djangorestframework-composed-permissions Documentation

Release 0.1

Andrey Antukh

Sep 27, 2017

Contents

A simple way to define complex permissions for django-rest-framework.

CHAPTER 1

Introduction

django-rest-framework by default has large ways to define permissions but none of them permits define a complex, multi depth and with logical operators permissions rules.

djangorestframework-composed-permissions is full compatible with a django-rest-framework permissions api but implements a simple way to build permission objects using reusable components (called permission components).

djangorestframework-composed-permissions has 3 type of classes:

- Permission components: a basic unit of permission.
- Permission set: a collection of components joined with logic operator (and, or & not)
- Composed permissions: container of both, components and sets having same interface as django-rest-framework default permission.

CHAPTER 2

How to install?

It is very simple, install it on your virtualenv with pip:

```
pip install djangorestframework-composed-permissions
```

CHAPTER 3

Quickstart

The best way to understand how djangorestframework-composed-permissions works, is looking some examples.

This package comes with some generic permission components defined for you, later we will see how to define a own component but for the first example we go to use one generic.

We go to define one permission and viewset that uses it:

```
from restfw_composed_permissions.base import (BaseComposedPermision, And, Or)
from restfw_composed_permissions.generics.components import (
        AllowAll, AllowOnlyAuthenticated, AllowOnlySafeHttpMethod)


class UserPermission(BaseComposedPermision):
    def global_permission_set(self):
        return Or(AllowOnlyAuthenticated,
                  And(AllowAll, AllowOnlySafeHttpMethod))

    def object_permission_set(self):
        return AllowAll


class UserViewSet(viewsets.ModelViewSet):
    """
    A viewset for viewing and editing user instances.
    """
    permission_classes = (UserPermission,)

    serializer_class = UserSerializer
    queryset = User.objects.all()
```

Note: *And* & *Or* classes are permission sets, that groups some components with logical operator. Also exists *Not* but we don't use it on this example.

global_permission_set method must return a permission set or only one component, and it is evaluted on every request, however *object_permission_set* is only evaluted when a create, update, delete operation is executed.

With *UserPermission* defined on previous example, we allow any authenticated user for do any thing and allow anonymous requests only if request method is safe.

CHAPTER 4

Low level api reference

This is a low level api documentation for 3 main components of this package.

Permission Component

class `restfw_composed_permissions.base.`**`BasePermissionComponent`**

`has_permission`(*self*, *permission*, *request*, *view*)
This method must be defined always, because it is used on global permission evaluation process and it is executed always on every request.

If you not implement this method, it will raise NotImplementedError exception.

Return type bool

`has_object_permission`(*self*, *permission*, *request*, *view*, *obj*)
This method must be defined if this component will be used for object permission checking.

By default, returns same thing as has_permission.

Return type bool

You can see *restfw_composed_permissions.generics.components* for more examples of how define own permission components.

Permission Sets

Permissions sets implement same interface as components. Permission sets groups N number of components with logical operator, with exception of *Not* that has special behavior and it accepts only one component as parameter.

And, *Or* & *Not* are the default permission sets defined on this package.

The usage of *And* example:

```
# Simple usage as class instance
class SomePermission1(BaseComposedPermision):
    global_permission_set = (lambda s: And(Component1, Component2))

# Using & operator of components
class SomePermission2(BaseComposedPermision):
    global_permission_set = (lambda s: Component1() & Component2())
```

The usage of *Or* examples:

```
# Simple usage as class instance
class SomePermission1(BaseComposedPermision):
    global_permission_set = (lambda s: Or(Component1(some_param), Component2))

# Using | operator of components
class SomePermission2(BaseComposedPermision):
    global_permission_set = (lambda s: Component1() | Component2())

# Returning a list of components
class SomePermission3(BaseComposedPermision):
    global_permission_set = (lambda s: [Component1, Component2])
```

Finally, *Not* usage examples:

```
# Simple usage as class instance
class SomePermission1(BaseComposedPermision):
    global_permission_set = (lambda s: Not(Component1))

# Using ~ operator of components
class SomePermission2(BaseComposedPermision):
    global_permission_set = (lambda s: ~Component1())
```

Composed Permission

This is a toplevel class of 3 main components of this package.

class `restfw_composed_permissions.base.`**`BaseComposedPermision`**

Any subclass of this must define *global_permission_set* as mandatory method and optionally *object_permission_set* method.

These methods must return a *`BasePermissionComponent`* subclass or `BasePermissionSet` subclass.

CHAPTER 5

Generics

Components

class `restfw_composed_permissions.generic.components.`**`AllowAll`**
Always allow all requests without any constraints.

class `restfw_composed_permissions.generic.components.`**`AllowOnlyAnonymous`**
Only allow anonymous requests.

class `restfw_composed_permissions.generic.components.`**`AllowOnlyAuthenticated`**
Only allow authenticated requests.

class `restfw_composed_permissions.generic.components.`**`AllowOnlySafeHttpMethod`**
Only allow safe http methods.

class `restfw_composed_permissions.generic.components.`**`ObjectAttrEqualToObjectAttr`**
This is a object level permision component and if is used on global permission context it always returns True.

This component checks the equality of two expressions that are evaluted in “safe” way. On the context of eval are exposed “obj” as current object and “request” as the current request.

This component works well for check a object owner os similary.

Example:

```
class SomePermission(BaseComposedPermision):
    global_permission_set = (lambda self: AllowAll)
    object_permission_set = (lambda self:
                                ObjectAttrEqualToObjectAttr("request.user", "obj.
↪owner"))
```

Index