
django-role-permissions Documentation

Release 0.1

Filipe Ximenes

Mar 27, 2017

Contents

1	Setup	3
1.1	Installation	3
1.2	Configuration	3
2	Quick Start	5
3	Roles	7
3.1	Roles File	7
3.2	Available Role Permissions	8
3.3	Role Limits	8
3.4	Available Permissions Limits	8
4	Object permission checkers	9
4.1	permissions.py file	9
4.2	Checking object permission	9
5	Utils	11
5.1	Shortcuts	11
5.2	Permission and role verification	12
5.3	Template tags	13
6	Mixins and Decorators	15
6.1	Decorators	15
6.2	Mixins	15
7	Settings	17
7.1	Redirect to the login page	17
8	Indices and tables	19

Contents:

Installation

Install from PyPI with `pip`:

```
pip install django-role-permissions
```

Configuration

Add `rolepermissions` to your `INSTALLED_APPS`

```
INSTALLED_APPS = (  
    ...  
    'rolepermissions',  
    ...  
)
```


CHAPTER 2

Quick Start

Create a `roles.py` file in the same folder as your `settings.py` and two roles:

```
from rolepermissions.roles import AbstractUserRole

class Doctor(AbstractUserRole):
    available_permissions = {
        'create_medical_record': True,
    }

class Nurse(AbstractUserRole):
    available_permissions = {
        'edit_patient_file': True,
    }
```

Add a reference to your roles module to your settings:

```
ROLEPERMISSIONS_MODULE = 'myapplication.roles'
```

When you create a new user, set its role using:

```
>>> from rolepermissions.shortcuts import assign_role
>>> user = User.objects.get(id=1)
>>> assign_role(user, 'doctor')
```

and check its permissions using

```
>>> from rolepermissions.verifications import has_permission
>>> has_permission(user, 'create_medical_record', 'doctor')
True
>>> has_permission(user, 'edit_patient_file')
False
```

You can also change users permissions:

```
>>> from rolepermissions.shortcuts import grant_permission, revoke_permission
>>>
>>> revoke_permission(user, 'create_medical_record', 'doctor')
>>> grant_permission(user, 'edit_patient_file')
>>>
>>> has_permission(user, 'create_medical_record')
False
>>> has_permission(user, 'edit_patient_file')
True
```

Roles File

Create a `roles.py` file anywhere inside your django project and reference it in the project settings file.

`my_project/roles.py`

```
from rolepermissions.roles import AbstractUserRole

class Doctor(AbstractUserRole):
    available_permissions = {
        'create_medical_record': True,
    }

class Nurse(AbstractUserRole):
    available_permissions = {
        'edit_patient_file': True,
    }
```

`settings.py`

```
ROLEPERMISSIONS_MODULE = 'my_project.roles'
```

Each class that imports `AbstractUserRole` is a role on the project and has a snake case string representation. For example:

```
from rolepermissions.roles import AbstractUserRole

class SystemAdmin(AbstractUserRole):
    available_permissions = {
        'drop_tables': True,
    }
```

will have the string representation: `system_admin`.

Available Role Permissions

The field `available_permissions` lists what permissions the role can be granted. The boolean referenced on the `available_permissions` dictionary is the default value to the referred permission.

Role Limits

The field `role_limits` contains a limit (whatever object you want) the role has.

```
from rolepermissions.roles import AbstractUserRole

class SystemAdmin(AbstractUserRole):
    available_permissions = {
        'drop_tables': True,
    }
    role_limit = 100
```

or

```
from rolepermissions.roles import AbstractUserRole

class SystemAdmin(AbstractUserRole):
    available_permissions = {
        'drop_tables': True,
    }
    role_limit = { 'create_offers': 10, 'create_auctions': 20 }
```

Role limits can be accessed

```
system_admin_role_limits = SystemAdmin.get_role_limit()
```

Available Permissions Limits

The field `available_permissions_limit` lists the limits for the permissions of the role. It is not mandatory to have a limit for each permission.

```
from rolepermissions.roles import AbstractUserRole

class SystemAdmin(AbstractUserRole):
    available_permissions = {
        'create_offers': True,
        'create_auctions': True,
    }
    available_permissions_limits = {
        'create_offers': 10,
        'create_auctions': 20
    }
```

Can be accessed as follow

```
system_admin_create_offers_limit = SystemAdmin.get_permission_limit('create_offers')
```

Object permission checkers

permissions.py file

You can add a `permissions.py` file to each app. This file should contain registered object permission checker functions.

`my_app/permissions.py`

```
from rolepermissions.permissions import register_object_checker
from my_project.roles import SystemAdmin

@register_object_checker()
def access_clinic(role, user, clinic):
    if role == SystemAdmin:
        return True

    if user.clinic == clinic:
        return True

    return False
```

when requested the object permission checker will receive the role of the user, the user and the object being verified and should return `True` if the permission is granted.

Checking object permission

Use the `has_object_permission` method to check for object permissions.

Shortcuts

get_user_role (*user*)

Returns the roles classes of the user.

```
from rolepermissions.shortcuts import get_user_roles

roles = get_user_roles(user)
```

assign_role (*user*, *role*)

Assigns a role to the user. Role parameter can be passed as string or role class object.

```
from rolepermissions.shortcuts import assign_role

assign_role(user, 'doctor')
```

remove_role (*user*)

Remove a role that was assigned to the specified user.

Remove all roles that was assigned to the specified user.

available_perm_status (*user*)

Returns a dictionary containing all permissions per role available to the role of the specified user. Role names are the keys of the dictionary with each role represented by another dictionary with permissions as keys. The permissions values are True or False indicating if the permission is granted or not.

```
from rolepermissions.shortcuts import available_perm_status

permissions = available_perm_status(user)

if permissions['create_medical_record']:
    print 'user can create medical record'
```

grant_permission (*user, permission_name, role=None*)

Grants a permission to a user for the given role. If no role passed, it will iterate through all roles of the given user and try to grant the permission for each role that has it. Will not grant a permission if the user doesn't have the role or the permission is not listed in the role's `available_permissions`.

```
from rolepermissions.shortcuts import grant_permission

grant_permission(user, 'create_medical_record', 'doctor')
>>> True
grant_permission(user, 'create_medical_record')
>>> True
```

revoke_permission (*user, permission_name, role=None*)

Revokes a permission for the given role. If no role passed, it will iterate through all roles to remove the permission from each role that contains it.

```
from rolepermissions.shortcuts import revoke_permission

revoke_permission(user, 'create_medical_record', 'doctor')
>>> True
revoke_permission(user, 'create_medical_record')
>>> True
```

Permission and role verification

The following functions will always return `True` for users with `supper_user` status.

has_role (*user, roles*)

Receives a user and a role and returns `True` if user has the specified role. Roles can be passed as object, snake cased string representation or inside a list.

```
from rolepermissions.verifications import has_role
from my_project.roles import Doctor

if has_role(user, [Doctor, 'nurse']):
    print 'User is a Doctor or a nurse'
```

has_permission (*user, permission, role=None*)

Receives a user and a permission and returns `True` if the user has the specified permission for the role given. If no role passed it will iterate through all user roles and will check if the user has the given permission for every role that has the permission in the `available_permissions` list.

```
from rolepermissions.verifications import has_permission
from my_project.roles import Doctor
from records.models import MedicalRecord

if has_permission(user, 'create_medical_record'):
    medical_record = MedicalRecord(...)
    medical_record.save()

if has_permission(user, 'create_medical_record', 'doctor'):
    medical_record = MedicalRecord(...)
    medical_record.save()
```

get_role_limit (*role*)

Receives the role or a string (role name) and returns the limit for that role. Limit can be anything.

```
from rolepermissions.shortcuts import get_role_limit

permission_limit = get_permission_limit('doctor')
```

def get_permission_limit(*role, permission_name*)

Receives the role or a string (role name) and the *permission_name* and returns the limit for the permission or none if it hasn't been declared.

```
from rolepermissions.shortcuts import get_permission_limit

permission_limit = get_permission_limit('doctor', 'create_medical_record')
```

has_object_permission (*checker_name, user, obj*)

Receives a string referencing the object permission checker, a user and the object to be verified.

```
from rolepermissions.verifications import has_object_permission
from clinics.models import Clinic

clinic = Clinic.objects.get(id=1)

if has_object_permission('access_clinic', user, clinic):
    print 'access granted'
```

Template tags

To load template tags use:

```
{% load permission_tags %}
```

***filter* has_role**

Receives a camel case representation of a role or more than one separated by coma.

```
{% load permission_tags %}
{% if user|has_role:'doctor,nurse' %}
    the user is a doctor or a nurse
{% endif %}
```

***filter* can**

Role permission filter. Role after permission is optional.

```
{% load permission_tags %}
{% if user|can:'create_medical_record:doctor' %}
    <a href="/create_record">create record</a>
{% endif %}
```

***tag* can**

If no user is passed to the tag, the logged user will be used in the verification.

```
{% load permission_tags %}

{% can "access_clinic" clinic user=user as can_access_clinic %}
{% if can_access_clinic %}
    <a href="/clinic/1/">Clinic</a>
{% endif %}
```

Mixins and Decorators

Decorators

Decorators require that the current logged user attend some permission grant. They are meant to be used on function based views.

has_role_decorator (*role*)

Accepts the same arguments as `has_role` function and raises `PermissionDenied` in case it returns `False`.

```
from rolepermissions.decorators import has_role_decorator

@has_role_decorator('doctor')
def my_view(request, *args, **kwargs):
    ...
```

has_permission_decorator (*permission_name*)

Accepts the same arguments as `has_permission` function and raises `PermissionDenied` in case it returns `False`.

```
from rolepermissions.decorators import has_permission_decorator

@has_permission_decorator('create_medical_record')
def my_view(request, *args, **kwargs):
    ...
```

Mixins

Mixins require that the current logged user attend some permission grant. They are meant to be used on class based views.

class HasRoleMixin(object)

Add `HasRoleMixin` mixin to the desired CBV (class based view) and use the `allowed_roles` attribute to set the roles that can access the view. `allowed_roles` attribute will be passed to `has_role` function, and `PermissionDenied` will be raised in case it returns `False`.

```
from django.views.generic import TemplateView
from rolepermissions.mixins import HasRoleMixin

class MyView(HasRoleMixin, TemplateView):
    allowed_roles = 'doctor'
    ...
```

class HasPermissionsMixin(object)

Add `HasPermissionsMixin` mixin to the desired CBV (class based view) and use the `required_permission` attribute to set the roles that can access the view. `required_permission` attribute will be passed to `has_permission` function, and `PermissionDenied` will be raised in case it returns `False`.

```
from django.views.generic import TemplateView
from rolepermissions.mixins import HasPermissionsMixin

class MyView(HasPermissionsMixin, TemplateView):
    required_permission = 'create_medical_record'
    ...
```

Redirect to the login page

Instead of getting a Forbidden (403) error when the user has no permission, you can make the request be redirected to the login page. Add the following variable to your django `settings.py`:

`settings.py`

```
ROLEPERMISSIONS_REDIRECT_TO_LOGIN = True
```


CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

A

`assign_role()` (built-in function), [11](#)
`available_perm_status()` (built-in function), [11](#)

G

`get_role_limit()` (built-in function), [12](#)
`get_user_role()` (built-in function), [11](#)
`grant_permission()` (built-in function), [11](#)

H

`has_object_permission()` (built-in function), [13](#)
`has_permission()` (built-in function), [12](#)
`has_permission_decorator()` (built-in function), [15](#)
`has_role()` (built-in function), [12](#)
`has_role_decorator()` (built-in function), [15](#)

R

`remove_role()` (built-in function), [11](#)
`revoke_permission()` (built-in function), [12](#)