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

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

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



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

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

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

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

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

```
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':
        '', '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:
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:
130         field = str(field)
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:
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
                inclusive."]
146         return [False, "Invalid preffered contact method. Try resubmitting."]
147 
148     # ensure description is less than 1001 chars long.
149     def validate_desc(field):
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:
155             return [False, "Invalid short description."]
156         return [True, field]
157 
158     # returns preferred contact value; should alk
159     def validate_preferred_contact(field):
160         try:
161             field = str(field)
162             if field == "email" or field == "phonenumber" or field == "personalemail":
163                 return [True, field]
164         except Exception as e:
165             return [False, "Invalid preffered contact method. Try resubmitting."]
166         return [False, "Invalid preffered contact method. Try resubmitting."]
167 
168     def validate_preferred_tags(field, title):
169         try:
170             title = str(title[1]).split()
171             field = str(field) + ',' + ','.join(title)
172             field = ''.join(field.split()).lower().split(',')
173             if len(field) < 900:
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:
181             return [False, "Too many tags. Maximum character limit is 900"]
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'])
188         result['description'] = validate_desc(forms['description'])
189         result['contactmethod'] = validate_preferred_contact(forms['preferredContact'])
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'])
232         new_account_info['bio']             = validate_bio(forms['bio'])
233         new_account_info['rating']          = [True, '5']
234         new_account_info['numRatings']      = [True, '0']
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:
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:
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):
258         phone = phone.replace('-', '')
259         phone = phone.replace('(', '')
260         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 != ''
                else ''
279             return [True, phone_number]
280         except Exception as e:
281             return [False, "Phone number must be 10 characters long or 11 characters long
                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:
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':
318                 return [True, 'delete']
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'])
331         new_account_info['bio']             = 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
341     def generate_claims_forms(forms, current_user, postid):
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:
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
```