Coverage for **django/learning_base/models.py** : 95%

149 statements | 141 run | 8 missing | 0 excluded

```
"""
This module contains all database models not provided by django
itself.
:author: Claas Voelcker
"""

from hashlib import sha512
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone
from polymorphic.models import PolymorphicModel

from .default_picture import default_picture


class Profile(models.Model):
    """
    A user profile that stores additional information about a user
    :author: Claas Voelcker
    """

    class Meta:
        ordering = ('ranking',)

    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
    )

    birth_date = models.DateField(
        blank=True,
        null=True,
    )

    last_modrequest = models.DateField(
        blank=True,
        null=True,
    )

    language = models.CharField(
        verbose_name='Language',
        max_length=2,
        default="en"
    )

    avatar = models.TextField(
        verbose_name="Avatar of the User",
        default=default_picture,
        null=True,
        blank=True,
    )

    ranking = models.IntegerField(
        default=0
    )

    def get_link_to_profile(self):
        """
        :return: the link to the users profile page
```

```
60          """
61          return "clonecademy.net/admin/profiles/{}/".format(self.user.id)
62
63      def modrequest_allowed(self):
64          """
65          :return: True if the user is allowed to request moderator rights
66          """
67          return (not self.is_mod()
68                  and (self.last_modrequest is None
69                       or (timezone.now() - self.last_modrequest).days >= 7))
70
71      def is_mod(self):
72          """
73          :return: True if the user is in the group moderators
74          """
75          return self.user.groups.filter(name="moderator").exists()
76
77      def is_admin(self):
78          """
79          Returns True if the user is in the group admin
80          :return: whether the user belong to the admin group
81          """
82          return self.user.groups.filter(name="admin").exists()
83
84      def get_hash(self):
85          """
86          calculates a hash to get anonymous user data
87          :return: the first 10 digits of the hash
88          """
89          return sha512(str.encode(self.user.username)).hexdigest()[:10]
90
91      def __str__(self):
92          return str(self.user)
93
94
95  class CourseCategory(models.Model):
96      """
97      The type of a course, meaning the field in which the course belongs, e.g.
98      biochemistry, cloning, technical details.
99      """
100     name = models.CharField(
101         help_text="Name of the category (e.g. biochemistry)",
102         max_length=144,
103         unique=True,
104     )
105
106     color = models.CharField(
107         help_text="Color that is used in the category context",
108         max_length=7,
109         default="#000000"
110     )
111
112     def __str__(self):
113         return self.name
114
115
116 class Course(models.Model):
117     """
118     One course is a group of questions which build on each other and should be
119     solved together. These questions should have similar topics, difficulty
120     and should form a compete unit for learning.
121     :author: Claas Voelcker
122     """
123
124     class Meta:
```

```
125         unique_together = ['category', 'name']
126 
127     # difficulty selection and mapping to human readable names
128     EASY = 0
129     MODERATE = 1
130     DIFFICULT = 2
131     EXPERT = 3
132     DIFFICULTY = (
133         (EASY, 'Easy (high school students)'),
134         (MODERATE, 'Moderate (college entry)'),
135         (DIFFICULT, 'Difficult (college students'),
136         (EXPERT, 'Expert (college graduates)')
137     )
138 
139     # language selection and mapping
140     GER = 'de'
141     ENG = 'en'
142     LANGUAGES = (
143         (GER, 'German/Deutsch'),
144         (ENG, 'English')
145     )
146 
147     # the name of the course
148     name = models.CharField(
149         verbose_name='Course name',
150         help_text="A short concise name for the course",
151         max_length=144
152     )
153 
154     # foreign key mapping to the CourseCategory object
155     category = models.ForeignKey(
156         CourseCategory,
157         null=True,
158         blank=True
159     )
160 
161     # choice field mapped to dictionary above
162     difficulty = models.IntegerField(
163         verbose_name='Course difficulty',
164         choices=DIFFICULTY,
165         default=MODERATE
166     )
167 
168     # choice field mapped to dictionary above
169     language = models.CharField(
170         verbose_name='Course Language',
171         max_length=2,
172         choices=LANGUAGES,
173         default=ENG
174     )
175 
176     # foreign key mapping to the user who can edit the course
177     responsible_mod = models.ForeignKey(
178         User,
179         on_delete=models.SET_NULL,
180         null=True,
181         blank=True
182     )
183 
184     # should the course be serialized for normal users
185     is_visible = models.BooleanField(
186         verbose_name='Is the course visible',
187         default=False
188     )
189 
```

```
    # a short description of the course
    description = models.CharField(
        max_length=144,
        null=True,
        blank=True,
        default=""
    )

    def __str__(self):
        return self.name

    def num_of_modules(self):
        """
        Returns the number of modules
        """
        return len(Module.objects.filter(course=self))


class Module(models.Model):
    """
    A Course is made out of several modules and a module contains the questions
    """

    class Meta:
        unique_together = ['order', 'course']
        ordering = ['order']

    name = models.CharField(
        help_text="A short concise name for the module",
        verbose_name='Module name',
        max_length=144
    )

    learning_text = models.TextField(
        help_text="The learning Text for the module",
        verbose_name="Learning text"
    )

    course = models.ForeignKey(
        Course,
        on_delete=models.CASCADE
    )

    order = models.IntegerField()

    description = models.CharField(
        max_length=144,
        null=True,
        blank=True
    )

    def __str__(self):
        return self.name

    def num_of_questions(self):
        """
        Returns the number of questions in the module
        """
        return len(self.question_set.all())

    def get_previous_in_order(self):
        """
        Gets the previous module in the ordering
        :return: the previous module in the same course
        """
```

```
255         modules = self.course.module_set.all()
256         if list(modules).index(self) <= 0:
257             return False
258         return modules[list(modules).index(self) - 1]
259 
260     def is_first_module(self):
261         """
262         checks whether the given module is the first in a course
263         :return: True, iff this module has the lowest order in the course
264         """
265         modules = self.course.module_set
266         return self == modules.first()
267 
268     def is_last_module(self):
269         """
270         Returns True if this is the final module in a course
271         """
272         modules = self.course.module_set
273         return self == modules.last()
274 
275 
276 class Question(PolymorphicModel):
277     """
278     A question is the smallest unit of the learning process. A question has a
279     task that can be solved by a user, a correct solution to evaluate the
280     answer and a way to provide feedback to the user.
281 
282     :author: Claas Voelcker
283     """
284 
285     class Meta:
286         unique_together = ['module', 'order']
287         ordering = ['module', 'order']
288 
289     # a title for the question
290     title = models.TextField(
291         verbose_name='Question title',
292         help_text="A short and concise name for the question",
293         blank=True,
294         null=True
295     )
296 
297     # the question text, that provides additional information to the user
298     text = models.TextField(
299         verbose_name='Question text',
300         help_text="This field can contain markdown syntax"
301     )
302 
303     # the specific question
304     question = models.TextField(
305         verbose_name='Question',
306         help_text="This field can contain markdown syntax",
307         blank=True,
308         null=True
309     )
310 
311     # a custom feedback that can be displayed
312     feedback = models.TextField(
313         verbose_name="feedback",
314         help_text="The feedback for the user after a sucessful answer",
315         blank=True,
316         null=True
317     )
318 
319     # the ordering attribute of the question (needs to be explicitly saved)
```

```
320     order = models.IntegerField()
321
322     # foreign key mapping to the module that contains this question
323     module = models.ForeignKey(
324         Module,
325         verbose_name="Module",
326         help_text="The corresponding module for the question",
327         on_delete=models.CASCADE
328     )
329
330     def is_first_question(self):
331         """
332         Checks whether this is the first question in the module
333
334         :author: Claas Voelcker
335         :return: whether this is the first question or not
336         """
337         questions = self.module.question_set
338         return self == questions.first()
339
340     def is_last_question(self):
341         """
342         Checks whether this is the last question in the module
343
344         :author: Claas Voelcker
345         :return: whether this is the last question or not
346         """
347         questions = self.module.question_set
348         return self == questions.last()
349
350     def get_previous_in_order(self):
351         """
352         Returns the previous question in the course
353         :author: Claas Voelcker
354         :return: the previous question in the same module
355         """
356         questions = self.module.question_set.all()
357         if list(questions).index(self) <= 0:
358             return False
359         return questions[list(questions).index(self) - 1]
360
361     def get_points(self):
362         """
363         Returns the number of ranking points for the question.
364         This method needs to be overridden by subclasses and
365         remains unimplemented here.
366         :author: Claas Voelcker
367         :return: the points
368         :raise: not implemented error
369         """
370         raise NotImplementedError
371
372     def __str__(self):
373         return self.title
374
375
376 class QuizQuestion(models.Model):
377     """
378     single Quiz Question with possible multiple answers
379     @author Leonhard Wiedmann
380     """
381     question = models.TextField(
382         verbose_name="quizQuestion",
383         help_text="The Question of this quiz question.",
384         default=""
```

```
385     )
386
387     image = models.TextField(
388         help_text="The image which is shown in this quiz",
389         default="",
390         blank=True
391     )
392
393     course = models.ForeignKey(
394         Course,
395         help_text="The Course of this question",
396         on_delete=models.CASCADE
397     )
398
399     def evaluate(self, data):
400         """
401         Checks whether the quiz question is answered correctly
402         :return: True iff all and only the correct answers are
403                  provided
404         """
405         answers = self.answer_set()
406         for ans in answers:
407             if ans.correct:
408                 for i in data['answers']:
409                     if 'id' in i and (i['id'] == ans.id and not i['chosen']):
410                         return False
411             if not ans.correct:
412                 for i in data:
413                     if 'id' in i and (i['id'] == ans.id and i['chosen']):
414                         return False
415         return True
416
417     def answer_set(self):
418         """
419         shortcut for all answers to a question
420         :return: all answers to the quizquestion
421         """
422         return self.quizanswer_set.all()
423
424     def is_solvable(self):
425         """
426         x
427         :return:
428         """
429         for ans in self.answer_set():
430             if ans.correct:
431                 return True
432         return False
433
434     def get_points(self):
435         """
436         returns the points for answering this question type
437         :return: 0 points
438         """
439         return 0
440
441
442 class QuizAnswer(models.Model):
443     """
444     Quiz answer with image and the value for correct answer
445     @author Leonhard Wiedmann
446     """
447     text = models.TextField(
448         help_text="The answer text"
449     )
```

```
450
451     img = models.TextField(
452         help_text="The image for this answer",
453         default="",
454         blank=True
455     )
456
457     correct = models.BooleanField(
458         help_text="If this answer is correct",
459         default=False
460     )
461
462     quiz = models.ForeignKey(QuizQuestion, on_delete=models.CASCADE)
463
464
465 class LearningGroup(models.Model):
466     """
467     A user group (currently not used)
468     """
469     name = models.CharField(
470         help_text="The name of the user group",
471         max_length=144)
472
473     def __str__(self):
474         return self.name
475
476
477 class Try(models.Model):
478     """
479     A try represents a submission of an answer. Each time an answer is
480     submitted, a Try object is created in the database, detailing answer,
481     whether it was answered correctly and the time of the submission.
482     :author: Claas Voelcker
483     """
484     user = models.ForeignKey(
485         User,
486         on_delete=models.SET_NULL,
487         null=True,
488     )
489
490     question = models.ForeignKey(
491         Question,
492         null=True,
493         on_delete=models.SET_NULL,
494     )
495
496     quiz_question = models.ForeignKey(
497         QuizQuestion,
498         null=True,
499         on_delete=models.SET_NULL,
500     )
501
502     answer = models.TextField(
503         verbose_name="The given answer",
504         help_text="The answers as pure string",
505         null=True
506     )
507
508     date = models.DateTimeField(
509         default=timezone.now,
510         null=True
511     )
512
513     solved = models.BooleanField(
514         default=False
```

```
515      )
516
517      def __str__(self):
518          return "Solution_{}_{}_{}".format(
519              self.question, self.solved, self.date)
520
521
522  def started_courses(user):
523      """
524      returns all courses started by a user
525      :param user: the user that is currently accessing the database
526      :return: all courses where the user has answered at least one course
527      """
528      courses = Course.objects.filter(
529          module__question__try__user=user)
530      return courses.distinct()
```