = new_user_info['email'],
            password        = new_user_info['password'],
            phonenumber     = new_user_info['phone'],
            personalemail   = new_user_info['personalemail'],
            bio             = new_user_info['bio'],
            rating          = float(new_user_info['rating']),
            numRatings      = int(new_user_info['numRatings']),
        )
        db.session.add(newUser)
        db.session.commit()
        db.session.refresh(newUser)
        if newUser.userid is None:
            return False
        return True
    except Exception as e:
        return False
    return False

def get_post_id(postid):
    try:
        if postid is None and not int(postid) > 0:
            return None
    except Exception as e:
        return None
    current_post_info = models.Posting.query.filter_by(postid=postid).join(
        models.User).with_entities(
        models.Posting.postid, models.Posting.title, models.Posting.category,
        models.Posting.price, models.Posting.description,
        models.Posting.contactmethod, models.Posting.tags,
        models.User.phonenumber, models.User.email, models.User.userid,
        models.User.username, models.User.rating, models.User.personalemail
    ).first()
    return current_post_info

def modify_post_by_id(results, postid):
    models.Posting.query.filter_by(postid=postid).update(dict(results))
    db.session.commit()

def update_current_user(current_user, result):
    if 'password' in result:
        current_user.password = result['password']
    current_user.phonenumber = result['phonenumber']
    current_user.personalemail = result['personalemail']
    current_user.bio = result['bio']
    db.session.commit()
    return True

def get_posting_by_id(postid):
    posting_info = models.Posting.query.filter_by(postid=postid).first()
    if posting_info is None:
        return None, None
    if posting_info.contactmethod == 'personalemail':
        poster_info =
        models.User.query.filter_by(userid=posting_info.userid).with_entities(
            models.User.personalemail, models.User.rating, models.User.userid,
            models.User.username
        ).first()
    elif posting_info.contactmethod == "phonenumber":
        poster_info =
        models.User.query.filter_by(userid=posting_info.userid).with_entities(
            models.User.phonenumber, models.User.rating, models.User.userid,
            models.User.username
        ).first()
    else:
        poster_info =
        models.User.query.filter_by(userid=posting_info.userid).with_entities(
            models.User.email, models.User.rating, models.User.userid,
            models.User.username
        ).first()
    return posting_info, poster_info

def get_user_by_email(email):
    return models.User.query.filter_by(email=email).first()

def add_claim(form, postid, user):
    try:
        post_info, poster_info = get_posting_by_id(postid)
        if user.userid == post_info.userid and not form['buyeremail'] == False:
            buyer_info = get_user_by_email(form['buyeremail'])
            newClaim = models.Claim(
                postid          = postid,
                sellerid        = user.userid,
                buyerid         = buyer_info.userid,
                usersubmitted   = user.userid,
                date            = datetime.now(),
                Rating          = form['rating']
            )
            db.session.add(newClaim)
            return True, newClaim
        elif not user.userid == post_info.userid:
            newClaim = models.Claim(
                postid          = postid,
                sellerid        = post_info.userid,
                buyerid         = user.userid,
                usersubmitted   = user.userid,
                date            = datetime.now(),
                Rating          = form['rating']
            )
            db.session.add(newClaim)
            return True, newClaim
    except Exception as e:
        db.session.rollback()
        return False, False
    return False, False

def check_for_transaction(claim):
    try:
        other_claim = models.Claim.query.filter(
            models.Claim.postid==claim.postid,
            models.Claim.sellerid==claim.sellerid,
            models.Claim.buyerid==claim.buyerid,
            models.Claim.usersubmitted!=claim.usersubmitted
        ).first()
        if other_claim is not None:
            newTransaction = models.Transaction(
                date            = datetime.now(),
                claimidseller   = claim.sellerid,
                claimidbuyer    = claim.buyerid
            )
            db.session.add(newTransaction)
            print("Try to Alter Ratings!")
            alter_ratings(claim, other_claim)
            print("Try to Archive!")
            if archive_posting(claim.postid, newTransaction):
                print("Archived! Now Delete The Claims:")
                delete_claim(claim.claimid)
                delete_claim(other_claim.claimid)
                print("Finished")
                return True
            else:
                raise ValueError
    except Exception as e:
        print("Rollback in check_for_transaction")
        db.session.rollback()
        return None
    return False

def delete_claim(claimid):
    try:
        models.Claim.query.filter_by(claimid=claimid).delete()
    except Exception as e:
        pass

def delete_user(userid):
    try:
        someUser = User.query.filter_by(userid=someUserID).first()
        db.session.delete(someUser)
        db.session.commit()
    except Exception as e:
        pass


def archive_posting(postid, transaction=None):
    try:
        if postid is not None:
            post = models.Posting.query.filter_by(postid=postid).first()
            print("got Post")
            archivedPost = models.ArchivedPosting(
                transactionid   = transaction.transactionid,
                postid      = post.postid,
                buyerid         = transaction.claimidbuyer,
                sellerid        = transaction.claimidseller,
                date      = datetime.now(),
                title       = post.title,
                description    = post.description,
                price       = post.price,
                category    = post.category,
                contactmethod   = post.contactmethod,
                tags      = post.tags
            )
        else:
            archivedPost = models.ArchivedPosting(
                postid      = post.postid,
                date      = datetime.now(),
                title       = post.title,
                description   = post.description,
                price       = post.price,
                category    = post.category,
                contactmethod   = post.contactmethod,
                tags      = post.tags
            )

        db.session.add(archivedPost)
        db.session.delete(post)
        return True
    except Exception as e:
        db.session.rollback()
    return False

def get_new_rating(current_rating, current_number, new_rating):
    current_number += 1
    current_rating = current_rating * (current_number-1)/current_number + new_rating * 1/current_number
    return [current_rating, current_number]

def alter_ratings(claim, other_claim):
    print("Getting Ratings")
    user1 = models.User.query.filter_by(userid=claim.usersubmitted).first()
    user2 = models.User.query.filter_by(userid=other_claim.usersubmitted).first()
    print("Update the Ratings Manually")
    [user1.rating, user1.numRatings] = get_new_rating(user1.rating, user1.numRatings, other_claim.Rating)
    [user2.rating, user2.numRatings] = get_new_rating(user2.rating, user2.numRatings, claim.Rating)
    print("FINISHED alter ratings")

def get_postings(userid):
    try:
        postings = models.Posting.query.filter_by(userid=userid).all()
        print("Got postings for user")
        return postings
    except Exception as e:
        pass
    return None

def get_claims(userid):
    try:
        claims = models.Claim.query.filter_by(usersubmitted=userid).join(models.Posting).with_entities(
            models.Claim.date, models.Posting.title, models.Posting.postid
        ).all()
        print("Got Claims")
        return claims
    except Exception as e:
        pass
    return None

def get_sales(userid):
    try:
        sales = models.ArchivedPosting.query.filter_by(sellerid=userid).with_entities(
            models.ArchivedPosting.title, models.ArchivedPosting.buyerid, models.ArchivedPosting.price,
            ).join(
            models.Transaction).join(models.User, models.User.userid == models.ArchivedPosting.buyerid).with_entities(
                    models.Transaction.date, models.ArchivedPosting.title, models.ArchivedPosting.price,
                    models.User.username, models.User.userid).all()
        print("Got sales")
        return sales
    except Exception as e:
        pass
    return None

def get_purchases(userid):
    try:
        purchases = models.ArchivedPosting.query.filter_by(buyerid=userid).with_entities(
            models.ArchivedPosting.title, models.ArchivedPosting.sellerid,
            models.ArchivedPosting.price,
            ).join(
            models.Transaction).join(models.User, models.User.userid ==
            models.ArchivedPosting.sellerid).with_entities(
                    models.Transaction.date, models.ArchivedPosting.title,
                    models.ArchivedPosting.price,
                    models.User.username, models.User.userid).all()
        print("Got purchases")
        return purchases
    except Exception as e:
        pass
    return None
```

```python
import re
from werkzeug.security import check_password_hash, generate_password_hash

# Returns email if in valid format; else returns false
def get_email(email_field):
    regex = '^\w+([\.-]?\w+ )*@(\w+.)*pitt.edu'
    old_regex = '\w+@(\w+.)*pitt.edu'
    try:
        if re.search(regex, str(email_field)) is not None:
            return email_field
        raise ValueError
    except Exception as e:
        return False

# verify password given in form; true for correct password, otherwise false
# user: takes USER class from model;
#       for TESTING object with value string of HASHED password
# password: string of UNhashed password passed in by the user on login
def verify_password(user, password):
    try:
        if user is None or not check_password_hash(user.password, str(password)):
            return False
        return True
    except Exception as e:
        raise



##########################################################################
# Home View Search and filter methods

def get_category(field):
    try:
        if str(field) == "All":
            return 'All'
        return str(field)
    except Exception as e:
        return 'All'

#               MAY WANT TO EDIT THIS
# takes inputted search textbox input for backend search
def get_search_text(field):
    try:
        if str(field).strip() == '':
            return ''
        return field
    except Exception as e:
        return ''

def get_search_elems(search):
    try:
        elems = search.split()
        return_search = []
        for e in elems:
            return_search.append('%' + e + '%')
        return return_search
    except Exception as e:
        pass
    return ''

def get_page_number(field, number_postings, posts_per_page):
    try:
        field = int(field)
        if field > 0 and field <= (int(number_postings/posts_per_page)+1):
            return field
    except Exception as e:
        pass
    return 1

def get_max_price(field, min_field):
    try:
        field = float(field)
        min_field = float(min_field)
        if field > 0 and field > min_field and field < 2000:
            return str(field)
        else:
            return '2000'
    except Exception as e:
        return '2000'

def get_min_price(field):
    try:
        field = float(field)
        if field > 0 and field <= 2000:
            return str(field)
        elif field > 2000:
            return '2000'
        else:
            return '0'
    except Exception as e:
        return '0'

def should_randomize(submitted):
    return submitted == {'minPrice': '0', 'maxPrice': '0', 'search': '', 'category':
    '', 'page': 1}

def get_page(field):
    try:
        return int(field)
    except Exception as e:
        pass
    return 1

def get_filters(forms, get_recieved):
    if get_recieved and forms.get('minPrice') is not None:
        submitted = {}
        submitted['minPrice'] = get_min_price(forms['minPrice'])
        submitted['maxPrice'] = get_max_price(forms['maxPrice'], submitted['minPrice'])
        submitted['search'] = get_search_text(forms['search'])
        submitted['category'] = get_category(forms['category'])
        submitted['page']       = get_page(forms['page']) if 'page' in forms else 1
        return [submitted, should_randomize(submitted)]
    else:
        return [{'minPrice': '0', 'maxPrice': '0', 'search': '', 'category': '',
        'page': 1}, True]


############################## New Posting Submission ####################
# verifies it is 30 chars or shorter
def validate_title(field):
    try:
        field = str(field)
        if len(field) > 0 and len(field) <31:
            return [True, field]
    except Exception as e:
        return [False, "The title needs to be 1-30 characters long."]
    return [False, "The title needs to be 1-30 characters long. Your input was " +
    str(len(field)) + " characters."]

# this should never really fail. It is on us if it does
def validate_category(field, CATEGORIES):
    try:
        field = str(field)
        if field in CATEGORIES:
            return [True, field]
    except Exception as e:
        return [False, "Category was not specified. Try resubmitting!"]
    return [False, "Invalid preffered contact method. Try resubmitting."]

# validates that price is in range of 0-2000; cuts of after 2nd decimal place
def validate_price(field):
    try:
        field = round(float(field), 2)
        if field > 0 and field <= 2000:
            return [True, field]
        raise ValueError
    except ValueError as e:
        return [False, "Invalid price. Needs to be in the range of 0 to 2,000
        inclusive."]
    return [False, "Invalid preffered contact method. Try resubmitting."]

# ensure description is less than 1001 chars long.
def validate_desc(field):
    try:
        field = str(field)
        if len(field) > 1000:
            return [False, "Your short description cannot be longer than 1,000
            characters long."]
    except Exception as e:
        return [False, "Invalid short description."]
    return [True, field]

# returns preferred contact value; should alk
def validate_preferred_contact(field):
    try:
        field = str(field)
        if field == "email" or field == "phonenumber" or field == "personalemail":
            return [True, field]
    except Exception as e:
        return [False, "Invalid preffered contact method. Try resubmitting."]
    return [False, "Invalid preffered contact method. Try resubmitting."]

def validate_preferred_tags(field, title):
    try:
        title = str(title[1]).split()
        field = str(field) + ',' + ','.join(title)
        field = ''.join(field.split()).lower().split(',')
        if len(field) < 900:
            if len(field) > 50:
                return [False, "Too many tags. The maximum tags are 50. Maximum
                character limit is 900"]
            field = list(set(field))
            field = ','.join(field)
            return [True, field]
        raise ValueError
    except Exception as e:
        return [False, "Too many tags. Maximum character limit is 900"]

def validate_input(forms, CATEGORIES):
    result = {}
    result['title'] = validate_title(forms['title'])
    result['category'] = validate_category(forms['category'], CATEGORIES)
    result['price'] = validate_price(forms['price'])
    result['description'] = validate_desc(forms['description'])
    result['contactmethod'] = validate_preferred_contact(forms['preferredContact'])
    result['tags'] = validate_preferred_tags(forms['tags'], result['title'])
    return result

def generate_return_values(given):
    error = []
    good_results = {}
    for key, elem in given.items():
        if elem[0]:
            good_results[key] = elem[1]
        else:
            error.append(str(elem[1]))
    return good_results, error


def get_form_create_post(forms, CATEGORIES):
    initial = validate_input(forms, CATEGORIES)
    return generate_return_values(initial)

########################################################################
# The CREATE ACCOUNT Functions
def generate_new_account_form(forms):
    results = generate_fields_create_account(forms)
    return generate_return_values(results)


def get_username(email):
    try:
        username = str(email).split('@')
        if len(username) == 2 and username[0] != '':
            return [True, username[0]]
        raise ValueError
    except Exception as e:
        return [False, 'An unexpected error has occurred.']


def generate_fields_create_account(forms):
    new_account_info = {}
    new_account_info['email']            = validate_email(forms['email'])
    new_account_info['username']         = get_username(new_account_info['email'][1])
    new_account_info['password']         = validate_password(forms['password'],
    forms['password2'])
    new_account_info['phone']            = validate_phone_number(forms['phonenumber'])
    new_account_info['personalemail']    = validate_personal_email(forms['personalemail'])
    new_account_info['bio']              = validate_bio(forms['bio'])
    new_account_info['rating']           = [True, '5']
    new_account_info['numRatings']       = [True, '0']
    return new_account_info

def validate_email(field):
    try:
        field = get_email(str(field))
        if not field == False and len(field) < 75:
            return [True, field]
        raise ValueError
    except Exception as e:
        return [False, "The university email must end with a school domain (pitt.edu)."]

def validate_password(password1, password2):
    try:
        password1 = str(password1)
        password2 = str(password2)
        if password1 == password2:
            if len(password1) > 7 and len(password1) < 33:
                return [True, generate_password_hash(password1)]
        raise ValueError
    except Exception as e:
        return [False, "Both Password fields must match and have between 8 and 32
        characters (inclusive)."]

def convert_number(phone):
    phone = phone.replace('-', '')
    phone = phone.replace('(', '')
    phone = phone.replace(')', '')
    if len(phone) == 10:
        phone = '1' + phone
    elif len(phone) == 0:
        return ''
    elif not len(phone) == 11 or not phone[0] or len(phone)  == '1':
        raise ValueError
    return int(phone)

def convert_again_number(phone):
    converted = phone[0] + "(" + phone[1:4] + ")" + phone[4:7] + "-" + phone[7:10]
    return converted

### ADD MORE HERE ###
def validate_phone_number(field):
    try:
        phone_number = str(field)
        phone_number = convert_number(phone_number)
        phone_number = convert_again_number(str(phone_number)) if phone_number != ''
        else ''
        return [True, phone_number]
    except Exception as e:
        return [False, "Phone number must be 10 characters long or 11 characters long
        with the country code being '1' in order to be processed."]
    return [True, phone_number]


def validate_personal_email(field):
    email_regex = '^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$'
    try:
        field = str(field)
        if re.search(email_regex, field) is not None:
            return [True, field]
    except Exception as e:
        return [False, "The personal email address is not a legal value."]
    return [True, field]

def validate_bio(field):
    try:
        field = str(field)
        if len(field) <  251:
            return [True, field]
    except Exception as e:
        return [False, "The biography is unable to be processed. Possibly an invalid
        symbol."]
    return [False, "Length exceeds 250 characters"]

def get_modified_account_info(forms, userid):
    result = generate_fields_edit_account(forms, userid)
    return generate_return_values(result)

def validate_password_simple(password):
    try:
        password = str(password)
        return [True, password]
    except Exception as e:
        return [False, "Try Resubmitting information."]

def validate_delete(forms):
    try:
        if forms['deleteaccount'] == 'delete':
            return [True, 'delete']
    except Exception as e:
        pass
    return [False, "nothing"]

def generate_fields_edit_account(forms, userid):
    new_account_info = {}
    new_account_info['userid']          = [True, str(userid)]
    if forms['newpassword'] and forms['newpassword'] != '':
        new_account_info['password']    = validate_password(forms['newpassword'],
        forms['newpassword2'])
    new_account_info['oldpassword']     = validate_password_simple(forms['oldpassword'])
    new_account_info['phonenumber']     = validate_phone_number(forms['phonenumber'])
    new_account_info['personalemail']   = validate_personal_email(forms['personalemail'])
    new_account_info['bio']             = validate_bio(forms['bio'])
    new_account_info['deleteaccount']   = validate_delete(forms)
    return new_account_info

################################################################################
# Claims
def get_new_claims_form(forms, current_user, postid):
    results = generate_claims_forms(forms, current_user, postid)
    return generate_return_values(results)

def generate_claims_forms(forms, current_user, postid):
    claim_info = {}
    claim_info['postid']        = [True, postid]
    claim_info['userid']        = [True, current_user.userid]
    claim_info['rating']        = validate_rating_claims(forms['rating'])
    claim_info['buyeremail']    = validate_buyer_email(forms)
    return claim_info

def validate_rating_claims(field):
    try:
        field = int(field)
        if field > 0 and field < 6:
            return [True, field]
        raise ValueError
    except Exception as e:
        return [False, "Please resubmit your claim!"]

def validate_buyer_email(forms):
    if 'buyeremail' in forms:
        try:
            field = str(forms['buyeremail'])
            if get_email(field) != False:
                return [True, get_email(field)]
        except Exception as e:
            pass
    return [True, False]
```

```python
from flask import Flask, session
from config import Config
from flask_sqlalchemy import SQLAlchemy
from flask_socketio import SocketIO, emit


print("APP NAME: " + str(__name__))
app = Flask(__name__)
app.config.from_object(Config)
db = SQLAlchemy(app)
db.init_app(app)
socketio = SocketIO(app)


from app import routes, models

from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import relationship
from app import app, db
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
from app.form_submissions import get_username, validate_phone_number
import csv


# INITS THE DBs for USERs
def add_postings(file):
    list = []
    key_list = ['userid', 'title', 'description', 'price', 'category', 'contactmethod']
    with open(file, 'r') as csv_file:
        reader = csv.DictReader(csv_file, delimiter=',', fieldnames=key_list)
        for row in reader:
            new_item = {}
            for key in key_list:
                new_item[key] = row[key]
            list.append(new_item)
    for value in list:
        tags = value['title'].split()
        tags = ','.join(tags)
        newPosting = Posting(
            userid          = value['userid'],
            date            = datetime.now(),
            title       = value['title'],
            description    = value['description'],
            price       = value['price'],
            category    = value['category'],
            contactmethod   = value['contactmethod'],
            tags            = tags
        )
        db.session.add(newPosting)
        db.session.commit()


def add_users(file):
    list = []
    key_list = ['phonenumber', 'email', 'personalemail', 'password', 'bio']
    with open(file, 'r') as csv_file:
        reader = csv.DictReader(csv_file, delimiter=',', fieldnames=key_list)
        for row in reader:
            new_item = {}
            for key in key_list:
                new_item[key] = row[key]
            new_item['phonenumber'] = validate_phone_number(new_item['phonenumber'])[1]
            new_item['username'] = get_username(new_item['email'])
            list.append(new_item)
    for value in list:
        newUser = User(
            username             = value['username'][1],
            email                = value['email'],
            personalemail        = value['personalemail'],
            password             = generate_password_hash(value['password']),
            phonenumber          = value['phonenumber'],
            bio                  = value['bio'],
            rating               = 5,
            numRatings           = 0
        )
        db.session.add(newUser)
        db.session.commit()


@app.cli.command('initdb')
def initdb_command():
    # wipeout
    db.drop_all()
    db.create_all()

    add_users("sampleUser.csv")
    # add some default data
    # db.session.add(User(username='jmd230', email="jmd230@pitt.edu", password=generate_password_hash('pass'), phonenumber='4121234567', personalemail='jordanmdeller@gmail.com', bio='Serious offers only', rating=2.51, numRatings=10))
    # db.session.add(User(username='admin', email="admin@pitt.edu", password=generate_password_hash('foobiz'), phonenumber='2341172381', personalemail='admin@admin.com', bio='I am an admin. This account is used to manage and test out the APP!', rating=5, numRatings=1))
    # db.session.add(User(username='tester1', email="tester1@pitt.edu", password=generate_password_hash('foobar'), phonenumber='2456734224', personalemail='tester1@gmail.com', bio='Tester is testing account for testing...', rating=3, numRatings=10))

    add_postings("postingsData.csv")

    db.session.add(Posting(userid=1, date=datetime.now(), title='Cool Book', description='Very good quality, barely used.', price=50.00, category='Textbooks', contactmethod='email', tags='book'))
    db.session.add(Posting(userid=2, date=datetime.now(), title='Brown couch', description='No signs of wear.', price=100.00, category='Furniture', contactmethod='phonenumber', tags='furniture, couch, seating, brown, comfy'))
    db.session.add(Posting(userid=2, date=datetime.now(), title='Cheap Book', description='Great quality.', price=20.00, category='Textbooks', contactmethod='personalemail', tags='book'))

    db.session.commit()

    print('Initialized the database.')


class User(db.Model):
    userid      = db.Column(db.Integer, primary_key = True)
    username      = db.Column(db.String(24), nullable = False)
    email       = db.Column(db.String(80), unique=True, nullable = False)
    # hashed password is ~100 chars ALWAYS
    password    = db.Column(db.String(128), nullable = False)
    phonenumber   = db.Column(db.String(64), nullable = False)
    personalemail   = db.Column(db.String(80), nullable = False)
    bio         = db.Column(db.String(250), nullable = False)
    rating      = db.Column(db.Float(2), nullable = False)
    numRatings    = db.Column(db.Integer, nullable = False)
    postings        = relationship("Posting", cascade="all,delete", backref="User")

    def __repr__(self):
        return '<User {}>'.format(self.username)


class Posting(db.Model):
    postid      = db.Column(db.Integer, primary_key = True)
    userid      = db.Column(db.Integer, db.ForeignKey("user.userid"))
    date      = db.Column(db.Date, nullable = False)
    title       = db.Column(db.String(30), nullable = False)
    description   = db.Column(db.String(250), nullable = False)
    price       = db.Column(db.Integer, nullable = False)
    category    = db.Column(db.String(80), nullable = False)
    contactmethod   = db.Column(db.String(80), nullable = True)
    tags      = db.Column(db.String(1000), nullable = True)
    claims        = relationship("Claim", cascade="all,delete", backref="Posting")

    def __repr__(self):
        return '<Posting {}: "{}">'.format(self.postid, self.title)

```

```python

class Claim(db.Model):
    __table_args__ = (
        db.UniqueConstraint('postid', 'sellerid', 'buyerid', 'usersubmitted', name='unique_claim_buyer_seller'),
    )
    claimid      = db.Column(db.Integer, primary_key = True)
    postid       = db.Column(db.Integer, db.ForeignKey("posting.postid"))
    sellerid     = db.Column(db.Integer, db.ForeignKey("user.userid"))
    buyerid      = db.Column(db.Integer, db.ForeignKey("user.userid"))
    usersubmitted = db.Column(db.Integer, db.ForeignKey("user.userid"))
    date      = db.Column(db.Date, nullable = False)
    Rating          = db.Column(db.Integer, nullable = False)

    def __repr__(self):
        return '<Claim {}: "{}">'.format(self.claimid)


class Transaction(db.Model):
    transactionid = db.Column(db.Integer, primary_key = True)
    date            = db.Column(db.Date, nullable = False)
    claimidseller = db.Column(db.Integer, db.ForeignKey("claim.claimid"), nullable = False)
    claimidbuyer  = db.Column(db.Integer, db.ForeignKey("claim.claimid"), nullable = False)

    def __repr__(self):
        return '<Transaction {}: "{}">'.format(self.transactionid)


class ArchivedPosting(db.Model):
    __table_args__ = (
        db.UniqueConstraint('postid', 'buyerid', 'archivedpostid', 'sellerid', name='unique_archive_posting_constraint'),
    )
    archivedpostid  = db.Column(db.Integer, primary_key = True)
    transactionid   = db.Column(db.Integer, db.ForeignKey('transaction.transactionid'), nullable = True)
    postid      = db.Column(db.Integer, nullable = False)
    buyerid         = db.Column(db.Integer, db.ForeignKey("user.userid"), nullable = True)
    sellerid        = db.Column(db.Integer, db.ForeignKey("user.userid"), nullable = True)
    date      = db.Column(db.Date, nullable = False)
    title       = db.Column(db.String(80), nullable = False)
    description   = db.Column(db.String(250), nullable = True)
    price       = db.Column(db.Integer, nullable = False)
    category    = db.Column(db.String(80), nullable = False)
    contactmethod   = db.Column(db.String(80), nullable = True)
    tags      = db.Column(db.String(1000), nullable = True)

    def __repr__(self):
        return '<Posting {}: "{}">'.format(self.postid, self.title)

```

```python
from app import app, helper_functions, socketio, form_submissions, database_helpers
from app.models import *
from werkzeug.security import generate_password_hash, check_password_hash
from flask import redirect, render_template, request, session, url_for, abort, g, flash
from random import shuffle
from datetime import datetime
from sqlalchemy import and_, or_

CATEGORIES = ['All', 'Textbooks', 'Furniture', 'Food', 'Events', 'Software',
'Electronics',
 'Beauty and Personal Care', 'Clothes', 'School Supplies', 'Appliances']
CONTACT_METHOD = {
    "email": "Email (university provided)",
    "phonenumber": "Phone Number (if provided)",
    "personalemail": "Personal Email (if provided)"
}

# forces logout on browser close(); aka senses packets have stopped flowing
@socketio.on('disconnect')
def disconnect_user():
    logout()

########################################
# ROUTES START HERE
########################################
# Log the user out
@app.route('/logout')
def logout():
    session.pop('userid', None)
    return redirect(url_for('login'))

# Run at the beginning of each request before functions run to check if logged in
@app.before_request
def request_authentication():
    g.user = None
    if 'userid' in session:
        g.user = User.query.filter_by(userid=session['userid']).first()

### ERROR HANDLING PAGES
@app.route('/error')
def not_found_error_item():
    return render_template('404.html'), 404

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

@app.errorhandler(500)
def internal_error(error):
    db.session.rollback()
    return render_template('500.html'), 500

@app.route('/')
def slash_redirect():
    return redirect(url_for('login'))
########################## LOGIN ###########################################
# The login screen
@app.route('/login', methods=['GET', 'POST'])
def login(error=""):
    title = "Login to Craigversity!"
    LOGIN_ERROR = "Invalid information was submitted. Please try again!"
    error = ''
    if g.user:
        return redirect(url_for('user_home_screen'))
    if request.method == "POST":
        error = LOGIN_ERROR
        email = form_submissions.get_email(request.form['email'])
        if not email == False:
            user = User.query.filter_by(email=email).first()
            if form_submissions.verify_password(user, request.form['password']):
                session['userid'] = user.userid
                return redirect(url_for('user_home_screen'))
    return render_template('login.html', current_user_is_auth=False, error=error,
    page_title=title, css_file=helper_functions.generate_linked_files('login'))

############################ ACCOUNT ROUTES ###############################
# The create account screen
@app.route('/create-account', methods=['GET', 'POST'])
def create_account(error=""):
    title = "Welcome to Craigversity!"
    CREATE_ERROR = "Need to fill in ALL fields marked with an '*'"
    errors = []
    if g.user:
        return redirect(url_for('user_home_screen'))
    if request.method == 'POST':
        [results, errors] = form_submissions.generate_new_account_form(request.form)
        if len(errors) == 0:
            added_successfully = database_helpers.add_user(results)
            if added_successfully:
                return redirect(url_for('login'))
            errors = "Could not process. Try again."
    return render_template('create-account.html', current_user_is_auth=False,
    error=errors, page_title=title,
    css_file=helper_functions.generate_linked_files('create-account'), )

# The user screen
@app.route('/user', methods=['GET', 'POST'])
def users_account():
    if g.user is None:
        return redirect(url_for('login'))
    account_info = g.user
    title = "USER: " + g.user.username
    if request.method == "GET":
        userid = (request.args.get('userid'))
        account_info = User.query.filter_by(userid=userid).first()
        if account_info is None:
            return redirect(url_for('not_found_error_item'))
        postings    = database_helpers.get_postings(account_info.userid)
        claims      = database_helpers.get_claims(account_info.userid)
        sales       = database_helpers.get_sales(account_info.userid)
        purchases   = database_helpers.get_purchases(account_info.userid)
        title = "USER: " + account_info.username
    return render_template('account-view.html', purchases=purchases, postings=postings,
    sales=sales, claims=claims, current_user_id=g.user.userid,
    current_user_is_auth=(g.user.userid > 0),  user_id=g.user.userid,
    CURRENT_USER_ID=g.user.userid, page_title=title,
    css_file=helper_functions.generate_linked_files('account-view'),
    account=account_info)

# The edit account screen
@app.route('/edit-account', methods=['GET', 'POST'])
def edit_account(error=""):
    if g.user is None:
        return redirect(url_for('login'))
    title = 'Edit Account'
    current_user = g.user
    error = ''
    if request.method == 'POST':
        [result, error] = form_submissions.get_modified_account_info(request.form,
        g.user.userid)
        if len(error) == 0 and check_password_hash(g.user.password,
        str(result['oldpassword'])):
            if result['deleteaccount'] == "delete":
                database_helpers.delete_user(g.user.userid)
```

```python
            if database_helpers.update_current_user(g.user, result):
                return redirect(url_for('login'))
    return render_template('edit-account.html', current_user_id=g.user.userid,
    current_user_is_auth=(g.user.userid > 0),  error=error, current_user=current_user,
    CURRENT_USER_ID=g.user.userid, page_title=title,
    css_file=helper_functions.generate_linked_files('create-account'), )

################################################################
###################### POSTINGS ##################################
# The new posting submission screen
@app.route('/new-posting-submission', methods=['GET', 'POST'])
def new_posting_submission(error=""):
    title = "Submit a New Posting!"
    error = []
    if g.user is None:
        return redirect(url_for('login'))
    if request.method == 'POST':
        [results, error] = form_submissions.get_form_create_post(request.form,
        CATEGORIES)
        if len(error) == 0:
            database_helpers.add_new_post(results, g.user)
            return redirect(url_for('login'))
    return render_template('create-posting-view.html', contact_options=CONTACT_METHOD,
    categories=CATEGORIES, current_user_id=g.user.userid, js_file="tag-javascript.js",
    current_user_is_auth=(g.user.userid > 0), page_title=title, error=error,
    css_file=helper_functions.generate_linked_files('create-posting-view'))

# NEED TO REWORK CONTACT METHOD!!!
@app.route('/edit-posting', methods=['GET', 'POST'])
def edit_posting(error=""):
    title = 'Edit Posting'
    if g.user is None:
        return redirect(url_for('login'))

    posting_info = database_helpers.get_post_id(request.args.get('postid'))
    if posting_info is None or g.user.userid != posting_info.userid:
        return redirect(url_for('user_home_screen'))

    if request.method == 'POST':
        [results, error] = form_submissions.get_form_create_post(request.form,
        CATEGORIES)
        if len(error) == 0:
            print(results)
            database_helpers.modify_post_by_id(results, posting_info[0])
            return redirect(url_for('user_home_screen'))
    return render_template('edit-posting-view.html',contact_options=CONTACT_METHOD,
    categories=CATEGORIES, js_file="tag-javascript.js", current_user_id=g.user.userid,
    current_user_is_auth=(g.user.userid > 0),  user_id=g.user.userid,
    CURRENT_USER_ID=g.user.userid, page_title=title, error=error,
    css_file=helper_functions.generate_linked_files('create-posting-view'),
    post=posting_info)

# The posting screen
@app.route('/posting', methods=['GET'])
def full_posting_view():
    if g.user is None:
        return redirect(url_for('login'))
    #posting_info = {}
    if request.method == 'GET':
        [posting_info, user_info] =
        database_helpers.get_posting_by_id(request.args.get('postid'))
        if posting_info is None:
            return redirect(url_for('not_found_error_item'))
        title = "POSTING: " + posting_info.title
    return render_template('full-posting-view.html',current_user_id=g.user.userid,
    current_user_is_auth=(g.user.userid > 0), page_title=title,
    css_file=helper_functions.generate_linked_files('full-posting-view'),
    post=posting_info, poster_info=user_info)

###################################################################
# The home user logged in screen that lists postings
@app.route('/search-and-filter-postings', methods=['GET'])
def user_home_screen():
    title = "Search and Filter Postings!"
    page = 1
    if g.user is None:
        return redirect(url_for('login'))
    if request.method == 'GET':
        [submitted, randomize] = form_submissions.get_filters(request.args, True)
        categoryIsAll = True if submitted['category'] == 'All' else False
        if randomize:
            postings = database_helpers.generate_random_postings()
        else:
            if submitted['search'] == '':
                postings = Posting.query.filter(
                                    and_(
                                        Posting.price >= float(submitted['minPrice']),
                                        Posting.price <= float(submitted['maxPrice']),
                                         or_(Posting.category.contains(submitted['category']), categoryIsAll)
                                         )
                                    ).join(User).with_entities(
                                        Posting.postid, Posting.userid, User.username,
                                        Posting.title, Posting.price, Posting.description,
                                        User.rating)
            else:
                search_elems = form_submissions.get_search_elems(submitted['search'])
                postings = Posting.query.filter(and_(
                                        and_(Posting.tags.like(e) for e in search_elems),
                                        Posting.price >= float(submitted['minPrice']),
                                        Posting.price <= float(submitted['maxPrice']),
                                        or_(Posting.category.contains(submitted['category']), categoryIsAll)
                                        )
                                    ).join(User).with_entities(
                                        Posting.postid, Posting.userid, User.username,
                                        Posting.title, Posting.price, Posting.description,
                                        User.rating)
    else:
        submitted = form_submissions.get_filters('', True)
        postings = database_helpers.generate_random_postings()

    page = form_submissions.get_page_number(submitted['page'], postings.count(), app.config['POSTS_PER_PAGE'])
    postings = postings.paginate(page, app.config['POSTS_PER_PAGE'], False)
    next_url = url_for('user_home_screen',search=submitted['search'], category=submitted['category'], minPrice=submitted['minPrice'], maxPrice=submitted['maxPrice'], page=postings.next_num) if postings.has_next else None
    prev_url = url_for('user_home_screen',search=submitted['search'], category=submitted['category'], minPrice=submitted['minPrice'], maxPrice=submitted['maxPrice'], page=postings.prev_num) if postings.has_prev else None
    return render_template('user-view.html', next_url=next_url, prev_url=prev_url, page=page, categories=CATEGORIES, able_to_filter=True, submitted=submitted, current_user_id=g.user.userid, current_user_is_auth=(g.user.userid > 0), page_title=title, css_file=helper_functions.generate_linked_files('user-view'), filtered_postings=postings.items)

############################ CLAIMS ###########################################
```

```python
# the claim pages
@app.route('/claim', methods=['GET', 'POST'])
def claim_submission():
    if g.user is None:
        return redirect(url_for('login'))
    title = "Claim Submission"
    error = ''
    IsSeller = False
    post_info = {'title': '', 'postid': '', 'username': '' }
    try:
        [posting_info, poster_info] = database_helpers.get_posting_by_id(request.args.get('postid'))
        if posting_info is None:
            return redirect(url_for('error'))
        post_info = helper_functions.get_post_info_claims(posting_info, poster_info)
        isSeller = (posting_info.userid == g.user.userid)
        if isSeller:
            title = "Seller " + title
        else:
            title = "Buyer " + title
    except Exception as e:
        error = "Please Try Resubmitting. Something went Wrong."
        return render_template('claim.html', error=error, post=post_info, isSeller=False, current_user_id=g.user.userid, current_user_is_auth=(g.user.userid > 0), page_title=title, css_file=helper_functions.generate_linked_files('claim') )

    if request.method == 'POST':
        [submitted, error] = form_submissions.get_new_claims_form(request.form, g.user, post_info['postid'])
        if len(error) == 0:
            [completed_claim, claim] = database_helpers.add_claim(submitted, post_info['postid'], g.user)
            if completed_claim:
                is_transaction_completed = database_helpers.check_for_transaction(claim)
                if is_transaction_completed:
                    db.session.commit()
                    return redirect(url_for('claim_completion', is_transaction_complete=True ))
                elif is_transaction_completed is not None:
                    db.session.commit()
                    return redirect(url_for('claim_completion', is_transaction_complete=False ))
            error = "Please resubmit your claim. There was an issue. You may have entered invalid data."
    return render_template('claim.html', error=error, post=post_info, isSeller=isSeller, current_user_id=g.user.userid, current_user_is_auth=(g.user.userid > 0), page_title=title, css_file=helper_functions.generate_linked_files('claim') )

@app.route('/claim-complete', methods=['GET', 'POST'])
def claim_completion(is_transaction_complete=False):
    if g.user is None:
        return redirect(url_for('login'))
    is_transaction_complete = request.args.get('is_transaction_complete')
    print(is_transaction_complete)
    return render_template('claim-complete.html', is_transaction_complete=is_transaction_complete=="True", title="Claim Complete", error='', current_user_id=g.user.userid, current_user_is_auth=(g.user.userid > 0), css_file=helper_functions.generate_linked_files('claim'))

# The HELP page
@app.route('/help-and-FAQ')
def help():
    return render_template('help.html')


```

```python
@app.route('/remove-posting', methods=['GET', 'POST'])
def remove_posting_view(error=""):
    title = 'Remove Posting'
    if g.user is None:
        return redirect(url_for('login'))

    posting_info = database_helpers.get_post_id(request.args.get('postid'))
    if posting_info is None or g.user.userid != posting_info.userid:
        return redirect(url_for('user_home_screen'))

    if request.method == 'POST':
        posting_info_remove =
        Posting.query.filter_by(postid=request.args.get('postid')).first()
        db.session.delete(posting_info_remove)
        db.session.commit()
        return redirect(url_for('user_home_screen'))

    return render_template('remove-posting-view.html',contact_options=CONTACT_METHOD,
    categories=CATEGORIES, js_file="tag-javascript.js", current_user_id=g.user.userid,
    current_user_is_auth=(g.user.userid > 0),  user_id=g.user.userid,
    CURRENT_USER_ID=g.user.userid, page_title=title, error=error,
    css_file=helper_functions.generate_linked_files('create-posting-view'),
    post=posting_info)

# Admin view of users
@app.route('/admin-view/users', methods=['GET', 'POST'])
def admin_view_users():
    if g.user is None:
        return redirect(url_for('login'))
    elif g.user.userid != 1:
        return redirect(url_for('user_home_screen'))
    else:
        if request.method == 'POST':
            database_helpers.delete_user(request.form.get('user_id'))
        UserQuery = User.query.order_by(User.userid).all()
        return render_template('admin-view-users.html', UserQuery=UserQuery)

# Admin view of postings
@app.route('/admin-view/postings', methods=['GET', 'POST'])
def admin_view_postings():
    if g.user is None:
        return redirect(url_for('login'))
    elif g.user.userid != 1:
        return redirect(url_for('user_home_screen'))
    else:
        if request.method == 'POST':
            database_helpers.archivedPost(request.form.get('post_id'))
            db.session.commit()
        PostingQuery = Posting.query.order_by(Posting.postid).all()
        return render_template('admin-view-postings.html', PostingQuery=PostingQuery)

# Admin view of creating accounts
@app.route('/admin-view/create-account', methods=['GET', 'POST'])
def admin_view_create_account(error=""):
    if g.user is None:
        return redirect(url_for('login'))
    elif g.user.userid != 1:
        return redirect(url_for('user_home_screen'))
    else:
        title = "Create a User"
        CREATE_ERROR = "Need to fill in ALL fields marked with an '*'"
        errors = []
        if request.method == 'POST':
            [results, errors] = form_submissions.generate_new_account_form(request.form)
            if len(errors) == 0:
                added_successfully = database_helpers.add_user(results)
                if added_successfully:
                    return redirect(url_for('admin_view_users'))
                errors = "Could not process. Try again."
        return render_template('create-account.html', current_user_is_auth=False,
        error=errors, page_title=title,
        css_file=helper_functions.generate_linked_files('create-account'), )

```