Coverage for **my-modules/aspire360_measures/controllers/controllers.py** : 87%

191 statements | 167 run | 24 missing | 0 excluded

```
1  # -*- coding: utf-8 -*-
2  from odoo import http
3  from odoo.http import request
4  from socket import *
5  import base64
6  import time
7
8  # Validation parameter. Turn off when developing and on when testing
9  VALIDATION = False
10
11 class Aspire360(http.Controller):
12     @http.route('/aspire360measures/', auth='public',website=True)
13     def index(self, **kw):
14         print("HELLO IN CONTROLLER")
15         print("HELLO IN CONTROLLER")
16         entrepreneurs = http.request.env['aspire360.entrepreneurs'].search([('user_id', '=', request.env.context.get ('uid')
17         venture_capitalists = http.request.env['aspire360.venturecapitalists'].search([('user_id', '=', request.env.context.
18         if not self.is_entrepreneur() and not self.is_venturecapitalist():
19             return http.request.redirect('/aspire360measures/setup')
20         elif self.is_venturecapitalist():
21             return http.request.render('aspire360_measures.v_index')
22         else:
23             return http.request.render('aspire360_measures.e_index')
24         # print("Entrepreneurs: ", entrepreneurs)
25         # print("Entrepreneurs: ", venture_capitalists)
26         # print("User uid: ", request.env.user)
27         # print("User uid: ",request.env.context)
28         # print("User uid: ", request.env.context.get ('uid'))
29         # print("Context: ", http.request.env['ir.config_parameter'].sudo().get_param('web.base.url'))
30         # If user doesn't exist in either, redirect to create page to get them to get them to decide whether entrepreneur o
31
32
33     @http.route('/aspire360measures/setup', auth='public',website=True)
34     def setup(self, **kw):
35         #Security measure to prevent someone from signing up twice
36         if VALIDATION:
37             if self.is_entrepreneur() or self.is_venturecapitalist():
38                 return http.request.redirect('/aspire360measures')
39         return http.request.render('aspire360_measures.setup')
40
41     @http.route('/aspire360measures/setup/v', auth='public',website=True)
42     def setup_v(self, **kw):
43         #Security measure to prevent someone from signing up twice
44         if VALIDATION:
45             if self.is_entrepreneur() or self.is_venturecapitalist():
46                 return http.request.redirect('/aspire360measures')
47             else:
48                 # Call respective model functions to create new user
49                 venture_capitalists = http.request.env['aspire360.venturecapitalists']
50                 new_record = {'name':request.env.user.name,
51                               'user_id':request.env.context.get ('uid')}
52                 venture_capitalists.create(new_record)
53         return http.request.redirect('/aspire360measures')
54
55
56     @http.route('/aspire360measures/setup/e', auth='public',website=True)
57     def setup_e(self, **kw):
58         #Security measure to prevent someone from signing up twice
59         if VALIDATION:
60             if self.is_entrepreneur() or self.is_venturecapitalist():
61                 return http.request.redirect('/aspire360measures')
62             else:
63                 # Call respective model functions to create new user
64                 entrepreneurs = http.request.env['aspire360.entrepreneurs']
65                 new_record = {'name':request.env.user.name,
66                               'user_id':request.env.context.get ('uid')}
67                 entrepreneurs.create(new_record)
68         return http.request.redirect('/aspire360measures')
69
70     @http.route('/aspire360measures/email', auth='public',website=True)
71     def setup_email(self, **kw):
```

```
72          #Security measure to prevent someone from signing up twice
73          return http.request.render('aspire360_measures.email_form')
74
75      def email_helper(self, rec, fro, msg, subj):
76          endmsg = "\r\n.\r\n"
77          mailserver = ("smtp.mailtrap.io", 2525)
78          clientSocket = socket(AF_INET, SOCK_STREAM)
79          clientSocket.connect(mailserver)
80          clientSocket.settimeout(None)
81          recv = clientSocket.recv(1024)
82          recv = recv.decode()
83          print(recv)
84          if recv[:3] != '220':
85              print('reply not received from server.')
86          heloCommand = 'EHLO Alice\r\n'
87          clientSocket.send(heloCommand.encode())
88          recv1 = clientSocket.recv(1024)
89          recv1 = recv1.decode()
90          print("1: " + recv1)
91          if recv1[:3] != '250':
92              print('250 reply not received from server.')
93
94          #Info for username and password
95          username = "404819e151fafc"
96          password = "3f6871122cd6fa"
97          Authentication = 'AUTH LOGIN\r\n'
98          clientSocket.send(Authentication.encode())
99          recv = clientSocket.recv(1024)
100         uname = base64.b64encode(username.encode()) + b'\r\n'
101         clientSocket.send(uname)
102         recv = clientSocket.recv(1024)
103         uname = base64.b64encode(password.encode()) + b'\r\n'
104         clientSocket.send(uname)
105         recv = clientSocket.recv(1024)
106
107         command = "MAIL FROM:<" + fro + ">\r\n"
108         clientSocket.send(str.encode(command))
109         recv2 = clientSocket.recv(1024)
110         recv2 = recv2.decode()
111
112         command = "RCPT TO:<" + rec + ">\r\n"
113         clientSocket.send(str.encode(command))
114         recv3 = clientSocket.recv(1024)
115         recv3 = recv3.decode()
116
117         command = "DATA\r\n"
118         clientSocket.send(str.encode(command))
119         recv4 = clientSocket.recv(1024)
120         recv4 = recv4.decode()
121
122         command = "Subject: " + subj + "\r\n\r\n"
123         clientSocket.send(str.encode(command))
124         date = time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime())
125         date = date + "\r\n\r\n"
126         clientSocket.send(str.encode(date))
127         clientSocket.send(str.encode(msg))
128         clientSocket.send(str.encode(endmsg))
129         recv_msg = clientSocket.recv(1024)
130
131         quit = "QUIT\r\n"
132         clientSocket.send(str.encode(quit))
133         recv5 = clientSocket.recv(1024)
134         print(recv5)
135         clientSocket.close()
136
137     @http.route('/aspire360measures/submit_email', auth='public',website=True, csrf=False)
138     def submit_email(self, **kw):
139         print("Params are: {}".format(kw))
140         # print("company name: ", kw["company_info"])
141
142         # print("company industry: ", kw["company_industry"])
143         # print("company employee: ", kw["company_employees"])
144         # print("company funding: ", kw["company_funding_stage"])
145         #return http.request.render('aspire360_measures.e_index')
146         msg = ""
147         if kw["email_template"] == "Follow-up":
148             msg = "This is a follow-up from the investor. This is a test message for development"
```

```
149         else:
150             msg = "This is an introduction from the investor. This is a test message for development"
151         self.email_helper(kw["email_recipient"], kw["email_sender"], msg, kw["email_template"])
152 
153     @http.route('/aspire360measures/survey/fundraise', auth='public', website=True)
154     def survey_1(self):
155         if VALIDATION and not self.is_entrepreneur():
156             return http.request.redirect('/aspire360measures')
157         surveys = http.request.env['survey.survey'].search([('title', '=', 'Readiness to Fundraise Assessment')])
158         end_url = ''
159         for survey in surveys:
160             user_inputs = survey._create_answer(user=request.env.user)
161             user_session = user_inputs.search([])[-1]
162             # print("Num user sessions is: ", len(user_inputs.search([])))
163             # print("Generated Access token is: ", user_session.access_token)
164             # print("Generated Access token is: ", user_session.get_start_url())
165             # print("Updating survey: ", user_session.survey_id)
166             # print(" with user_id: ", request.env.context.get ('uid'))
167             user_session.update_entrepreneur(user_session.access_token, request.env.context.get ('uid'), "fundraise")
168             end_url = user_session.get_start_url()
169             # print("Updated record, now to test if it is actually stored")
170             # # Test
171         # #Check to see if record is updated:
172         # updated_records = http.request.env['survey.user_input'].search([('aspire_entrepreneur', '=', request.env.context.g
173         # for record in updated_records:
174         #     print("Survey being tested is: ", record.access_token)
175         survey_url = http.request.env['ir.config_parameter'].sudo().get_param('web.base.url') + end_url
176         return http.request.redirect(survey_url)
177 
178     @http.route('/aspire360measures/survey/sell', auth='public', website=True)
179     def survey_2(self):
180         if VALIDATION and not self.is_entrepreneur():
181             return http.request.redirect('/aspire360measures')
182         surveys = http.request.env['survey.survey'].search([('title', '=', 'Readiness to Sell Assessment')])
183         end_url = ''
184         for survey in surveys:
185             user_inputs = survey._create_answer(user=request.env.user)
186             # print("User id is: ", request.env.user)
187             user_session = user_inputs.search([])[-1]
188             # print("Num user sessions is: ", len(user_inputs.search([])))
189             print("Generated Access token is: ", user_session.access_token)
190             # print("Generated Answer token is: ", user_session.answer_token)
191             # print("Generated Access token is: ", user_session.get_start_url())
192             user_session.update_entrepreneur(user_session.access_token, request.env.context.get ('uid'),"sell")
193             end_url = user_session.get_start_url()
194         survey_url = http.request.env['ir.config_parameter'].sudo().get_param('web.base.url') + end_url
195         return http.request.redirect(survey_url)
196 
197     @http.route('/aspire360measures/entrepreneur_edit_profile', auth='public',website=True)
198     def entrepreneur_edit_profile(self, **kw):
199         #Validate current user is entrepreneur
200         if VALIDATION and not self.is_entrepreneur():
201             return http.request.redirect('/aspire360measures')
202         return http.request.render('aspire360_measures.entrepreneur_edit_profile')
203 
204     @http.route('/aspire360measures/submit_info', auth='public',website=True, csrf=False)
205     def submit_info(self, **kw):
206         if VALIDATION and not self.is_entrepreneur():
207             return http.request.redirect('/aspire360measures')
208         entrepreneurs = http.request.env['aspire360.entrepreneurs']
209         entrepreneurs.edit_profile(kw, request.env.context.get ('uid'))
210         print("Params are: {}".format(kw))
211         # print("company name: ", kw["company_info"])
212 
213         # print("company industry: ", kw["company_industry"])
214         # print("company employee: ", kw["company_employees"])
215         # print("company funding: ", kw["company_funding_stage"])
216         return http.request.render('aspire360_measures.e_index')
217 
218     @http.route('/aspire360measures/search', auth='public',website=True, csrf=False)
219     def search(self, **kw):
220         if VALIDATION and not self.is_venturecapitalist():
221             return http.request.redirect('/aspire360measures')
222         #Validate current user is venture capitalisst
223         # venture_capitalists = http.request.env['aspire360.venturecapitalists'].search([('user_id', '=', request.env.contex
224         # if len(venture_capitalists) == 0:
225         #     return http.request.redirect('/aspire360measures')
```

```
226         print("Params are: {}".format(kw))
227         #TODO: UPDATE SEARCH FUNCTION BASED ON PARAMS
228         params = list()
229         if "company_name" in kw and kw["company_name"] != "":
230             params.append(("company_name","=",kw["company_name"]))
231         if "industry" in kw and kw["industry"] != "All":
232             params.append(("company_industry","=",kw["industry"]))
233         if "employees" in kw and kw["employees"] != "All":
234             params.append(("company_size","=",kw["employees"]))
235         if "funding_stage" in kw and kw["funding_stage"] != "All":
236             params.append(("company_funding","=",kw["funding_stage"]))
237         entrepreneurs = http.request.env['aspire360.entrepreneurs'].search(params)
238         print("Num entries: ", len(entrepreneurs))
239         return http.request.render('aspire360_measures.search',{
240             'companies': entrepreneurs
241         })
242 
243     @http.route('/aspire360measures/display_fundraise', auth='public',website=True, csrf=False)
244     def display_fundraise(self, **kw):
245         if VALIDATION and not self.is_venturecapitalist():
246             return http.request.redirect('/aspire360measures')
247         print("Params for display_fundraise are: {}".format(kw))
248         survey = http.request.env['survey.survey'].search([('title', '=', 'Readiness to Fundraise Assessment')])[0]
249         survey_id = survey["access_token"]
250 
251         latest_survey_entry = http.request.env['survey.user_input'].search([("aspire_entrepreneur","=", int(kw["user_id"])),
252                                                                             ("state","=","done"),
253                                                                             ("aspire_type","=","fundraise")])
254         print("User_id is: ", kw["user_id"])
255         print("Num Results is:", len(latest_survey_entry))
256         if len(latest_survey_entry) > 0:
257             latest_survey_token = latest_survey_entry[-1]["access_token"]
258             survey_results_url = http.request.env['ir.config_parameter'].sudo().get_param('web.base.url') + "/survey/print/"
259             # print("Num entries: ", len(entrepreneurs))
260             print("link to survey is: ", survey_results_url)
261             return http.request.redirect(survey_results_url)
262         else:
263             return http.request.render("aspire360_measures.survey_error")
264 
265     @http.route('/aspire360measures/display_sell', auth='public',website=True, csrf=False)
266     def display_sell(self, **kw):
267         if VALIDATION and not self.is_venturecapitalist():
268             return http.request.redirect('/aspire360measures')
269         print("Params for display_sell are: {}".format(kw))
270         survey = http.request.env['survey.survey'].search([('title', '=', 'Readiness to Sell Assessment'),])[0]
271         survey_id = survey["access_token"]
272 
273         latest_survey_entry = http.request.env['survey.user_input'].search([("aspire_entrepreneur","=", int(kw["user_id"])),
274                                                                             ("state","=","done"),
275                                                                             ("aspire_type","=","sell")])
276         if len(latest_survey_entry) > 0:
277             latest_survey_token = latest_survey_entry[-1]["access_token"]
278             survey_results_url = http.request.env['ir.config_parameter'].sudo().get_param('web.base.url') + "/survey/print/"
279             # print("Num entries: ", len(entrepreneurs))
280             print("link to survey is: ", survey_results_url)
281             return http.request.redirect(survey_results_url)
282         else:
283             return http.request.render("aspire360_measures.survey_error")
284 
285     """ --- Helper functions --- """
286     # Validation helpers
287     def is_entrepreneur(self):
288         entrepreneurs = http.request.env['aspire360.entrepreneurs'].search([('user_id', '=', request.env.context.get ('uid')
289         if len(entrepreneurs) > 0:
290             return True
291         else:
292             return False
293 
294     def is_venturecapitalist(self):
295         venture_capitalists = http.request.env['aspire360.venturecapitalists'].search([('user_id', '=', request.env.context.
296         if len(venture_capitalists) > 0:
297             return True
298         else:
299             return False
300 
301 
302     # @http.route('/academy/teacher/<model("academy.teachers"):teacher>/', auth='public', website=True)
```

```
303     # def teacher(self, teacher):
304     #     return http.request.render('academy.biography', {
305     #         'person': teacher
306     #     })
307
308 # class Aspire360Measures(http.Controller):
309 #     @http.route('/aspire360_measures/aspire360_measures/', auth='public')
310 #     def index(self, **kw):
311 #         return "Hello, world"
312
313 #     @http.route('/aspire360_measures/aspire360_measures/objects/', auth='public')
314 #     def list(self, **kw):
315 #         return http.request.render('aspire360_measures.listing', {
316 #             'root': '/aspire360_measures/aspire360_measures',
317 #             'objects': http.request.env['aspire360_measures.aspire360_measures'].search([]),
318 #         })
319
320 #     @http.route('/aspire360_measures/aspire360_measures/objects/<model("aspire360_measures.aspire360_measures"):obj>/', au
321 #     def object(self, obj, **kw):
322 #         return http.request.render('aspire360_measures.object', {
323 #             'object': obj
324 #         })
```