django-edit-suggestion

Release 1

vladimir gorea

Jul 04, 2021

CONTENTS:

CHAPTER

ONE

INSTALL

Install from `PyPI` with `pip`:

```
$ pip install django-edit-suggestion
```

1.1 Settings

Add `django_edit_suggestion` to your `INSTALLED_APPS`

```
INSTALLED_APPS = [
    # ...
    'django_edit_suggestion',
]
```

Requires to have Users

CHAPTER

TWO

USAGE

It's attached to a model via a field that during django setup phase creates a model related to that specific parent model.

EditSuggestion instances: - can be modified/deleted by the author of each instance - status can be "under review", "rejected" and "published" - status change need to pass a condition - changing the status to "published" updates the tracked model and locks the edit suggestion from being edited/deleted

2.1 Parent Model Example

Model has a field "edit_suggestion" that instantiates EditSuggestion A serializer module and parent serializer is passed as a tuple ex:

```
class Tag(models.Model)
    name = models.CharField(max_length=126)

def condition_check(user, parent_model_instance, edit_suggestion_instance):
    # do some checks and return a boolean
    if user.is_superuser or parent_model_instance.author == user:
        return True
    return False

class ParentModel(models.Model):
    excluded_field = models.IntegerField()
    m2m_type_field = models.ManyToManyField(Tags)
    edit_suggestions = EditSuggestion(
        excluded_fields=['excluded_field'],
        m2m_fields=({
            'name': 'm2m_type_field',
            'model': Tag,
            'through': 'optional. empty if not used',
            }),
        change_status_condition=condition_check,
        bases=(VotableMixin,), # optional. bases are used to build the edit␣
↪suggestion model upon them
        user_class=CustomUser, # optional. uses the default user model
    )
```

At django initializing stage the Edit Suggestion App creates a model for each Model having this field ex: "EditSuggestionParentModel"

Can access the model by ParentModel.edit_suggestions.model

2.2 How to use

2.3 Create new edit suggestion

After setting up the field inside the parent model just create a new edit suggestion by invoking the model `new()` method:

```
edit_suggestion = parentModelInstance.edit_suggestions.new({
    **edit_data,
    'edit_suggestion_author': user_instance
 })
```

2.4 Diff against the parent

Can see the differences between the parent instance and the curent edit:

```
changes = edit_suggestion.diff_against_parent()
```

It will return an object `ModelDelta` that has the attributes: - object.changes: tracked changes - object.changed_fields: changed fields name - object.old_record: parent instance - object.new_record: current edit instance

2.5 Publish

To publish an edit suggestion you need to pass in an user. If the `change_status_condition` does not pass, a `django.contrib.auth.models.PermissionDenied` exception will be raised.

```
edit_suggestion.edit_suggestion_publish(user)
```

This will change the status from `edit_suggestion.Status.UNDER_REVIEWS` to `edit_suggestion.Status.PUBLISHED`. After publishing, the edit suggestion won't be able to be edited anymore.

2.6 Reject

To reject an edit suggestion you need to pass in an user and a reason. If the `change_status_condition` does not pass, a `django.contrib.auth.models.PermissionDenied` exception will be raised.

```
edit_suggestion.edit_suggestion_reject(user, reason)
```

This will change the status from `edit_suggestion.Status.UNDER_REVIEWS` to `edit_suggestion.Status.REJECTED`. After rejecting, the edit suggestion won't be able to be edited anymore.

2.7 Foreign Fields different than type ForeignField

If using a foreign field different than ForeignField, like `mptt.fields.TreeForeignKey` use argument `special_foreign_fields` when initializing the EditSuggestion:

```
edit_suggestions = EditSuggestion(
    excluded_fields=(
        'created_at', 'updated_at', 'author', 'thumbs_up_array', 'thumbs_down_array'),
    special_foreign_fields=['parent',],
    change_status_condition=edit_suggestion_change_status_condition,
    post_publish=post_publish_edit,
    post_reject=post_reject_edit
)
```

2.8 M2M Fields

Can add ManyToManyField references by passing actual model or string. For referencing self instance use `'self'`:

```
class M2MSelfModel(models.Model):
    name = models.CharField(max_length=64)
    children = models.ManyToManyField('M2MSelfModel')
    edit_suggestions = EditSuggestion(
        m2m_fields=(({
                        'name': 'children',
                        'model': 'self',
                    },)),
        change_status_condition=condition_check,
    )
```

2.9 M2M Through support

Can use ManyToManyField with `through` table. The original pivot table will get copied and modified to point to the edit suggestion model. To save/edit the edit suggestion with `m2m through` field need to use a custom method.

```
class SharedChild(models.Model):
    name = models.CharField(max_length=64)

    def __str__(self):
        return self.name


class SharedChildOrder(models.Model):
    parent = models.ForeignKey('ParentM2MThroughModel', on_delete=models.CASCADE)
    shared_child = models.ForeignKey(SharedChild, on_delete=models.CASCADE)
    order = models.IntegerField(default=0)


class ParentM2MThroughModel(models.Model):
    name = models.CharField(max_length=64)
    children = models.ManyToManyField(SharedChild, through=SharedChildOrder)
    edit_suggestions = EditSuggestion(
        m2m_fields=(({
```

(continues on next page)

(continued from previous page)

```
                            'name': 'children',
                            'model': SharedChild,
                            'through': {
                                'model': SharedChildOrder,
                                'self_field': 'parent',
                            },
                        },)),
        change_status_condition=condition_check,
        bases=(VotableMixin,),  # optional. bases are used to build the edit␣
↪suggestion model upon them
        user_model=User,  # optional. uses the default user model
    )
```

2.10 Django REST integration

In 1.23 comes with EditSuggestionSerializer and ModelViewsetWithEditSuggestion.

There are 2 serializers: the one for listing (with minimal informations) and the one for detail/form view with all info.

The serializer is used for supplying the method `get_edit_suggestion_serializer` to the serializer for the model that receives edit suggestions. This method should return the edit suggestion serializer.

The serializer is used for supplying the method `get_edit_suggestion_listing_serializer` to the serializer for the model that receives edit suggestions. This method should return the edit suggestion serializer.

```
class TagSerializer(ModelSerializer):
    queryset = Tag.objects

    class Meta:
        model = Tag
        fields = ['name', ]

class ParentEditListingSerializer(ModelSerializer):
queryset = ParentModel.edit_suggestions

class Meta:
    model = ParentModel.edit_suggestions.model
    fields = ['pk', 'edit_suggestion_reason', 'edit_suggestion_author', 'edit_
↪suggestion_date_created']

class ParentEditSerializer(ModelSerializer):
    queryset = ParentModel.edit_suggestions
    tags = TagSerializer(many=True)

    class Meta:
        model = ParentModel.edit_suggestions.model
        fields = ['name', 'tags', 'edit_suggestion_reason', 'edit_suggestion_author']

class ParentSerializer(EditSuggestionSerializer):
    queryset = ParentModel.objects
    tags = TagSerializer(many=True)

    class Meta:
        model = ParentModel
        fields = ['name', 'tags']
```

(continues on next page)

(continued from previous page)

```
    @staticmethod
    def get_edit_suggestion_serializer():
        return ParentEditSerializer

    @staticmethod
    def get_edit_suggestion_listing_serializer():
        return ParentEditListingSerializer
```

The ModelViewsetWithEditSuggestion is to be inherited from when creating the model viewset:

```
class ParentViewset(ModelViewsetWithEditSuggestion):
serializer_class = ParentSerializer
queryset = ParentSerializer.queryset
```

It will add `edit_suggestions` for GET and `create_edit_suggestion` for POST requests.

Have `edit_suggestion_publish` and `edit_suggestion_reject` for POST requests.

Thus, to **retrieve the edit suggestions** for a specific resource using django rest we would send a GET request to `reverse('parent-viewset-edit-suggestions', kwargs={'pk': 1})`.

The url in string form would be `/api/parent/1/create_edit_suggestion/`.

To create an edit suggestion for a resource there are 2 ways: 1. POST request to `reverse('parent-viewset-create-edit-suggestion', kwargs={'pk': 1})` The url in string form would be `/api/parent/1/edit_suggestions/`.

2. use `ModelViewsetWithEditSuggestion` method `edit_suggestion_perform_create` since 1.34 the foreign key fields are handled as well

To **publish** using the viewset send a POST request to `reverse('parent-viewset-edit-suggestion-publish', kwargs={'pk': 1})` with a json object having `edit_suggestion_id` key with the edit suggestion pk.

To **reject** using the viewset send a POST request to `reverse('parent-viewset-edit-suggestion-reject', kwargs={'pk': 1})` with a json object having `edit_suggestion_id` key with the edit suggestion pk and `edit_suggestion_reject_reason` as the reason for rejection.

The responses will return status 403 if the rule does not verify, 401 for another exception and 200 for success.

2.11 Django REST integration for `m2m through`

In 1.30 we can handle creating edit suggestions with through m2m fields. It's the same procedure as with creating a normal edit suggestion but for the through m2m data we are using this data structure in the POST:

The creation is handled by the `edit_suggestion_handle_m2m_through_field` method of `ModelViewsetWithEditSuggestion` viewset. If there is a need to handle this in a different way, just override the method in your viewset.

CHAPTER

THREE

ABOUT

A django package for enabling django resources to be edited by users other than admin or resource author. The resource (an instance of a django model saved to database) will have a list of "edit suggestions" created by other users. The "edit suggestions" can be published, which will update the resource, or can be rejected. Users that pass a condition can publish or reject edit suggestions.

CHAPTER

FOUR

GITHUB REPO

https://github.com/smileservices/django-edit-suggestion

CHAPTER

FIVE

TODO

14/02/2021 Tests for file field, signals and copying parent model attributes

CHAPTER

SIX

CHANGES

1.40

1. fix m2m field str type

1.39

1. m2m fields work with specifying to relation by string (django style)

1.38

1. add support for foreign fields of type other than ForeignField

1.37

1. add support for file field. field is copied entirely
2. add support for signals
3. add support for copying parent model attributes

1.36

1. edit suggestions viewset uses parent *run_validation* method to get new suggestion data

1.35

1. bugfix tracking foreign field changes

1.34

1. add fix to handling foreign key fields on *ModelViewsetWithEditSuggestion* method *edit_suggestion_perform_create*

1.33

1. edit suggestion publish: fix m2m through copying of instance children to parent

1.32

1. rest_views: publish/reject edit suggestion: add success messages

1.31

1. rest_views: use edit suggestion listing serializer when retrieving the list of edit suggestions

1.30

1. add m2m through support in rest views and refactor the create edit suggestion

1.29

1. add m2m through support

1.28

1. `edit_suggestions` REST view returns paginated results. Can be filtered by status ex: `api/resources/88/edit_suggestions/?status=0`

1.27

1. Add post_publish/post_reject hooks

1.26

1. REST viewset `edit-suggestion-create` returns serialized instance of edit suggestion

1.25

1. add `edit_suggestion_publish` and `edit_suggestion_reject` to `ModelViewsetWithEditSuggestion` viewset
2. create tests for them

1.24

1. add m2m support to `diff_against_parent`

1.23

1. change status to Production/Stable
2. add django_rest support with adding EditSuggestionSerializer and ModelViewsetWithEditSuggestion