

```
1  import os
2  basedir = os.path.abspath(os.path.dirname(__file__))
3
4  class Config(object):
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
```

```
1  from app import app, socketio
2
3
4  if __name__ == '__main__':
5      socketio.run(app, debug=True)
6
```

```

1  from app import models, db
2  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'],
17         description = form_input['description'],
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
24
25 def add_user(new_user_info):
26     try:
27         newUser = models.User(
28             username = new_user_info['username'],
29             email = new_user_info['email'],
30             password = new_user_info['password'],
31             phonenumber = new_user_info['phone'],
32             personalemail = new_user_info['personalemail'],
33             bio = new_user_info['bio'],
34             rating = float(new_user_info['rating']),
35             numRatings = int(new_user_info['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':
81         poster_info =
82             models.User.query.filter_by(userid=posting_info.userid).with_entities(
83                 models.User.personalemail, models.User.rating, models.User.userid,
84                 models.User.username
85             ).first()
86     elif posting_info.contactmethod == "phonenumber":
87         poster_info =
88             models.User.query.filter_by(userid=posting_info.userid).with_entities(
89                 models.User.phonenumber, models.User.rating, models.User.userid,
90                 models.User.username
91             ).first()
92     else:
93         poster_info =
94             models.User.query.filter_by(userid=posting_info.userid).with_entities(
95                 models.User.email, models.User.rating, models.User.userid,
96                 models.User.username
97             ).first()
98     return posting_info, poster_info
99
100 def get_user_by_email(email):
101     return models.User.query.filter_by(email=email).first()
102
103 def add_claim(form, postid, user):
104     try:
105         post_info, poster_info = get_posting_by_id(postid)
106         if user.userid == post_info.userid and not form['buyeremail'] == False:
107             buyer_info = get_user_by_email(form['buyeremail'])
108             newClaim = models.Claim(
109                 postid = postid,
110                 sellerid = user.userid,
111                 buyerid = buyer_info.userid,
112                 usersubmitted = user.userid,
113                 date = datetime.now(),
114                 Rating = form['rating']
115             )
116             db.session.add(newClaim)
117             return True, newClaim
118         elif not user.userid == post_info.userid:
119             newClaim = models.Claim(
120                 postid = postid,
121                 sellerid = post_info.userid,
122                 buyerid = user.userid,
123                 usersubmitted = user.userid,
124                 date = datetime.now(),
125                 Rating = form['rating']
126             )
127             db.session.add(newClaim)
128             return True, newClaim
129     except Exception as e:
130         db.session.rollback()
131         return False, False
132     return False, False
133
134 def check_for_transaction(claim):

```

```

129     try:
130         other_claim = models.Claim.query.filter(
131             models.Claim.postid==claim.postid,
132             models.Claim.sellerid==claim.sellerid,
133             models.Claim.buyerid==claim.buyerid,
134             models.Claim.usersubmitted!=claim.usersubmitted
135         ).first()
136         if other_claim is not None:
137             newTransaction = models.Transaction(
138                 date = datetime.now(),
139                 claimidseller = claim.sellerid,
140                 claimidbuyer = claim.buyerid
141             )
142             db.session.add(newTransaction)
143             print("Try to Alter Ratings!")
144             alter_ratings(claim, other_claim)
145             print("Try to Archive!")
146             if archive_posting(claim.postid, newTransaction):
147                 print("Archived! Now Delete The Claims:")
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):
176     try:
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                 buyerid = transaction.claimidbuyer,
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         date      = datetime.now(),
197         title      = post.title,
198         description = post.description,
199         price      = post.price,
200         category   = post.category,
201         contactmethod = post.contactmethod,
202         tags       = 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
214     current_rating = current_rating * (current_number-1)/current_number + new_rating *
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     [user1.rating, user1.numRatings] = get_new_rating(user1.rating, user1.numRatings,
other_claim.Rating)
223     [user2.rating, user2.numRatings] = get_new_rating(user2.rating, user2.numRatings,
claim.Rating)
224     print("FINISHED alter ratings")
225
226 def get_postings(userid):
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
ties(
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:
248         sales = models.ArchivedPosting.query.filter_by(sellerid=userid).with_entities(
models.ArchivedPosting.title, models.ArchivedPosting.buyerid,
models.ArchivedPosting.price,
250         ).join(
models.Transaction).join(models.User, models.User.userid ==
models.ArchivedPosting.buyerid).with_entities(
252             models.Transaction.date, models.ArchivedPosting.title,
models.ArchivedPosting.price,
models.User.username, models.User.userid).all()
253         print("Got sales")
254

```

```
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,
264             models.ArchivedPosting.price,
265             ).join(
266                 models.Transaction).join(models.User, models.User.userid ==
267                 models.ArchivedPosting.sellerid).with_entities(
268                     models.Transaction.date, models.ArchivedPosting.title,
269                     models.ArchivedPosting.price,
270                     models.User.username, models.User.userid).all()
271         print("Got purchases")
272         return purchases
273     except Exception as e:
274         pass
275     return None
```

```

1  import re
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):
6      regex = '^\\w+([\\.-]?\\w+ )*@([\\w+.)]*pitt.edu'
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
16     # 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
40     #             MAY WANT TO EDIT THIS
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 + '%')
56             return return_search
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):
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:
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:
83         field = float(field)
84         if field > 0 and field <= 2000:
85             return str(field)
86         elif field > 2000:
87             return '2000'
88         else:
89             return '0'
90     except Exception as e:
91         return '0'
92
93 def should_randomize(submitted):
94     return submitted == {'minPrice': '0', 'maxPrice': '0', 'search': '', 'category':
95         '', 'page': 1}
96
97 def get_page(field):
98     try:
99         return int(field)
100    except Exception as e:
101        pass
102    return 1
103
104 def get_filters(forms, get_recieved):
105     if get_recieved and forms.get('minPrice') is not None:
106         submitted = {}
107         submitted['minPrice'] = get_min_price(forms['minPrice'])
108         submitted['maxPrice'] = get_max_price(forms['maxPrice'], submitted['minPrice'])
109         submitted['search'] = get_search_text(forms['search'])
110         submitted['category'] = get_category(forms['category'])
111         submitted['page'] = get_page(forms['page']) if 'page' in forms else 1
112         return [submitted, should_randomize(submitted)]
113     else:
114         return [{'minPrice': '0', 'maxPrice': '0', 'search': '', 'category': '',
115             'page': 1}, True]
116
117 ##### New Posting Submission #####
118 # verifies it is 30 chars or shorter
119 def validate_title(field):
120     try:
121         field = str(field)
122         if len(field) > 0 and len(field) <31:
123             return [True, field]
124     except Exception as e:
125         return [False, "The title needs to be 1-30 characters long."]
126     return [False, "The title needs to be 1-30 characters long. Your input was " +
127         str(len(field)) + " characters."]
128
129 # this should never really fail. It is on us if it does
130 def validate_category(field, CATEGORIES):
131     try:
132         field = str(field)
133         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:
142             return [True, field]
143         raise ValueError
144     except ValueError as e:
145         return [False, "Invalid price. Needs to be in the range of 0 to 2,000
146             inclusive."]
147     return [False, "Invalid preffered contact method. Try resubmitting."]
148
149 # ensure description is less than 1001 chars long.
150 def validate_desc(field):
151     try:
152         field = str(field)
153         if len(field) > 1000:
154             return [False, "Your short description cannot be longer than 1,000
155                 characters long."]
156     except Exception as e:
157         return [False, "Invalid short description."]
158     return [True, field]
159
160 # returns preferred contact value; should alk
161 def validate_preferred_contact(field):
162     try:
163         field = str(field)
164         if field == "email" or field == "phonenumber" or field == "personalemail":
165             return [True, field]
166     except Exception as e:
167         return [False, "Invalid preffered contact method. Try resubmitting."]
168     return [False, "Invalid preffered contact method. Try resubmitting."]
169
170 def validate_preferred_tags(field, title):
171     try:
172         title = str(title[1]).split()
173         field = str(field) + ',' + ','.join(title)
174         field = ''.join(field.split()).lower().split(',')
175         if len(field) < 900:
176             if len(field) > 50:
177                 return [False, "Too many tags. The maximum tags are 50. Maximum
178                     character limit is 900"]
179             field = list(set(field))
180             field = ','.join(field)
181             return [True, field]
182         raise ValueError
183     except Exception as e:
184         return [False, "Too many tags. Maximum character limit is 900"]
185
186 def validate_input(forms, CATEGORIES):
187     result = {}
188     result['title'] = validate_title(forms['title'])
189     result['category'] = validate_category(forms['category'], CATEGORIES)
190     result['price'] = validate_price(forms['price'])
191     result['description'] = validate_desc(forms['description'])
192     result['contactmethod'] = validate_preferred_contact(forms['preferredContact'])
193     result['tags'] = validate_preferred_tags(forms['tags'], result['title'])
194     return result
195
196 def generate_return_values(given):
197     error = []
198     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'],
230 forms['password2'])
231     new_account_info['phone'] = validate_phone_number(forms['phonenumber'])
232     new_account_info['personalemail'] = validate_personal_email(forms['personalemail'])
233     new_account_info['bio'] = validate_bio(forms['bio'])
234     new_account_info['rating'] = [True, '5']
235     new_account_info['numRatings'] = [True, '0']
236     return new_account_info
237
238 def validate_email(field):
239     try:
240         field = get_email(str(field))
241         if not field == False and len(field) < 75:
242             return [True, field]
243         raise ValueError
244     except Exception as e:
245         return [False, "The university email must end with a school domain (pitt.edu)."]
246
247 def validate_password(password1, password2):
248     try:
249         password1 = str(password1)
250         password2 = str(password2)
251         if password1 == password2:
252             if len(password1) > 7 and len(password1) < 33:
253                 return [True, generate_password_hash(password1)]
254             raise ValueError
255     except Exception as e:
256         return [False, "Both Password fields must match and have between 8 and 32
257 characters (inclusive)."]
258
259 def convert_number(phone):
260     phone = phone.replace('-', '')
261     phone = phone.replace('(', '')
262     phone = phone.replace(')', '')

```

```

261     if len(phone) == 10:
262         phone = '1' + phone
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 != ''
279         else ''
280         return [True, phone_number]
281     except Exception as e:
282         return [False, "Phone number must be 10 characters long or 11 characters long
283         with the country code being '1' in order to be processed."]
284     return [True, phone_number]
285
286 def validate_personal_email(field):
287     email_regex = '^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*\\.\\w{2,3}+$'
288     try:
289         field = str(field)
290         if re.search(email_regex, field) is not None:
291             return [True, field]
292     except Exception as e:
293         return [False, "The personal email address is not a legal value."]
294     return [True, field]
295
296 def validate_bio(field):
297     try:
298         field = str(field)
299         if len(field) < 251:
300             return [True, field]
301     except Exception as e:
302         return [False, "The biography is unable to be processed. Possibly an invalid
303         symbol."]
304     return [False, "Length exceeds 250 characters"]
305
306 def get_modified_account_info(forms, userid):
307     result = generate_fields_edit_account(forms, userid)
308     return generate_return_values(result)
309
310 def validate_password_simple(password):
311     try:
312         password = str(password)
313         return [True, password]
314     except Exception as e:
315         return [False, "Try Resubmitting information."]
316
317 def validate_delete(forms):
318     try:
319         if forms['deleteaccount'] == 'delete':
320             return [True, 'delete']
321     except Exception as e:
322         pass
323     return [False, "nothing"]
324
325 def generate_fields_edit_account(forms, userid):
326     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'],
328             forms['newpassword2'])
329     new_account_info['oldpassword']      = validate_password_simple(forms['oldpassword'])
330     new_account_info['phonenummer']      = validate_phone_number(forms['phonenummer'])
331     new_account_info['personalemail']    = validate_personal_email(forms['personalemail'])
332     new_account_info['bio']              = validate_bio(forms['bio'])
333     new_account_info['deleteaccount']    = validate_delete(forms)
334     return new_account_info
335
336 #####
337 # Claims
338 def get_new_claims_form(forms, current_user, postid):
339     results = generate_claims_forms(forms, current_user, postid)
340     return generate_return_values(results)
341
342 def generate_claims_forms(forms, current_user, postid):
343     claim_info = {}
344     claim_info['postid']          = [True, postid]
345     claim_info['userid']          = [True, current_user.userid]
346     claim_info['rating']          = validate_rating_claims(forms['rating'])
347     claim_info['buyeremail']      = validate_buyer_email(forms)
348     return claim_info
349
350 def validate_rating_claims(field):
351     try:
352         field = int(field)
353         if field > 0 and field < 6:
354             return [True, field]
355         raise ValueError
356     except Exception as e:
357         return [False, "Please resubmit your claim!"]
358
359 def validate_buyer_email(forms):
360     if 'buyeremail' in forms:
361         try:
362             field = str(forms['buyeremail'])
363             if get_email(field) != False:
364                 return [True, get_email(field)]
365         except Exception as e:
366             pass
367     return [True, False]

```

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

```

1  from flask_sqlalchemy import SQLAlchemy
2  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):
12     list = []
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)
16         for row in reader:
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             userid      = value['userid'],
26             date        = datetime.now(),
27             title       = value['title'],
28             description  = value['description'],
29             price       = value['price'],
30             category    = value['category'],
31             contactmethod = value['contactmethod'],
32             tags        = tags
33         )
34         db.session.add(newPosting)
35         db.session.commit()
36
37
38 def add_users(file):
39     list = []
40     key_list = ['phonenumber', 'email', 'personalemail', 'password', 'bio']
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             username      = value['username'][1],
53             email         = value['email'],
54             personalemail = value['personalemail'],
55             password      = generate_password_hash(value['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')
66 def initdb_command():
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))
75 # 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))
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'))
80 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'))
81 db.session.add(Posting(userid=2, date=datetime.now(), title='Cheap Book',
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     username = db.Column(db.String(24), nullable = False)
91     email = db.Column(db.String(80), unique=True, nullable = False)
92     # hashed password is ~100 chars ALWAYS
93     password = db.Column(db.String(128), nullable = False)
94     phonenumber = db.Column(db.String(64), nullable = False)
95     personalemail = db.Column(db.String(80), nullable = False)
96     bio = db.Column(db.String(250), nullable = False)
97     rating = db.Column(db.Float(2), nullable = False)
98     numRatings = db.Column(db.Integer, nullable = False)
99     postings = relationship("Posting", cascade="all,delete", backref="User")
100
101     def __repr__(self):
102         return '<User {}>'.format(self.username)
103
104
105 class Posting(db.Model):
106     postid = db.Column(db.Integer, primary_key = True)
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)
114     tags = db.Column(db.String(1000), nullable = True)
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):
122     __table_args__ = (
123         db.UniqueConstraint('postid', 'sellerid', 'buyerid', 'usersubmitted',
124                             name='unique_claim_buyer_seller'),
125     )
126     claimid = db.Column(db.Integer, primary_key = True)
127     postid = db.Column(db.Integer, db.ForeignKey("posting.postid"))
128     sellerid = db.Column(db.Integer, db.ForeignKey("user.userid"))
129     buyerid = db.Column(db.Integer, db.ForeignKey("user.userid"))
130     usersubmitted = db.Column(db.Integer, db.ForeignKey("user.userid"))
131     date = db.Column(db.Date, nullable = False)
132     Rating = db.Column(db.Integer, nullable = False)
133
134     def __repr__(self):
135         return '<Claim {}: "{}">'.format(self.claimid)
136
137 class Transaction(db.Model):
138     transactionid = db.Column(db.Integer, primary_key = True)
139     date = db.Column(db.Date, nullable = False)
140     claimidseller = db.Column(db.Integer, db.ForeignKey("claim.claimid"), nullable =
141                               False)
142     claimidbuyer = db.Column(db.Integer, db.ForeignKey("claim.claimid"), nullable =
143                               False)
144
145     def __repr__(self):
146         return '<Transaction {}: "{}">'.format(self.transactionid)
147
148 class ArchivedPosting(db.Model):
149     __table_args__ = (
150         db.UniqueConstraint('postid', 'buyerid', 'archivedpostid', 'sellerid',
151                             name='unique_archive_posting_constraint'),
152     )
153     archivedpostid = db.Column(db.Integer, primary_key = True)
154     transactionid = db.Column(db.Integer, db.ForeignKey('transaction.transactionid'),
155                               nullable = True)
156     postid = db.Column(db.Integer, nullable = False)
157     buyerid = db.Column(db.Integer, db.ForeignKey("user.userid"), nullable =
158                               True)
159     sellerid = db.Column(db.Integer, db.ForeignKey("user.userid"), nullable =
160                               True)
161     date = db.Column(db.Date, nullable = False)
162     title = db.Column(db.String(80), nullable = False)
163     description = db.Column(db.String(250), nullable = True)
164     price = db.Column(db.Integer, nullable = False)
165     category = db.Column(db.String(80), nullable = False)
166     contactmethod = db.Column(db.String(80), nullable = True)
167     tags = db.Column(db.String(1000), nullable = True)
168
169     def __repr__(self):
170         return '<Posting {}: "{}">'.format(self.postid, self.title)

```

```

1  from app import app, helper_functions, socketio, form_submissions, database_helpers
2  from app.models import *
3  from werkzeug.security import generate_password_hash, check_password_hash
4  from flask import redirect, render_template, request, session, url_for, abort, g, flash
5  from random import shuffle
6  from datetime import datetime
7  from sqlalchemy import and_, or_
8
9  CATEGORIES = ['All', 'Textbooks', 'Furniture', 'Food', 'Events', 'Software',
10 'Electronics',
11 'Beauty and Personal Care', 'Clothes', 'School Supplies', 'Appliances']
12 CONTACT_METHOD = {
13     "email": "Email (university provided)",
14     "phonenumber": "Phone Number (if provided)",
15     "personalemail": "Personal Email (if provided)"
16 }
17
18 # forces logout on browser close(); aka senses packets have stopped flowing
19 @socketio.on('disconnect')
20 def disconnect_user():
21     logout()
22
23 #####
24 # ROUTES START HERE
25 #####
26 # Log the user out
27 @app.route('/logout')
28 def logout():
29     session.pop('userid', None)
30     return redirect(url_for('login'))
31
32 # Run at the beginning of each request before functions run to check if logged in
33 @app.before_request
34 def request_authentication():
35     g.user = None
36     if 'userid' in session:
37         g.user = User.query.filter_by(userid=session['userid']).first()
38
39 ### ERROR HANDLING PAGES
40 @app.route('/error')
41 def not_found_error_item():
42     return render_template('404.html'), 404
43
44 @app.errorhandler(404)
45 def not_found_error(error):
46     return render_template('404.html'), 404
47
48 @app.errorhandler(500)
49 def internal_error(error):
50     db.session.rollback()
51     return render_template('500.html'), 500
52
53 @app.route('/')
54 def slash_redirect():
55     return redirect(url_for('login'))
56
57 ##### LOGIN #####
58 # The login screen
59 @app.route('/login', methods=['GET', 'POST'])
60 def login(error=""):
61     title = "Login to Craigiversity!"
62     LOGIN_ERROR = "Invalid information was submitted. Please try again!"
63     error = ''
64     if g.user:
65         return redirect(url_for('user_home_screen'))
66     if request.method == "POST":
67         error = LOGIN_ERROR
68         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']):
70                 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,
73                                page_title=title, css_file=helper_functions.generate_linked_files('login'))
74 ##### ACCOUNT ROUTES #####
75 # The create account screen
76 @app.route('/create-account', methods=['GET', 'POST'])
77 def create_account(error=""):
78     title = "Welcome to Craigiversity!"
79     CREATE_ERROR = "Need to fill in ALL fields marked with an '*'"
80     errors = []
81     if g.user:
82         return redirect(url_for('user_home_screen'))
83     if request.method == 'POST':
84         [results, errors] = form_submissions.generate_new_account_form(request.form)
85         if len(errors) == 0:
86             added_successfully = database_helpers.add_user(results)
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,
91                            error=errors, page_title=title,
92                            css_file=helper_functions.generate_linked_files('create-account'), )
93
94 # The user screen
95 @app.route('/user', methods=['GET', 'POST'])
96 def users_account():
97     if g.user is None:
98         return redirect(url_for('login'))
99     account_info = g.user
100     title = "USER: " + g.user.username
101     if request.method == "GET":
102         userid = (request.args.get('userid'))
103         account_info = User.query.filter_by(userid=userid).first()
104         if account_info is None:
105             return redirect(url_for('not_found_error_item'))
106         postings = database_helpers.get_postings(account_info.userid)
107         claims = database_helpers.get_claims(account_info.userid)
108         sales = database_helpers.get_sales(account_info.userid)
109         purchases = database_helpers.get_purchases(account_info.userid)
110         title = "USER: " + account_info.username
111     return render_template('account-view.html', purchases=purchases, postings=postings,
112                            sales=sales, claims=claims, current_user_id=g.user.userid,
113                            current_user_is_auth=(g.user.userid > 0), user_id=g.user.userid,
114                            CURRENT_USER_ID=g.user.userid, page_title=title,
115                            css_file=helper_functions.generate_linked_files('account-view'),
116                            account=account_info)
117
118 # The edit account screen
119 @app.route('/edit-account', methods=['GET', 'POST'])
120 def edit_account(error=""):
121     if g.user is None:
122         return redirect(url_for('login'))
123     title = 'Edit Account'
124     current_user = g.user
125     error = ''
126     if request.method == 'POST':
127         [result, error] = form_submissions.get_modified_account_info(request.form,
128                                g.user.userid)
129         if len(error) == 0 and check_password_hash(g.user.password,
130            str(result['oldpassword'])):
131             if result['deleteaccount'] == "delete":
132                 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 ##### POSTINGS #####
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:
136         return redirect(url_for('login'))
137     if request.method == 'POST':
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)
141             return redirect(url_for('login'))
142     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'))

143
144 # NEED TO REWORK CONTACT METHOD!!!
145 @app.route('/edit-posting', methods=['GET', 'POST'])
146 def edit_posting(error=""):
147     title = 'Edit Posting'
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:
153         return redirect(url_for('user_home_screen'))
154
155     if request.method == 'POST':
156         [results, error] = form_submissions.get_form_create_post(request.form,
CATEGORIES)
157         if len(error) == 0:
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
164 @app.route('/posting', methods=['GET'])
165 def full_posting_view():
166     if g.user is None:
167         return redirect(url_for('login'))
168     #posting_info = {}
169     if request.method == 'GET':
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'))
173         title = "POSTING: " + posting_info.title
174     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'),

```

```

175     post=posting_info, poster_info=user_info)
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():
180     title = "Search and Filter Postings!"
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         if randomize:
188             postings = database_helpers.generate_random_postings()
189         else:
190             if submitted['search'] == '':
191                 postings = Posting.query.filter(
192                     and_(
193                         Posting.price >= float(submitted['minPrice']),
194                         Posting.price <= float(submitted['maxPrice']),
195
196                         or_(Posting.category.contains(submitted['category']), categoryIsAll)
197                     )
198                 ).join(User).with_entities(
199                     Posting.postid, Posting.userid, User.username,
200                     Posting.title, Posting.price,
201                     Posting.description,
202                     User.rating)
203             else:
204                 search_elems = form_submissions.get_search_elems(submitted['search'])
205                 postings = Posting.query.filter(and_(
206                     and_(Posting.tags.like(e) for e in search_elems),
207                     Posting.price >= float(submitted['minPrice']),
208                     Posting.price <= float(submitted['maxPrice']),
209
210                     or_(Posting.category.contains(submitted['category']), categoryIsAll)
211                 )
212                 ).join(User).with_entities(
213                     Posting.postid, Posting.userid, User.username,
214                     Posting.title, Posting.price,
215                     Posting.description,
216                     User.rating)
217             else:
218                 submitted = form_submissions.get_filters('', True)
219                 postings = database_helpers.generate_random_postings()
220
221     page = form_submissions.get_page_number(submitted['page'], postings.count(),
222     app.config['POSTS_PER_PAGE'])
223     postings = postings.paginate(page, app.config['POSTS_PER_PAGE'], False)
224     next_url = url_for('user_home_screen', search=submitted['search'],
225     category=submitted['category'], minPrice=submitted['minPrice'],
226     maxPrice=submitted['maxPrice'], page=postings.next_num) if postings.has_next else
227     None
228     prev_url = url_for('user_home_screen', search=submitted['search'],
229     category=submitted['category'], minPrice=submitted['minPrice'],
230     maxPrice=submitted['maxPrice'], page=postings.prev_num) if postings.has_prev else
231     None
232     return render_template('user-view.html', next_url=next_url, prev_url=prev_url,
233     page=page, categories=CATEGORIES, able_to_filter=True, submitted=submitted,
234     current_user_id=g.user.userid, current_user_is_auth=(g.user.userid > 0),
235     page_title=title, css_file=helper_functions.generate_linked_files('user-view'),
236     filtered_postings=postings.items)
237 ##### CLAIMS #####

```

```

224 # the claim pages
225 @app.route('/claim', methods=['GET', 'POST'])
226 def claim_submission():
227     if g.user is None:
228         return redirect(url_for('login'))
229     title = "Claim Submission"
230     error = ''
231     IsSeller = False
232     post_info = {'title': '', 'postid': '', 'username': '' }
233     try:
234         [posting_info, poster_info] =
235         database_helpers.get_posting_by_id(request.args.get('postid'))
236         if posting_info is None:
237             return redirect(url_for('error'))
238         post_info = helper_functions.get_post_info_claims(posting_info, poster_info)
239         isSeller = (posting_info.userid == g.user.userid)
240         if isSeller:
241             title = "Seller " + title
242         else:
243             title = "Buyer " + title
244     except Exception as e:
245         error = "Please Try Resubmitting. Something went Wrong."
246         return render_template('claim.html', error=error, post=post_info,
247                                isSeller=False, current_user_id=g.user.userid,
248                                current_user_is_auth=(g.user.userid > 0), page_title=title,
249                                css_file=helper_functions.generate_linked_files('claim') )
250
251 if request.method == 'POST':
252     [submitted, error] = form_submissions.get_new_claims_form(request.form, g.user,
253                                                                post_info['postid'])
254     if len(error) == 0:
255         [completed_claim, claim] = database_helpers.add_claim(submitted,
256                                                                post_info['postid'], g.user)
257         if completed_claim:
258             is_transaction_completed = database_helpers.check_for_transaction(claim)
259             if is_transaction_completed:
260                 db.session.commit()
261                 return redirect(url_for('claim_completion',
262                                         is_transaction_complete=True ))
263             elif is_transaction_completed is not None:
264                 db.session.commit()
265                 return redirect(url_for('claim_completion',
266                                         is_transaction_complete=False ))
267         error = "Please resubmit your claim. There was an issue. You may have
268         entered invalid data."
269     return render_template('claim.html', error=error, post=post_info,
270                           isSeller=isSeller, current_user_id=g.user.userid,
271                           current_user_is_auth=(g.user.userid > 0), page_title=title,
272                           css_file=helper_functions.generate_linked_files('claim') )
273
274 @app.route('/claim-complete', methods=['GET', 'POST'])
275 def claim_completion(is_transaction_complete=False):
276     if g.user is None:
277         return redirect(url_for('login'))
278     is_transaction_complete = request.args.get('is_transaction_complete')
279     print(is_transaction_complete)
280     return render_template('claim-complete.html',
281                           is_transaction_complete=is_transaction_complete=="True", title="Claim Complete",
282                           error='', current_user_id=g.user.userid, current_user_is_auth=(g.user.userid > 0),
283                           css_file=helper_functions.generate_linked_files('claim'))
284
285 # The HELP page
286 @app.route('/help-and-FAQ')
287 def help():
288     return render_template('help.html')

```

```

276 @app.route('/remove-posting', methods=['GET', 'POST'])
277 def remove_posting_view(error=""):
278     title = 'Remove Posting'
279     if g.user is None:
280         return redirect(url_for('login'))
281
282     posting_info = database_helpers.get_post_id(request.args.get('postid'))
283     if posting_info is None or g.user.userid != posting_info.userid:
284         return redirect(url_for('user_home_screen'))
285
286     if request.method == 'POST':
287         posting_info_remove =
288         Posting.query.filter_by(postid=request.args.get('postid')).first()
289         db.session.delete(posting_info_remove)
290         db.session.commit()
291         return redirect(url_for('user_home_screen'))
292
293     return render_template('remove-posting-view.html', contact_options=CONTACT_METHOD,
294     categories=CATEGORIES, js_file="tag-javascript.js", current_user_id=g.user.userid,
295     current_user_is_auth=(g.user.userid > 0), user_id=g.user.userid,
296     CURRENT_USER_ID=g.user.userid, page_title=title, error=error,
297     css_file=helper_functions.generate_linked_files('create-posting-view'),
298     post=posting_info)
299
300 # Admin view of users
301 @app.route('/admin-view/users', methods=['GET', 'POST'])
302 def admin_view_users():
303     if g.user is None:
304         return redirect(url_for('login'))
305     elif g.user.userid != 1:
306         return redirect(url_for('user_home_screen'))
307     else:
308         if request.method == 'POST':
309             database_helpers.delete_user(request.form.get('user_id'))
310             UserQuery = User.query.order_by(User.userid).all()
311             return render_template('admin-view-users.html', UserQuery=UserQuery)
312
313 # Admin view of postings
314 @app.route('/admin-view/postings', methods=['GET', 'POST'])
315 def admin_view_postings():
316     if g.user is None:
317         return redirect(url_for('login'))
318     elif g.user.userid != 1:
319         return redirect(url_for('user_home_screen'))
320     else:
321         if request.method == 'POST':
322             database_helpers.archivedPost(request.form.get('post_id'))
323             db.session.commit()
324             PostingQuery = Posting.query.order_by(Posting.postid).all()
325             return render_template('admin-view-postings.html', PostingQuery=PostingQuery)
326
327 # Admin view of creating accounts
328 @app.route('/admin-view/create-account', methods=['GET', 'POST'])
329 def admin_view_create_account(error=""):
330     if g.user is None:
331         return redirect(url_for('login'))
332     elif g.user.userid != 1:
333         return redirect(url_for('user_home_screen'))
334     else:
335         title = "Create a User"
336         CREATE_ERROR = "Need to fill in ALL fields marked with an '*'"
337         errors = []
338         if request.method == 'POST':
339             [results, errors] = form_submissions.generate_new_account_form(request.form)
340             if len(errors) == 0:
341                 added_successfully = database_helpers.add_user(results)
342                 if added_successfully:

```

```
337         return redirect(url_for('admin_view_users'))
338         errors = "Could not process. Try again."
339     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'), )
340
```