```
1
   import os
   basedir = os.path.abspath(os.path.dirname( file ))
3
   class Config(object):
4
5
       SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'
6
        SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
7
            'sqlite:///' + os.path.join(basedir, 'app.db')
8
        SQLALCHEMY_TRACK_MODIFICATIONS = False
9
        POSTS PER PAGE = 30
10
```

```
from app import app, socketio

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

```
from app import models, db
     from sqlalchemy import *
 3
     from datetime import datetime
 4
 5
    def generate random postings():
 6
         result = models.Posting.query.join(models.User).with entities(
 7
             models.Posting.postid, models.Posting.userid, models.User.username,
 8
             models.Posting.title, models.Posting.price, models.Posting.description,
 9
             models.User.rating)
10
         return result
11
12
     def add new post(form input, current user):
13
         db.session.add (models.Posting (
14
             userid = current user.userid,
15
             date = datetime.now(),
16
             title = form input['title'],
             description = form input['description'],
17
18
             price = form input['price'],
19
             category = form input['category'],
20
             contactmethod = form input['contactmethod'],
21
             tags = form input['tags']))
22
         db.session.commit()
23
         return True
2.4
25
    def add user(new user info):
26
         try:
27
             newUser = models.User(
28
                username = new user info['username'],
29
                                 = new user info['email'],
                 email
30
                 password
                                = new_user_info['password'],
31
                 phonenumber
                                = new_user_info['phone'],
                 personalemail = new user info['personalemail'],
32
33
                 bio
                                 = new_user_info['bio'],
34
                 rating
                                 = float(new user info['rating']),
35
                                 = int(new user info['numRatings']),
                 numRatings
36
37
            db.session.add(newUser)
38
            db.session.commit()
39
             db.session.refresh(newUser)
40
             if newUser.userid is None:
41
                 return False
42
             return True
43
         except Exception as e:
44
             return False
45
         return False
46
47
    def get post id(postid):
48
         try:
49
             if postid is None and not int(postid) > 0:
50
                 return None
51
         except Exception as e:
52
             return None
53
         current post info = models.Posting.query.filter_by(postid=postid).join(
54
             models.User).with entities (
55
             models.Posting.postid, models.Posting.title, models.Posting.category,
56
             models.Posting.price, models.Posting.description,
57
             models.Posting.contactmethod, models.Posting.tags,
58
             models.User.phonenumber, models.User.email, models.User.userid,
59
             models.User.username, models.User.rating, models.User.personalemail
60
         ).first()
61
         return current post info
62
63
    def modify post by id(results, postid):
64
         models.Posting.query.filter by(postid=postid).update(dict(results))
65
         db.session.commit()
66
67
     def update current user(current user, result):
```

```
68
          if 'password' in result:
 69
              current user.password = result['password']
 70
          current user.phonenumber = result['phonenumber']
 71
          current user.personalemail = result['personalemail']
 72
          current user.bio = result['bio']
 73
          db.session.commit()
 74
          return True
 75
 76
      def get posting by id (postid):
 77
          posting info = models.Posting.query.filter by(postid=postid).first()
 78
          if posting info is None:
 79
              return None, None
 80
          if posting info.contactmethod == 'personalemail':
              poster info =
 81
              models.User.query.filter by (userid=posting info.userid).with entities (
 82
                  models.User.personalemail, models.User.rating, models.User.userid,
                  models.User.username
 83
              ).first()
 84
          elif posting info.contactmethod == "phonenumber":
 85
              poster info =
              models.User.query.filter by (userid=posting info.userid).with entities (
 86
                  models.User.phonenumber, models.User.rating, models.User.userid,
                  models.User.username
 87
              ).first()
 88
          else:
 89
              poster info =
              models.User.query.filter by (userid=posting info.userid).with entities (
 90
                  models.User.email, models.User.rating, models.User.userid,
                  models.User.username
 91
              ).first()
 92
          return posting_info, poster_info
 93
 94
      def get user by email(email):
 95
          return models.User.query.filter by(email=email).first()
 96
 97
      def add claim(form, postid, user):
 98
          try:
 99
              post info, poster info = get posting by id(postid)
              if user.userid == post info.userid and not form['buyeremail'] == False:
100
101
                  buyer info = get user by email(form['buyeremail'])
102
                  newClaim = models.Claim(
                      postid
103
                                      = postid,
104
                      sellerid
                                      = user.userid,
105
                      buyerid
                                      = buyer info.userid,
106
                      usersubmitted = user.userid,
107
                      date
                                      = datetime.now(),
108
                      Rating
                                      = form['rating']
109
                  )
110
                  db.session.add(newClaim)
111
                  return True, newClaim
112
              elif not user.userid == post info.userid:
113
                  newClaim = models.Claim(
114
                      postid
                                      = postid,
115
                      sellerid
                                      = post info.userid,
116
                      buyerid
                                      = user.userid,
117
                      usersubmitted = user.userid,
118
                      date
                                      = datetime.now(),
119
                      Rating
                                      = form['rating']
120
                  )
121
                  db.session.add(newClaim)
122
                  return True, newClaim
123
          except Exception as e:
              db.session.rollback()
124
125
              return False, False
126
          return False, False
127
128
      def check for transaction(claim):
```

```
129
          try:
130
              other claim = models.Claim.query.filter(
131
                  models.Claim.postid == claim.postid,
                  models.Claim.sellerid==claim.sellerid,
132
133
                  models.Claim.buyerid == claim.buyerid,
134
                  models.Claim.usersubmitted!=claim.usersubmitted
135
136
              if other claim is not None:
137
                  newTransaction = models.Transaction(
138
                                      = datetime.now(),
139
                      claimidseller
                                     = claim.sellerid,
140
                      claimidbuyer
                                     = claim.buyerid
141
                  )
142
                  db.session.add(newTransaction)
                  print("Try to Alter Ratings!")
143
144
                  alter ratings (claim, other claim)
145
                  print("Try to Archive!")
146
                  if archive posting(claim.postid, newTransaction):
                      print("Archived! Now Delete The Claims:")
147
148
                      delete claim(claim.claimid)
149
                      delete claim (other claim.claimid)
150
                      print("Finished")
151
                      return True
152
                  else:
153
                      raise ValueError
154
          except Exception as e:
155
              print("Rollback in check for transaction")
156
              db.session.rollback()
157
              return None
158
          return False
159
160
      def delete claim(claimid):
161
          try:
162
              models.Claim.query.filter by(claimid=claimid).delete()
163
          except Exception as e:
164
              pass
165
166
     def delete user(userid):
167
          try:
168
              someUser = User.query.filter by(userid=someUserID).first()
169
              db.session.delete(someUser)
170
              db.session.commit()
171
          except Exception as e:
172
              pass
173
174
175
      def archive posting(postid, transaction=None):
          try:
176
177
              if postid is not None:
178
                  post = models.Posting.query.filter by(postid=postid).first()
179
                  print("got Post")
180
                  archivedPost = models.ArchivedPosting(
181
                      transactionid = transaction.transactionid,
182
                      postid = post.postid,
183
                                      = transaction.claimidbuyer,
                      buyerid
184
                      sellerid
                                       = transaction.claimidseller,
185
                      date
                                = datetime.now(),
186
                      title
                                  = post.title,
187
                      description = post.description,
188
                      price
                                  = post.price,
189
                      category
                                  = post.category,
190
                      contactmethod
                                     = post.contactmethod,
191
                      tags
                                = post.tags
192
                  )
193
              else:
194
                  archivedPost = models.ArchivedPosting(
195
                      postid
                                  = post.postid,
```

```
196
                                = datetime.now(),
                      date
197
                      title
                                  = post.title,
198
                      description = post.description,
199
                      price
                                  = post.price,
200
                      category = post.category,
201
                      contactmethod = post.contactmethod,
202
                                = post.tags
203
                  )
204
205
              db.session.add(archivedPost)
206
              db.session.delete(post)
207
              return True
208
          except Exception as e:
209
              db.session.rollback()
210
          return False
211
212
      def get new rating(current rating, current number, new rating):
213
          current number += 1
          current rating = current rating * (current number-1)/current number + new rating *
214
          1/current number
215
          return [current rating, current number]
216
217
      def alter ratings(claim, other claim):
218
          print("Getting Ratings")
219
          user1 = models.User.query.filter by(userid=claim.usersubmitted).first()
220
          user2 = models.User.query.filter by(userid=other claim.usersubmitted).first()
221
          print("Update the Ratings Manually")
222
          [userl.rating, userl.numRatings] = get new rating(userl.rating, userl.numRatings,
          other claim. Rating)
223
          [user\overline{2}.rating, user2.numRatings] = get new rating(user2.rating, user2.numRatings,
          claim.Rating)
224
          print("FINISHED alter ratings")
225
     def get postings(userid):
226
227
          try:
228
              postings = models.Posting.query.filter by (userid=userid).all()
229
              print("Got postings for user")
230
              return postings
231
          except Exception as e:
232
              pass
233
          return None
234
235
      def get claims (userid):
236
          try:
237
              claims =
              models.Claim.query.filter by (usersubmitted=userid).join(models.Posting).with enti
238
                  models.Claim.date, models.Posting.title, models.Posting.postid
239
              ).all()
240
              print("Got Claims")
241
              return claims
242
          except Exception as e:
243
              pass
244
          return None
245
246
      def get sales(userid):
247
          try:
              sales = models.ArchivedPosting.query.filter by(sellerid=userid).with entities(
248
249
                  models.ArchivedPosting.title, models.ArchivedPosting.buyerid,
                  models.ArchivedPosting.price,
250
                  ).join(
251
                  models.Transaction).join(models.User, models.User.userid ==
                  models.ArchivedPosting.buyerid).with entities(
                          models.Transaction.date, models.ArchivedPosting.title,
252
                          models.ArchivedPosting.price,
253
                          models.User.username, models.User.userid).all()
254
              print("Got sales")
```

```
255
              return sales
256
          except Exception as e:
257
              pass
258
          return None
259
260
     def get purchases(userid):
261
          try:
262
              purchases = models.ArchivedPosting.query.filter_by(buyerid=userid).with_entities(
263
                  models.ArchivedPosting.title, models.ArchivedPosting.sellerid,
                  models.ArchivedPosting.price,
264
                  ).join(
265
                  models.Transaction).join(models.User, models.User.userid ==
                  models.ArchivedPosting.sellerid).with entities(
266
                          models.Transaction.date, models.ArchivedPosting.title,
                          models.ArchivedPosting.price,
267
                          models.User.username, models.User.userid).all()
268
              print("Got purchases")
269
              return purchases
270
          except Exception as e:
271
              pass
          return None
272
```

273

```
import re
1
2
    from werkzeug.security import check password hash, generate password hash
3
4
    # Returns email if in valid format; else returns false
5
    def get email(email field):
        regex = '^{+} + ([\overline{\ \ \ \ }] ? \w+ ) *@ (\w+.) *pitt.edu'
6
7
        old regex = '\w+@(\w+.)*pitt.edu'
8
        try:
9
             if re.search(regex, str(email field)) is not None:
10
                return email field
11
            raise ValueError
12
        except Exception as e:
13
            return False
14
15
   # verify password given in form; true for correct password, otherwise false
# user: takes USER class from model;
17
            for TESTING object with value string of HASHED password
18
    # password: string of UNhashed password passed in by the user on login
19
    def verify password(user, password):
20
        try:
21
            if user is None or not check password hash(user.password, str(password)):
22
                return False
23
            return True
24
        except Exception as e:
25
            raise
26
27
28
29
    30
   # Home View Search and filter methods
31
32
   def get category(field):
33
        try:
34
            if str(field) == "All":
35
                return 'All'
36
            return str(field)
37
        except Exception as e:
38
            return 'All'
39
                    MAY WANT TO EDIT THIS
40
41
    # takes inputted search textbox input for backend search
42
    def get search text(field):
43
        try:
44
             if str(field).strip() == '':
45
                return ''
46
            return field
47
        except Exception as e:
48
            return ''
49
50 def get search elems(search):
51
        try:
52
            elems = search.split()
53
            return search = []
54
            for e in elems:
55
                return search.append('%' + e + '%')
            return return search
56
57
        except Exception as e:
58
            pass
59
        return ''
60
61
    def get page number(field, number postings, posts per page):
62
        try:
63
            field = int(field)
64
            if field > 0 and field <= (int(number postings/posts per page)+1):</pre>
65
                return field
66
        except Exception as e:
67
            pass
```

```
68
          return 1
 69
 70
      def get max price(field, min field):
 71
          try:
 72
              field = float(field)
 73
              min field = float (min field)
 74
              if field > 0 and field > min field and field < 2000:</pre>
 75
                   return str(field)
 76
              else:
 77
                   return '2000'
 78
          except Exception as e:
 79
              return '2000'
 80
 81
      def get min price(field):
 82
          try:
              field = float(field)
 83
 84
              if field > 0 and field <= 2000:</pre>
 85
                   return str(field)
 86
              elif field > 2000:
 87
                  return '2000'
 88
              else:
                   return '0'
 89
 90
          except Exception as e:
              return '0'
 91
 92
 93
      def should randomize(submitted):
 94
          return submitted == {'minPrice': '0', 'maxPrice': '0', 'search': '', 'category':
          '', 'page': 1}
 95
 96
      def get page(field):
 97
          try:
 98
              return int(field)
 99
          except Exception as e:
100
              pass
101
          return 1
102
103
      def get filters(forms, get recieved):
104
          if get recieved and forms.get('minPrice') is not None:
105
              submitted = {}
106
              submitted['minPrice'] = get min price(forms['minPrice'])
107
              submitted['maxPrice'] = get max price(forms['maxPrice'], submitted['minPrice'])
108
              submitted['search'] = get search text(forms['search'])
109
              submitted['category'] = get category(forms['category'])
110
              submitted['page']
                                       = get page(forms['page']) if 'page' in forms else 1
111
              return [submitted, should randomize(submitted)]
112
          else:
113
              return [{'minPrice': '0', 'maxPrice': '0', 'search': '', 'category': '',
               'page': 1}, True]
114
115
116
      ############################### New Posting Submission #########################
117
      # verifies it is 30 chars or shorter
118
      def validate title(field):
119
          try:
120
              field = str(field)
121
              if len(field) > 0 and len(field) <31:</pre>
122
                   return [True, field]
123
          except Exception as e:
124
              return [False, "The title needs to be 1-30 characters long."]
125
          return [False, "The title needs to be 1-30 characters long. Your input was " +
          str(len(field)) + " characters."]
126
127
      # this should never really fail. It is on us if it does
128
      def validate category(field, CATEGORIES):
129
          try:
              field = str(field)
130
131
              if field in CATEGORIES:
```

```
132
                  return [True, field]
133
          except Exception as e:
134
              return [False, "Category was not specified. Try resubmitting!"]
135
          return [False, "Invalid preffered contact method. Try resubmitting."]
136
137
      # validates that price is in range of 0-2000; cuts of after 2nd decimal place
138
      def validate price(field):
139
          try:
140
              field = round(float(field), 2)
141
              if field > 0 and field <= 2000:</pre>
142
                  return [True, field]
143
              raise ValueError
144
          except ValueError as e:
              return [False, "Invalid price. Needs to be in the range of 0 to 2,000
145
              inclusive."]
146
          return [False, "Invalid preffered contact method. Try resubmitting."]
147
      # ensure description is less than 1001 chars long.
148
      def validate desc(field):
149
150
          try:
151
              field = str(field)
152
              if len(field) > 1000:
153
                  return [False, "Your short description cannot be longer than 1,000
                  characters long."]
154
          except Exception as e:
              return [False, "Invalid short description."]
155
156
          return [True, field]
157
158
      # returns preferred contact value; should alk
159
      def validate preferred contact(field):
160
          try:
161
              field = str(field)
              if field == "email" or field == "phonenumber" or field == "personalemail":
162
163
                  return [True, field]
164
          except Exception as e:
165
              return [False, "Invalid preffered contact method. Try resubmitting."]
          return [False, "Invalid preffered contact method. Try resubmitting."]
166
167
168
      def validate preferred tags(field, title):
169
          try:
170
              title = str(title[1]).split()
              field = str(field) + ',' + ','.join(title)
171
172
              field = ''.join(field.split()).lower().split(',')
173
              if len(field) < 900:</pre>
174
                  if len(field) > 50:
175
                      return [False, "Too many tags. The maximum tags are 50. Maximum
                      character limit is 900"]
176
                  field = list(set(field))
177
                  field = ','.join(field)
178
                  return [True, field]
179
              raise ValueError
180
          except Exception as e:
              return [False, "Too many tags. Maximum character limit is 900"]
181
182
183
      def validate input(forms, CATEGORIES):
184
          result = {}
185
          result['title'] = validate title(forms['title'])
186
          result['category'] = validate category(forms['category'], CATEGORIES)
187
          result['price'] = validate price(forms['price'])
          result['description'] = validate desc(forms['description'])
188
          result['contactmethod'] = validate preferred contact(forms['preferredContact'])
189
190
          result['tags'] = validate preferred tags(forms['tags'], result['title'])
191
          return result
192
193
     def generate return values(given):
194
          error = []
195
          good results = {}
```

```
196
          for key, elem in given.items():
197
              if elem[0]:
198
                  good results[key] = elem[1]
199
              else:
200
                  error.append(str(elem[1]))
201
          return good results, error
202
203
204
      def get form create post(forms, CATEGORIES):
205
          initial = validate input(forms, CATEGORIES)
206
          return generate return values (initial)
207
      208
209
      # The CREATE ACCOUNT Functions
210
      def generate new account form(forms):
211
          results = generate fields create account (forms)
212
          return generate return values (results)
213
214
215
     def get username(email):
216
          try:
217
              username = str(email).split('@')
218
              if len(username) == 2 and username[0] != '':
219
                  return [True, username[0]]
220
              raise ValueError
221
          except Exception as e:
222
              return [False, 'An unexpected error has occurred.']
223
224
225
     def generate fields create account(forms):
226
         new account info = {}
227
          new account info['email']
                                              = validate email(forms['email'])
228
         new_account_info['username']
                                             = get username(new account info['email'][1])
229
         new account info['password']
                                             = validate password(forms['password'],
         forms['password2'])
230
         new account info['phone']
                                              = validate phone number (forms ['phonenumber'])
231
         new account info['personalemail']
                                            = validate personal email(forms['personalemail'])
                                             = validate bio(forms['bio'])
         new account info['bio']
232
                                             = [True, \overline{5}]
233
         new account info['rating']
                                             = [True, '0']
          new account info['numRatings']
234
235
          return new account info
236
237
     def validate email(field):
238
          try:
239
              field = get email(str(field))
240
              if not field == False and len(field) < 75:</pre>
241
                  return [True, field]
242
              raise ValueError
243
          except Exception as e:
244
              return [False, "The university email must end with a school domain (pitt.edu)."]
245
246
      def validate password(password1, password2):
247
          try:
248
              password1 = str(password1)
249
              password2 = str(password2)
250
              if password1 == password2:
251
                  if len(password1) > 7 and len(password1) < 33:</pre>
252
                      return [True, generate password hash(password1)]
253
              raise ValueError
254
          except Exception as e:
255
              return [False, "Both Password fields must match and have between 8 and 32
              characters (inclusive)."]
256
257
      def convert number(phone):
         phone = phone.replace('-', '')
258
259
         phone = phone.replace('(', '')
260
         phone = phone.replace(')', '')
```

```
261
          if len(phone) == 10:
              phone = '1' + phone
262
263
          elif len(phone) == 0:
264
              return ''
265
          elif not len(phone) == 11 or not phone[0] or len(phone) == '1':
266
              raise ValueError
267
          return int(phone)
268
269
      def convert again number(phone):
270
          converted = phone[0] + "(" + phone[1:4] + ")" + phone[4:7] + "-" + phone[7:10]
271
          return converted
272
273
      ### ADD MORE HERE ###
274
     def validate phone number(field):
275
          try:
276
              phone number = str(field)
277
              phone number = convert number(phone number)
278
              phone number = convert again number(str(phone number)) if phone number != ''
              else ''
279
              return [True, phone number]
280
          except Exception as e:
              return [False, "Phone number must be 10 characters long or 11 characters long
281
              with the country code being '1' in order to be processed."]
282
          return [True, phone number]
283
284
285
      def validate personal email(field):
286
          email regex = '^{w+([.-]?\w+)*@\w+([.-]?\w+)*(.\w{2,3})+$'}
287
          try:
288
              field = str(field)
289
              if re.search(email regex, field) is not None:
290
                  return [True, field]
291
          except Exception as e:
292
              return [False, "The personal email address is not a legal value."]
293
          return [True, field]
294
295
     def validate bio(field):
296
          try:
297
              field = str(field)
298
              if len(field) < 251:</pre>
299
                  return [True, field]
300
          except Exception as e:
301
              return [False, "The biography is unable to be processed. Possibly an invalid
              symbol."]
302
          return [False, "Length exceeds 250 characters"]
303
304
      def get modified account info(forms, userid):
305
          result = generate fields edit account(forms, userid)
306
          return generate return values(result)
307
308
     def validate password simple (password):
309
          try:
310
              password = str(password)
311
              return [True, password]
312
          except Exception as e:
313
              return [False, "Try Resubmitting information."]
314
315
      def validate delete(forms):
316
          try:
317
              if forms['deleteaccount'] == 'delete':
                  return [True, 'delete']
318
319
          except Exception as e:
320
              pass
321
          return [False, "nothing"]
322
323
      def generate fields edit account(forms, userid):
324
          new account info = {}
```

```
325
         new account info['userid']
                                             = [True, str(userid)]
326
         if forms['newpassword'] and forms['newpassword'] != '':
327
              new account info['password']
                                            = validate password(forms['newpassword'],
              forms['newpassword2'])
328
         new account info['oldpassword']
                                             = validate password simple(forms['oldpassword'])
329
         new account info['phonenumber']
                                             = validate phone number (forms['phonenumber'])
330
         new account info['personalemail']
                                             = validate personal email(forms['personalemail'])
         new_account info['bio']
331
                                             = validate bio(forms['bio'])
332
         new account info['deleteaccount']
                                            = validate delete(forms)
333
         return new account info
334
      335
336
     # Claims
337
     def get new claims form(forms, current user, postid):
338
         results = generate claims forms (forms, current user, postid)
339
         return generate return values(results)
340
     def generate claims forms (forms, current user, postid):
341
342
         claim info = {}
343
         claim info['postid']
                                     = [True, postid]
344
         claim info['userid']
                                     = [True, current user.userid]
345
         claim info['rating']
                                     = validate rating claims (forms ['rating'])
346
         claim info['buyeremail']
                                    = validate buyer email(forms)
347
         return claim info
348
349
     def validate rating claims(field):
350
         try:
351
              field = int(field)
352
             if field > 0 and field < 6:</pre>
353
                 return [True, field]
354
             raise ValueError
355
         except Exception as e:
356
             return [False, "Please resubmit your claim!"]
357
358
     def validate buyer email(forms):
359
         if 'buyeremail' in forms:
360
             try:
361
                 field = str(forms['buyeremail'])
362
                 if get email(field) != False:
363
                     return [True, get email(field)]
364
             except Exception as e:
365
                 pass
366
         return [True, False]
```

367

```
from flask import Flask, session
    from config import Config
    from flask_sqlalchemy import SQLAlchemy
3
    from flask_socketio import SocketIO, emit
4
5
6
7
   print("APP NAME: " + str(__name__))
8 app = Flask(__name__)
9 app.config.from object(Config)
10 db = SQLAlchemy(app)
db.init app(app)
12
   socketio = SocketIO(app)
13
14
15
   from app import routes, models
16
```

```
1
     from flask sqlalchemy import SQLAlchemy
     from sqlalchemy.orm import relationship
 3
     from app import app, db
 4
     from werkzeug.security import generate password hash, check password hash
 5
     from datetime import datetime
 6
     from app.form submissions import get username, validate phone number
 7
     import csv
 8
 9
10
     # INITS THE DBs for USERs
11
     def add postings(file):
         list = []
12
13
         key list = ['userid', 'title', 'description', 'price', 'category', 'contactmethod']
14
         with open(file, 'r') as csv file:
15
             reader = csv.DictReader(csv file, delimiter=',', fieldnames=key list)
             for row in reader:
16
17
                 new item = \{ \}
18
                 for key in key list:
19
                     new item[key] = row[key]
20
                 list.append(new item)
21
         for value in list:
22
             tags = value['title'].split()
23
             tags = ','.join(tags)
24
             newPosting = Posting(
25
                                 = value['userid'],
                 userid
26
                 date
                                 = datetime.now(),
27
                 title
                            = value['title'],
28
                 description = value['description'],
29
                            = value['price'],
                 price
30
                             = value['category'],
                 category
31
                 contactmethod = value['contactmethod'],
32
                                 = tags
33
             )
34
             db.session.add(newPosting)
35
             db.session.commit()
36
37
    def add users(file):
38
39
         list = []
         key list = ['phonenumber', 'email', 'personalemail', 'password', 'bio']
40
41
         with open(file, 'r') as csv file:
42
             reader = csv.DictReader(csv file, delimiter=',', fieldnames=key list)
43
             for row in reader:
44
                 new item = \{\}
45
                 for key in key list:
46
                     new item[key] = row[key]
47
                 new item['phonenumber'] = validate phone number(new item['phonenumber'])[1]
48
                 new item['username'] = get username(new item['email'])
49
                 list.append(new item)
50
         for value in list:
51
             newUser = User(
52
                                    = value['username'][1],
                 username
53
                                    = value['email'],
                 email
54
                                    = value['personalemail'],
                 personalemail
                                     = generate_password hash(value['password']),
55
                 password
56
                 phonenumber
                                    = value['phonenumber'],
57
                 bio
                                     = value['bio'],
58
                 rating
                                     = 5,
59
                 numRatings
                                     = 0
60
61
             db.session.add(newUser)
62
             db.session.commit()
63
64
65
    @app.cli.command('initdb')
     def initdb command():
66
67
         # wipeout
```

```
68
          db.drop all()
 69
          db.create all()
 70
 71
          add users("sampleUser.csv")
 72
          # add some default data
 73
          # 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))
 74
          # 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",
 75
          password=generate password hash('foobar'), phonenumber='2456734224',
          personalemail='tester1@gmail.com', bio='Tester is testing account for testing...',
          rating=3, numRatings=10))
 76
 77
          add postings("postingsData.csv")
 78
 79
          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',
 80
          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',
 81
          description='Great quality.', price=20.00, category='Textbooks',
          contactmethod='personalemail', tags='book'))
 82
 83
          db.session.commit()
 84
 85
          print('Initialized the database.')
 86
 87
 88
      class User(db.Model):
 89
         userid
                      = db.Column(db.Integer, primary key = True)
 90
                        = db.Column(db.String(24), nullable = False)
          username
 91
                      = db.Column(db.String(80), unique=True, nullable = False)
          email
 92
          # hashed password is ~100 chars ALWAYS
 93
          password
                    = db.Column(db.String(128), nullable = False)
          phonenumber = db.Column(db.String(64), nullable = False)
 94
 95
          personalemail = db.Column(db.String(80), nullable = False)
                      = db.Column(db.String(250), nullable = False)
 96
          bio
 97
          rating
                      = db.Column(db.Float(2), nullable = False)
 98
                        = db.Column(db.Integer, nullable = False)
          numRatings
 99
                          = relationship("Posting", cascade="all,delete", backref="User")
          postings
100
101
          def repr (self):
102
              return '<User {}>'.format(self.username)
103
104
105
      class Posting(db.Model):
106
                      = db.Column(db.Integer, primary key = True)
          postid
107
          userid
                      = db.Column(db.Integer, db.ForeignKey("user.userid"))
108
          date
                    = db.Column(db.Date, nullable = False)
109
          title
                     = db.Column(db.String(30), nullable = False)
110
          description = db.Column(db.String(250), nullable = False)
111
          price
                     = db.Column(db.Integer, nullable = False)
112
          category
                      = db.Column(db.String(80), nullable = False)
113
          contactmethod
                        = db.Column(db.String(80), nullable = True)
                    = db.Column(db.String(1000), nullable = True)
114
          tags
115
          claims
                        = relationship("Claim", cascade="all,delete", backref="Posting")
116
117
          def repr (self):
118
              return '<Posting {}: "{}">'.format(self.postid, self.title)
119
```

```
120
121
     class Claim(db.Model):
          _{\rm table\ args} = (
122
              db. UniqueConstraint ('postid', 'sellerid', 'buyerid', 'usersubmitted',
123
              name='unique claim buyer seller'),
124
          )
125
          claimid
                      = db.Column(db.Integer, primary key = True)
126
                      = db.Column(db.Integer, db.ForeignKey("posting.postid"))
         postid
127
         sellerid
                      = db.Column(db.Integer, db.ForeignKey("user.userid"))
                    = db.Column(db.Integer, db.ForeignKey("user.userid"))
128
         buyerid
129
         usersubmitted = db.Column(db.Integer, db.ForeignKey("user.userid"))
130
                    = db.Column(db.Date, nullable = False)
          date
131
         Rating
                          = db.Column(db.Integer, nullable = False)
132
133
          def repr (self):
134
              return '<Claim {}: "{}">'.format(self.claimid)
135
136
     class Transaction(db.Model):
137
138
          transactionid = db.Column(db.Integer, primary key = True)
139
                          = db.Column(db.Date, nullable = False)
140
          claimidseller = db.Column(db.Integer, db.ForeignKey("claim.claimid"), nullable =
          False)
141
          claimidbuyer = db.Column(db.Integer, db.ForeignKey("claim.claimid"), nullable =
          False)
142
143
          def __repr__(self):
144
              return '<Transaction {}: "{}">'.format(self.transactionid)
145
146
147
     class ArchivedPosting(db.Model):
148
          _{\rm table\_args} = (
              db.UniqueConstraint('postid', 'buyerid', 'archivedpostid', 'sellerid',
149
              name='unique archive posting constraint'),
150
151
          archivedpostid = db.Column(db.Integer, primary key = True)
                        = db.Column(db.Integer, db.ForeignKey('transaction.transactionid'),
152
          transactionid
         nullable = True)
153
                     = db.Column(db.Integer, nullable = False)
          postid
                         = db.Column(db.Integer, db.ForeignKey("user.userid"), nullable =
154
          buyerid
          True)
155
                         = db.Column(db.Integer, db.ForeignKey("user.userid"), nullable =
          sellerid
         True)
156
         date
                    = db.Column (db.Date, nullable = False)
157
         title
                     = db.Column(db.String(80), nullable = False)
158
         description = db.Column(db.String(250), nullable = True)
159
                    = db.Column(db.Integer, nullable = False)
         price
160
         category = db.Column(db.String(80), nullable = False)
161
         contactmethod = db.Column(db.String(80), nullable = True)
162
                   = db.Column(db.String(1000), nullable = True)
163
164
          def repr (self):
165
              return '<Posting {}: "{}">'.format(self.postid, self.title)
```

166

```
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
4
    from random import shuffle
    from datetime import datetime
7
    from sqlalchemy import and , or
8
9
    CATEGORIES = ['All', 'Textbooks', 'Furniture', 'Food', 'Events', 'Software',
    'Electronics',
    'Beauty and Personal Care', 'Clothes', 'School Supplies', 'Appliances']
10
    CONTACT METHOD = {
11
12
        "email": "Email (university provided)",
13
        "phonenumber": "Phone Number (if provided)",
14
        "personalemail": "Personal Email (if provided)"
15
16
17
    # forces logout on browser close(); aka senses packets have stopped flowing
    @socketio.on('disconnect')
18
19
   def disconnect user():
20
        logout()
21
    2.2
23
   # ROUTES START HERE
24
    25
    # Log the user out
26
   @app.route('/logout')
27
   def logout():
28
        session.pop('userid', None)
29
        return redirect(url_for('login'))
30
31
    # Run at the beginning of each request before functions run to check if logged in
32
    @app.before request
33
   def request authentication():
34
        q.user = None
35
        if 'userid' in session:
            g.user = User.query.filter by(userid=session['userid']).first()
36
37
38
    ### ERROR HANDLING PAGES
39
    @app.route('/error')
40
    def not found error item():
41
        return render template('404.html'), 404
42
43 @app.errorhandler(404)
44 def not found error (error):
45
        return render template('404.html'), 404
46
47
   @app.errorhandler(500)
48
   def internal error(error):
49
        db.session.rollback()
50
        return render template('500.html'), 500
51
52
   @app.route('/')
53
    def slash redirect():
54
        return redirect(url for('login'))
55
    56
   # The login screen
57 @app.route('/login', methods=['GET', 'POST'])
58
   def login(error=""):
        title = "Login to Craigversity!"
59
        LOGIN ERROR = "Invalid information was submitted. Please try again!"
60
61
        error = ''
62
        if g.user:
63
            return redirect(url for('user home screen'))
64
        if request.method == "POST":
65
            error = LOGIN ERROR
66
            email = form submissions.get email(request.form['email'])
```

```
67
              if not email == False:
 68
                  user = User.query.filter by(email=email).first()
 69
                  if form submissions.verify password(user, request.form['password']):
                      session['userid'] = user.userid
 71
                      return redirect(url for('user home screen'))
 72
          return render template ('login.html', current user is auth=False, error=error,
          page title=title, css file=helper functions.generate linked files('login'))
 73
 74
      75
      # The create account screen
      @app.route('/create-account', methods=['GET', 'POST'])
 76
      def create account(error=""):
 77
 78
          title = "Welcome to Craigversity!"
 79
          CREATE ERROR = "Need to fill in ALL fields marked with an '*'"
 80
          errors = []
 81
          if g.user:
              return redirect(url for('user home screen'))
 82
 83
          if request.method == 'POST':
              [results, errors] = form submissions.generate new account form(request.form)
 84
 85
              if len(errors) == 0:
                  added successfully = database helpers.add user(results)
 86
 87
                  if added successfully:
 88
                      return redirect(url for('login'))
 89
                  errors = "Could not process. Try again."
 90
          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'), )
 91
 92
      # The user screen
 93
      @app.route('/user', methods=['GET', 'POST'])
 94
      def users account():
 95
          if g.user is None:
 96
              return redirect(url for('login'))
          account info = g.user
 97
          title = "USER: " + g.user.username
 98
 99
          if request.method == "GET":
100
              userid = (request.args.get('userid'))
             account info = User.query.filter by(userid=userid).first()
101
             if account info is None:
102
103
                  return redirect(url for('not found error item'))
104
                         = database helpers.get postings (account info.userid)
             postings
105
             claims
                         = database helpers.get claims (account info.userid)
106
             sales
                         = database helpers.get sales (account info.userid)
107
                         = database helpers.get purchases (account info.userid)
108
              title = "USER: " + account info.username
109
          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)
110
111
      # The edit account screen
      @app.route('/edit-account', methods=['GET', 'POST'])
112
113
      def edit_account(error=""):
114
          if q.user is None:
115
              return redirect(url for('login'))
116
          title = 'Edit Account'
117
         current user = g.user
          error = ''
118
          if request.method == 'POST':
119
120
              [result, error] = form submissions.get modified account info(request.form,
              g.user.userid)
121
              if len(error) == 0 and check password hash(g.user.password,
              str(result['oldpassword'])):
                  if result['deleteaccount'] == "delete":
122
123
                      database helpers.delete user(g.user.userid)
```

```
124
                 if database helpers.update current user(g.user, result):
125
                     return redirect(url for('login'))
126
          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'), )
127
     128
      129
130
      # The new posting submission screen
131
     @app.route('/new-posting-submission', methods=['GET', 'POST'])
132
     def new posting submission(error=""):
133
         title = "Submit a New Posting!"
134
         error = []
135
         if g.user is None:
             return redirect(url for('login'))
136
          if request.method == 'POST':
137
138
              [results, error] = form submissions.get form create post(request.form,
             CATEGORIES)
139
             if len(error) == 0:
140
                 database helpers.add new post (results, g.user)
                 return redirect(url for('login'))
141
         return render template ('create-posting-view.html', contact options=CONTACT METHOD,
142
          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'))
143
144
     # NEED TO REWORK CONTACT METHOD!!!
145
      @app.route('/edit-posting', methods=['GET', 'POST'])
146
     def edit_posting(error=""):
         title = 'Edit Posting'
147
148
         if g.user is None:
149
             return redirect(url for('login'))
150
151
         posting info = database helpers.get post id(request.args.get('postid'))
152
          if posting info is None or g.user.userid != posting info.userid:
             return redirect(url for('user home screen'))
153
154
          if request.method == 'POST':
155
156
              [results, error] = form submissions.get form create post(request.form,
             CATEGORIES)
             if len(error) == 0:
157
158
                 print(results)
159
                 database helpers.modify post by id(results, posting info[0])
160
                 return redirect(url for('user home screen'))
161
         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)
162
163
     # The posting screen
      @app.route('/posting', methods=['GET'])
164
165
     def full posting view():
166
         if q.user is None:
167
             return redirect(url for('login'))
168
          #posting info = {}
          if request.method == 'GET':
169
170
              [posting info, user info] =
              database helpers.get posting by id(request.args.get('postid'))
171
             if posting info is None:
172
                 return redirect(url for('not found error item'))
             title = "POSTING: " + posting info.title
173
         return render template ('full-posting-view.html', current user id=g.user.userid,
174
          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)
175
     176
177
     # The home user logged in screen that lists postings
178
     @app.route('/search-and-filter-postings', methods=['GET'])
179
     def user home screen():
         title = "Search and Filter Postings!"
180
181
         page = 1
182
         if g.user is None:
183
             return redirect(url for('login'))
184
         if request.method == 'GET':
185
              [submitted, randomize] = form submissions.get filters(request.args, True)
186
             categoryIsAll = True if submitted['category'] == 'All' else False
187
188
                 postings = database helpers.generate random postings()
189
             else:
                 if submitted['search'] == '':
190
191
                     postings = Posting.guery.filter(
                                         and (
192
                                             Posting.price >= float(submitted['minPrice']),
193
194
                                             Posting.price <= float(submitted['maxPrice']),</pre>
195
                                              or (Posting.category.contains(submitted['categor
                                              y']), categoryIsAll)
196
197
                                         ).join(User).with entities(
198
                                             Posting.postid, Posting.userid, User.username,
199
                                             Posting.title, Posting.price,
                                             Posting.description,
200
                                            User.rating)
201
                 else:
202
                     search elems = form submissions.get search elems(submitted['search'])
203
                     postings = Posting.query.filter(and (
                                            and (Posting.tags.like(e) for e in search elems),
204
205
                                             Posting.price >= float(submitted['minPrice']),
206
                                             Posting.price <= float(submitted['maxPrice']),</pre>
                                             or (Posting.category.contains(submitted['category
                                             ']), categoryIsAll)
208
209
                                         ).join(User).with entities(
210
                                             Posting.postid, Posting.userid, User.username,
211
                                             Posting.title, Posting.price,
                                             Posting.description,
212
                                            User.rating)
213
         else:
214
             submitted = form submissions.get filters('', True)
215
             postings = database helpers.generate random postings()
216
217
         page = form submissions.get page number(submitted['page'], postings.count(),
         app.config['POSTS PER PAGE'])
218
         postings = postings.paginate(page, app.config['POSTS PER PAGE'], False)
219
         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
220
         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
221
         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)
222
223
```

```
224
      # the claim pages
225
      @app.route('/claim', methods=['GET', 'POST'])
226
      def claim submission():
227
          if q.user is None:
228
              return redirect(url for('login'))
229
          title = "Claim Submission"
230
          error = ''
231
          IsSeller = False
          post info = {'title': '', 'postid': '', 'username': '' }
232
233
              [posting info, poster info] =
234
              database helpers.get posting by id(request.args.get('postid'))
235
              if posting info is None:
236
                  return redirect(url for('error'))
237
              post info = helper functions.get post info claims (posting info, poster info)
238
              isSeller = (posting info.userid == g.user.userid)
239
              if isSeller:
                  title = "Seller " + title
240
241
              else:
                  title = "Buyer " + title
2.42
243
          except Exception as e:
244
              error = "Please Try Resubmitting. Something went Wrong."
              return render template('claim.html', error=error, post=post info,
245
              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') )
246
247
          if request.method == 'POST':
248
              [submitted, error] = form submissions.get new claims form(request.form, g.user,
              post info['postid'])
249
              if len(error) == 0:
250
                  [completed claim, claim] = database helpers.add claim(submitted,
                  post_info['postid'], g.user)
251
                  if completed claim:
                      is transaction completed = database helpers.check for transaction(claim)
252
253
                      if is transaction completed:
254
                          db.session.commit()
                          return redirect (url for ('claim completion',
255
                          is transaction complete=True ))
256
                      elif is transaction completed is not None:
257
                          db.session.commit()
                          return redirect (url for ('claim completion',
2.58
                          is transaction complete=False ))
259
                  error = "Please resubmit your claim. There was an issue. You may have
                  entered invalid data."
260
          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') )
261
262
      @app.route('/claim-complete', methods=['GET', 'POST'])
263
      def claim completion(is transaction complete=False):
264
          if g.user is None:
265
              return redirect(url for('login'))
266
          is transaction complete = request.args.get('is transaction complete')
267
          print(is transaction complete)
268
          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'))
269
270
      # The HELP page
271
      @app.route('/help-and-FAQ')
272
      def help():
273
          return render template('help.html')
274
275
```

```
276
      @app.route('/remove-posting', methods=['GET', 'POST'])
277
      def remove_posting view(error=""):
278
          title = 'Remove Posting'
279
          if q.user is None:
280
              return redirect(url for('login'))
281
282
          posting info = database helpers.get post id(request.args.get('postid'))
          if posting info is None or g.user.userid != posting info.userid:
283
              return redirect(url for('user home screen'))
284
285
286
          if request.method == 'POST':
287
              posting info remove =
              Posting.query.filter by(postid=request.args.get('postid')).first()
288
              db.session.delete(posting info remove)
289
              db.session.commit()
290
              return redirect(url for('user home screen'))
291
292
          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)
293
294
      # Admin view of users
295
      @app.route('/admin-view/users', methods=['GET', 'POST'])
296
      def admin view users():
297
          if g.user is None:
298
              return redirect(url for('login'))
299
          elif g.user.userid != 1:
300
              return redirect(url for('user home screen'))
301
          else:
302
              if request.method == 'POST':
303
                  database helpers.delete user(request.form.get('user id'))
304
              UserQuery = User.query.order by(User.userid).all()
305
              return render template ('admin-view-users.html', UserQuery=UserQuery)
306
307
      # Admin view of postings
308
      @app.route('/admin-view/postings', methods=['GET', 'POST'])
309
      def admin view postings():
310
          if q.user is None:
311
              return redirect(url for('login'))
312
          elif q.user.userid != 1:
313
              return redirect(url for('user home screen'))
314
          else:
              if request.method == 'POST':
315
316
                  database helpers.archivedPost(request.form.get('post id'))
317
                  db.session.commit()
318
              PostingQuery = Posting.query.order by (Posting.postid).all()
319
              return render template ('admin-view-postings.html', PostingQuery=PostingQuery)
320
321
      # Admin view of creating accounts
322
      @app.route('/admin-view/create-account', methods=['GET', 'POST'])
323
      def admin view create account(error=""):
324
          if q.user is None:
325
              return redirect(url for('login'))
326
          elif q.user.userid != 1:
327
              return redirect(url for('user home screen'))
328
          else:
329
              title = "Create a User"
              CREATE ERROR = "Need to fill in ALL fields marked with an '*'"
330
331
              errors = []
332
              if request.method == 'POST':
333
                  [results, errors] = form submissions.generate new account form(request.form)
334
                  if len(errors) == 0:
335
                      added successfully = database helpers.add user(results)
336
                      if added successfully:
```