drfdapc Documentation

Release 0.4

Christian Ledermann

Jul 26, 2018

Contents

CHAPTER 1

License

 DRF Deny All - Allow Specific Permission Classes.

In Django rest Framework the permission classes work as:

> *If any permission check fails an [. . .] exception will be raised, and the main body of the view will not run.*

This implementation of permission classes takes a list of functions that determine if the access is allowed. If **any** of the functions return *True*, the access is *allowed*, if **none** of the permission checks passes the access will be *denied*. This enables to write small, reusable and chainable permissions

The BasePermission classes provide the *has_permission(self, request, view)* and *has_object_permission(self, request, view, obj)* methods.

The **Default** is *deny_all* which means when you subclass *DABasePermission*, *DARWBasePermission* or *DACrudBasePermission* you have to set **_permissions* explicitly on your class to allow access.

If you only need view level security you may set *object_rw_permissions = (allow_all,)* otherwise your view will reject users when *.get_object()* is called through REST framework's view machinery.

class permissions.**DABasePermission**

Deny Allow Base Permisson.

Permissions subclassed from this Base class will run all permission checks specified in the *rw_permissions* tuple.

It does not check if it is a read or a write access and treat **all** access methods in the same way.

has_object_permission(*request*, *view*, *obj*)
Object level permissions.

All request methods are checked against the *object_rw_permissions*. If None of those permissions returns True the access is denied.

This is run by REST framework's generic views when *.get_object()* is called. If you're writing your own views and want to enforce object level permissions, or if you override the get_object method on a generic view, then you'll need to explicitly call the *.check_object_permissions(request, obj)* method on the view at the point at which you've retrieved the object.

has_permission(*request*, *view*)
Check permissions.

Before running the main body of the view each permission in *rw_permissions* is checked.

All request methods are treated in the same way.

class permissions.**DACrudBasePermission**
Deny Allow Base Read/Write specific Permisson.

Permissions subclassed from this Base class will run all permission checks specified in the *rw_permissions* tuple for all read and write access methods.

If none of the *rw_permissions* passed it will check the permissions based on the http access methods.

For read access (*options*, *head*, *get*) methods all permissions in the *read_permissions* methods are checked.

For create access (*post*) all permissions in the *add_permissions* are checked.

For update access (*put*) all permissions in the *change_permissions* are checked.

For delete access (*delete*) all permissions in the *delete_permissions* are checked.

has_object_permission(*request*, *view*, *obj*)
Object level permissions.

All request methods are checked against the *object_rw_permissions*. If None of those Permissions returns True the permissions are checked against *object_read_permissions* if the request method is a *get*, *head* or *options*, or against *object_change_permissions* for *put'and 'patch*, against *object_add_permissions* for *post* and against *object_delete_permissions* for *delete* methods.

This is run by REST framework's generic views when .get_object() is called. If you're writing your own views and want to enforce object level permissions, or if you override the get_object method on a generic view, then you'll need to explicitly call the *.check_object_permissions(request, obj)* method on the view at the point at which you've retrieved the object.

has_permission(*request*, *view*)
Check permissions.

Before running the main body of the view each permission in *rw_permissions* is checked. If None of these permissions allows access then the permissions in *read_permissions* are checked for the (*options*, *head*, *get*) methods. For the *post* method all permissions in the *add_permissions* are checked. For *put* and *patch* methods all permissions in the *change_permissions* are checked. For the *delete* method all permissions in the *delete_permissions* are checked.

class permissions.**DARWBasePermission**
Deny Allow Base Read/Write specific Permisson.

Permissions subclassed from this Base class will run all permission checks specified in the *rw_permissions* tuple for all read and write access methods.

If none of the *rw_permissions* passed it will check the permissions based on the http access methods.

has_object_permission(*request*, *view*, *obj*)
Object level permissions.

All request methods are checked against the *object_rw_permissions*. If None of those Permissions returns True the permissions are checked against *object_read_permissions* if the request method is a *get*, *head* or *options*, or against *object_write_permissions* for *put*, *patch*, *post* and *delete* methods.

This is run by REST framework's generic views when .get_object() is called. If you're writing your own views and want to enforce object level permissions, or if you override the get_object method on a generic view, then you'll need to explicitly call the *.check_object_permissions(request, obj)* method on the view at the point at which you've retrieved the object.

has_permission(*request*, *view*)
Check permissions.

Before running the main body of the view each permission in *rw_permissions* is checked. If None of these permissions allows access then the permissions in *read_permissions* are checked for the (*options*, *head*, *get*) methods. For write access (*post*, *put*, *patch*, *delete*) methods all permissions in the *write_permissions* methods are checked.

permissions.**allow_all**(**args*, ***kwargs*)
Allow anyone.

This permission will allow unrestricted access, regardless of the request being authenticated or unauthenticated.

permissions.**allow_authenticated**(*request*, **args*, ***kwargs*)
Allow authenticated users.

This permission class will deny permission to any unauthenticated user, and allow permission to any authenticated user.

permissions.**allow_authorized_key**(*request*, *view*, **args*, ***kwargs*)
Allow access with a shared secret.

The request must contain a authentication header that matches one of the API Keys.

The API Keys are set in the authorized_keys attribute of the view. This is useful for authorization between services that communicate via drf where you'd rather have the keys as configuration and connect without authentication.

permissions.**allow_staff**(*request*, **args*, ***kwargs*)
Allow staff access.

This permission allows access to any user that has the *is_staff* flag set.

permissions.**allow_superuser**(*request*, **args*, ***kwargs*)
Superuser access.

This permission allows access to any user that has the *is_superuser* flag set.

permissions.**authenticated_users**(*func*)
Abstract common authentication checks as a decorator.

request is required either as the first positional argument or as a Keyword argument

permissions.**deny_all**(**args*, ***kwargs*)
Deny Access to everyone.

This permission is not strictly required, since you can achieve the same result by using an empty tuple for the permissions setting, but you may find it useful to specify this class because it makes the intention explicit.

This permission on it's own is not useful as *nobody* will ever be able to access a view protected with it.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index