Coverage for **django/learning_base/multiple_choice/serializer.py** : 98%

51 statements | 50 run | 1 missing | 0 excluded

```
"""
serializers for MultipleChoice question types
"""

from rest_framework import serializers
from rest_framework.exceptions import ParseError
from .models import MultipleChoiceAnswer, MultipleChoiceQuestion


class MultipleChoiceAnswerSerializer(serializers.ModelSerializer):
    """
    Serializer for MultipleChoice answers
    :author: Leonhard Wiedmann
    """
    class Meta:
        model = MultipleChoiceAnswer
        fields = ('text', 'id', 'img')

    def create(self, validated_data):
        """
        creation method for new db entries
        :author: Leonhard Wiedmann
        :param: validated_data valid data for the creation
        """
        answer = MultipleChoiceAnswer(**validated_data)
        answer.question = validated_data['question']
        answer.save()


class MultipleChoiceQuestionPreviewSerializer(serializers.ModelSerializer):
    """
    Serializer for MultipleChoice question preview
    :author: Leonhard Wiedmann
    """
    class Meta:
        model = MultipleChoiceQuestion
        fields = ('body', "id",)


class MultipleChoiceAnswerEditSerializer(serializers.ModelSerializer):
    """
    Serializer for MultipleChoice answer editing
    :author: Leonhard Wiedmann
    """
    class Meta:
        model = MultipleChoiceAnswer
        fields = ("text", "id", "is_correct", "img")


class MultipleChoiceQuestionEditSerializer(serializers.ModelSerializer):
    """
    Serializer for MultipleChoice question editing
    :author: Leon Wiedmann
    """
    class Meta:
        model = MultipleChoiceQuestion
```

```
57          fields = ("id", 'question_image', 'feedback_image')
58
59      def to_representation(self, obj):
60          """
61          responsible for json serialization of objects
62          :author: Leonhard Wiedmann
63          :param obj: the object that should be serialized
64          :return: a json representation of the object
65          """
66          values = super(MultipleChoiceQuestionEditSerializer,
67                         self).to_representation(obj)
68          answers = obj.answer_set()
69          values['answers'] = MultipleChoiceAnswerEditSerializer(answers,
70                                                                 many=True).data
71          return values
72
73
74  class MultipleChoiceQuestionSerializer(serializers.ModelSerializer):
75      """
76      Serializer for MultipleChoice questions editing
77      :author: Leon Wiedmann
78      """
79      class Meta:
80          model = MultipleChoiceQuestion
81          fields = ('id', 'question_image')
82
83      def to_representation(self, obj):
84          """
85          responsible for json serialization of objects
86          :author: Leonhard Wiedmann
87          :param obj: the object that should be serialized
88          :return: a json representation of the object
89          """
90          values = super(MultipleChoiceQuestionSerializer,
91                         self).to_representation(obj)
92          answers = obj.answer_set()
93          values['answers'] = MultipleChoiceAnswerSerializer(answers,
94                                                             many=True).data
95          return values
96
97      def create(self, validated_data):
98          """
99          creating new database entries while editing
100         :author: Leonhard Wiedmann
101         :param validated_data: valid data
102         """
103         answers = validated_data.pop('answers')
104         question = MultipleChoiceQuestion(**validated_data)
105         question.module = validated_data['module']
106         question.save()
107
108         for answer in answers:
109             answer['question'] = question
110             answer_serializer = MultipleChoiceAnswerSerializer(data=answer)
111             if not answer_serializer.is_valid():
112                 raise ParseError(detail=answer_serializer.errors, code=None)
113             else:
114                 answer_serializer.create(answer)
115         if question.not_solvable():
116             question.delete()
117             raise ParseError(
118                 detail="Unsolvable question {}".format(question.title),
119                 code=None)
```