Django REST Framework Delegated Permissions Documentation

Release 0.5

Mirek Simek

Jun 06, 2018

Contents:

CHAPTER 1

List of classes

CHAPTER 2

Getting started

For this sample we will have an Invoice containing an Address. We want to set up permissions on Address so that the user that holds permissions to Invoice instance can view/modify the associated Address instance.

2.1 Preparation

Install `django>=1.11`, `djangorestframework` and `django-rest-delegated-permissions`

```
virtualenv --python=python3 venv
source venv/bin/activate
pip install django djangorestframework django-rest-delegated-permissions
```

Create your models:

```
class Address(models.Model):
    ... address fields
    class Meta:
        permissions = (
          ('view_address', 'Can view address'),
        )

class Invoice(models.Model):
    address = models.OneToOneKey(Address, related_name='invoice')
    class Meta:
        permissions = (
          ('view_invoice', 'Can view invoice'),
        )
```

Note: we have added declaration of an extra permission in the format of `appname.view_model`, users with this permission will be able to have a read access to the model instances (via REST's GET method either on collection of models or instance of a model).

And REST viewsets and serializers:

```
class AddressViewSet(ModelViewSet):
    queryset = Address.objects.all()
    serializer = AddressSerializer
    ...

class InvoiceViewSet(ModelViewSet):
    queryset = Invoice.objects.all()
    serializer = InvoiceSerializer
    ...
```

Add routing to your urls.py, generate a couple of instances of `Invoice` and `Address` and check that it works so far:

```
router = DefaultRouter()
router.register(r'invoice', InvoiceViewSet)
router.register(r'address', AddressViewSet)

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url('', include(router.urls))
]
```

2.2 Adding permissions

To add secure the viewset endpoints, you traditionally have to:

1. secure the modification operations via permissions defined on the viewset class
2. override its filter_queryset method to include only those objects that user has rights to (for example for resource listing)

`django-rest-related-permissions` groups these two cases into one `rest_delegated_permissions.BasePermission` class and as a convenience provides `rest_delegated_permissions.DjangoCombinedPermission` that contains these two steps for django model and django guardian permissions.

To be able to use permissions from related class (referenced via ForeignKey, m2m, etc) one needs to:

- remove permissions from viewsets
- link them to model classes so that Address's permission checker has an access to the Invoice permission checker
- put them into one registry that than composes queryset filters (so it is able to provide the `filter_queryset` implementation) and validates if user has rights to perform operation on related class.
- override `filter_queryset` and `get_permissions` methods with implementation that uses the permissions from the registry

These steps are implemented inside the `rest_delegated_permissions.RestPermissions` class. Let's use it:

```
perms = RestPermissions(
    add_django_permissions=True
)

# apply the same permissions that are on invoice
```

(continues on next page)

(continued from previous page)

```
# to address + add django and django guardian permissions
# defined on Address model
@perms.apply(permissions=DelegatedPermission(perms, 'invoice'))
class AddressViewSet(ModelViewSet):
    queryset = Address.objects.all()
    serializer = AddressSerializer
    ...

# apply only django and django guardian permissions on this model
@perms.apply()
class InvoiceViewSet(ModelViewSet):
    queryset = Invoice.objects.all()
    serializer = InvoiceSerializer
    ...
```

The code above is a shortcut - it at first extracts model class from ViewSet `queryset` field, associates the permissions given inside the decorator call with this class and sets up `filter_queryset` and `get_permissions`. More explicitly, the code can be rewritten as:

```
perms = RestPermissions(
    add_django_permissions=True
)
# the permissions below are combined via OR operator
perms.set_model_permissions(Address, [
        DelegatedPermission(perms, 'invoice'),
        DjangoCombinedPermission()
])
perms.set_model_permissions(Invoice, [
        DjangoCombinedPermission()
])

@perms.apply()
class AddressViewSet(ModelViewSet):
    ...

@perms.apply()
class InvoiceViewSet(ModelViewSet):
    ...
```

Note: Each model can be registered into RestPermissions only once. If you need multiple viewsets with different settings of permissions, use multiple instances of RestPermissions.

You can even use `rest-condition` package to combine permissions via AND or OR:

```
perms = RestPermissions(
    add_django_permissions=False
)
# the permissions below must both hold to give access
perms.set_model_permissions(Address, [
        Condition.And(
            DelegatedPermission(perms, 'invoice'),
            DjangoCombinedPermission())
])
```

If you try the code now, you will not have the access to any objects and listing will return an empty answer.

```
curl -u username:password http://localhost:8000/address
```

If you set up django permissions on User and try again, you will the access:

```
user = User.objects.get(...)
user.user_permissions.add(
    Permission.objects.get('app.view_invoice'))
```

```
curl -u username:password http://localhost:8000/address

# returns the previously created addresses
```

Similarly, you can add django guardian permission on a singe invoice and then see the address of that single invoice.

CHAPTER 3

Custom permissions

To implement a custom permission, base it on `rest_delegated_permissions.BasePermission`. For example the following permission grants access to the user that is the owner of an invoice:

```
...

class Invoice(models.Model):
    owner = models.ForeingKey(User, ...)
    ...

class OwnerPermission(BasePermission):

    def has_object_permission(self, request, view, obj):
        return obj.owner == request.user

    def filter(self, rest_permissions, filtered_queryset,
               user, action):
        yield filtered_queryset.filter(owner=user)

@perms.apply(permissions=OwnerPermission())
class InvoiceViewSet(ModelViewSet):
    ...
```

Now, when you GET `.../invoice`, you'll receive only those to which you are the owner (or have django/django guardian) permissions. If you GET `.../address`, you'll receive those addresses that belong only to the invoices to which you have the rights.