django-permissionsx Documentation

Release 1.3.4

Robert Pogorzelski

Apr 19, 2017

Contents

- **Version:**
  - 1.3.4
- **Python package:**
  - http://pypi.python.org/pypi/django-permissionsx/
- **Source code:**
  - http://github.com/robpogorzelski/django-permissionsx/
- **Bug tracker:**
  - http://github.com/robpogorzelski/django-permissionsx/issues/
- **Example project:**
  - http://github.com/robpogorzelski/django-permissionsx-example/

CHAPTER 1

Introduction

django-permissionsx is an alternative to Django permissions system. The main difference is that this package does not store authorization logic in database, but instead allows defining permissions on the view level using concise syntax (similar to complex lookups using *Q*) and performs authorization checks against HttpRequest object. You could think of it as a wrapper around common patterns such as *@login_required* decorator or checking *request.user.is_authenticated()*.

You will find that defining permissions is similar to filtering QuerySets and complex lookups with Q objects. For example:

```
P(user__is_authenticated=True) & P(P(user__is_staff=True) | P(user__is_
↪superuser=True))
```

means that the user will be granted access if is logged in **and** is either a staff member, **or** a superuser.

The goal of this project is to make authorization related code as much reusable and consistent as possible. Therefore, permissions can be easily used for multiple views, inherited, used in templates or for building mobile API using django-tastypie.

CHAPTER 2

Contents

Quick Start

Compatibility note

This package is intended to work with Python 3.5+ and Django 1.10+.

1. Install `django-permissionsx` package:

```
pip install django-permissionsx
```

2. Define permissions in a module of your choice:

```
from permissionsx.models import P
from permissionsx.models import Permissions


class ManagerPermissions(Permissions):

    rules = P(user__is_staff=True) & P(user__has_company_assigned=True)
```

3. Add permissions to your views, e.g.:

```
from permissionsx.contrib.django.views import PermissionsListView

from example.profiles.permissions import ManagerPermissions
```

```
class AuthenticatedListView(PermissionsListView):

    queryset = Item.objects.all()
    permissions = Permissions(
        P(user__is_authenticated=True)
    )


class ManagerListView(PermissionsListView):

    queryset = Item.objects.all()
    permissions = ManagerPermissions()
```

4. Don't forget to add `permissionsx` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.staticfiles',
    [...]
    'permissionsx',
```

5. Now add request context processor, so you can use permissions in your templates:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    [...]
    'django.core.context_processors.request',
    [...]
)
```

6. Apply permissions in templates if you need:

```
{% load permissionsx_tags %}
{% block content %}
{% permissions 'example.profiles.permissions.ManagerPermissions' as user_is_manager %}
<ul id="utility-navigation">
    {% if user_is_manager %}
        <a href="#">Publish article</a>
    {% endif %}
</ul>
{% endblock content %}
```

7. That's all!

User will be redirected to `LOGIN_URL` by default, if:

- not logged in and tries to access `AuthenticatedListView`;
- not a staff member, `request.user.profile.is_manager` is set to `False` and tries to access `ManagerListView`;

- *Publish article* option will be displayed only if user meets `ManagerPermissions` conditions.

Settings

PERMISSIONSX_REDIRECT_URL

Defaults to *settings.LOGIN_URL*.

If user has not been granted permission to access a Django view, redirect to *PERMISSIONSX_REDIRECT_URL*.

PERMISSIONSX_LOGOUT_IF_DENIED

Defaults to *False*.

If user has not been granted permission to access a Django view, log the user out before redirecting to *PERMISSIONSX_REDIRECT_URL*.

Tutorial

Table of Contents

Example

Please visit http://github.com/robpogorzelski/django-permissionsx-example/ for a full working example of a Django project utilizing class-based views permissions checking and Tastypie integration.

Deeper look

P

permissionsx.models.P

`P` is the smallest building block. Permissions are defined using keyword arguments, for example:

```
P(user__is_superuser=True)
```

It means that the value of `request.user.is_superuser` will be compared with `True`. If the final result is `True`, the user will be granted access. Otherwise the user will be redirected to the `settings.LOGIN_URL` by default.

`P` objects can be negated and combined using ~, & and | operators, exactly the same way as Q objects.

Optionally, for more advanced workflows, `P` can be passed additional two keyword arguments for overriding default behavior:

- `if_false`
- `if_true`

It is useful in situations where user needs to be redirected to different URLs when specific conditions are met. For example, if user:

- is not authenticated, redirect to login view by default;
- is authenticated, but has insufficient permissions (e.g. needs to upgrade account), redirect to a view with payment options and show message using `django.contrib.messages`;
- is authenticated and has sufficient permissions, let in.

Arg

permissionsx.models.Arg

`Arg` is used when permissions checking involves passing parameter to a method of an object attached to the request. This is most often used for checking access to specific objects, e.g.:

```
P(user__has_access_to=Arg('invoice'))
```

Note that `Arg` parameter is passed as a string. Basically, it is equivalent to:

```
request.user.has_access_to(request.invoice)
```

Cmp

permissionsx.models.Cmp

`Cmp` is used when permissions require comparing values of objects attached to the request even if the compared attributes are not currently available in the method scope. Also, `Cmp` prevents exceptions from non-existing relations (e.g. *request.user.company* while *company* can be null).

```
P(company__main_address__city=Cmp('user.address.city'))
```

Note that `Cmp` parameter is passed as a string. It is equivalent to:

```
request.company.main_address.city == request.user.address.city
```

So in this scenario, view is passed e.g. *kwargs* containing *{'slug': 'company-xyz'}*. Company XYZ instance is retrieved from database and its headquarter's city is compared to the one of a user currently accessing view. If these match, user is allowed to view page, can be redirected, shown a message etc.

Permissions

permissionsx.models.Permissions

`Permissions` may be passed as an instance or a class to Django views or Tastypie authorization classes and it encapsulates `P` definitions, e.g.:

```
class UserPermissions(Permissions):

    rules = P(user__is_authenticated=True)


class ArticleDetailView(PermissionsDetailView):

    model = Article
    permissions = UserPermissions()


class StaffOnlyAuthorization(TastypieAuthorization):

    permissions = UserPermissions()
```

Or the same just without subclassing `Permissions`:

```
class ArticleDetailView(PermissionsDetailView):

    model = Article
    permissions = Permissions(P(user__is_authenticated=True))
```

And yet another example, this time by reusing single definition:

```
is_authenticated = P(user__is_authenticated=True)


class ArticleDetailView(PermissionsDetailView):

    model = Article
    permissions = Permissions(is_authenticated)
```

Attributes:

- `permissions` - required.

Important Notes

Django Views

- If an anonymous user is being redirected, current `request.get_full_path()` will be added to the URL as *next* parameter.
- Overrides for Django views must accept `(request, *args, **kwargs)`.

Compatibility

Python 3.5 & 3.6

Package	django-permissionsx 1.4.0
Django 1.10	
Django 1.11	
django-tastypie 0.13.3	
django-debug-toolbar 1.7	

Changelog

1.4.0

- Updated package to work with Django 1.1x and Python 3.5+.

1.3.4

- Adding redirects inside of a wrapping P() is now possible, e.g.

```
class TestView(PermissionsTemplateView):

    permissions = Permissions(
        P(user__is_authenticated=True) &
        P(
            P(user__is_superuser=True) | P(object__owner=Cmp('user')), if_
↪false=AccessDeniedView.as_view()
        )
    )
```

1.3.3

- Minor maintenance release. No changes affecting current installations.

1.3.2

- Minor maintenance release. No changes affecting current installations.
- Changed policy on compatibility. Each release is now guaranteed to support:
- latest stable release and next upcoming of Django;
- latest Python 2.x and 3.x versions;
- latest stable versions of interoperable packages (currently *django-debug-toolbar* and *django-tastypie*)

1.3.1

- Minor maintenance release. No changes affecting current installations.

1.3.0

- On a view level *permissions_class* is now *permissions*.
- Renamed *get_permissions* to *get_rules*.
- Renamed *permissions* to *rules*.
- Renamed *check_permissions* to *check*.
- Other internal API changes.
- Object passed to *permissions* must be an instance.
- Added example project at http://github.com/robpogorzelski/django-permissionsx-example.
- Removed *PERMISSIONSX_DEBUG* setting.
- Renamed *PermissionsDebugPanel* to *PermissionsPanel* (following django-debug-toolbar).

1.2.1

- Bugfix release. Merging permissions with a `Permissions` instance with no rules defined was raising *TypeError* exception.

1.2.0

- Removed *set_request_context()* method from `Permissions`. This was adding unjustified complexity. Instead, inheritance and *super()* calls can be used.
- Added new operator: *Cmp()*. This allows to compare permission rules to request object even if they are not currently available in the method scope. Also, this prevents exceptions from non-existing relations (e.g. *request.user.company* while *company* can be null).
- Simplification. Removed dependency on Django patches or middleware tricks. Now if a user is anonymous and permissions are checked, and they fail on specific attributes of the *User* instance (e.g. *get_profile()*), user will be denied access for that specific rule by default.
- Updated Django Debug Toolbar integration.
- Added support for passing permission rules to classes having permissions already defined. This will cause all rules to be merged using AND (&). For example, following is now possible:

`accounts/permissions.py`

```
class ContentEditablePermissions(Permissions):

    def get_permissions(self, request, **kwargs):
        try:
            request.content = Content.objects.get(slug=kwargs.get('slug'))
        except Content.DoesNotExist:
            request.content = None
        return P(user__is_author_of=Arg('content')) | P(content__publisher=Cmp('user.
↪publisher'))
```

`content/views.py`

```
class ContentUpdateView(DjangoViewMixin, UpdateView):

    model = Content
    template_name = 'content/content_edit.html'
    form_class = ContentCreateUpdateForm
    permissions_class = ContentEditablePermissions(
        P(content__can_change_price=True)
    )
```

So the final result would be:

```
request.content.can_change_price() & (request.user.is_author_of(request.content) |␣
↪(request.content.publisher == request.user.publisher))
```

1.1.4

- Fixed Django debug toolbar panel.
- Removed caching (explanation).

1.1.3

- Added in-memory caching (`settings.PERMISSIONSX_CACHING`).
- Added tests for Django Views, settings and overrides.
- Changed the way overrides work. Few things got simplified by the way. Now it is possible to use multiple overrides attached to `P` objects, not the top-level `Permissions`.

1.1.2

- Added support over overriding response behavior on a permission level.
- One-liners for defining permissions.
- `Arg` allows passing request object to permission checking function.
- Package `django-classy-tags` is no longer a requirement.
- Added Sphinx documentation with extended examples.

1.1.0

- New syntax possible for retrieving related objects, e.g. `P(user__get_profile__related_object__is_something=`

1.0.0

- Added support for custom response classes (e.g. for changing redirect URL, adding custom user message).
- Added tests for checking permissions.
- Minor fixes and improvements.

0.0.9

- Added support for Django templates, including per-object checks.
- Renamed class-level `permissions` to `permissions_class`.
- Dropped support for simple permissions defining for the benefit of greater flexibility.
- Renaming and refactoring, again. Good stuff: managed to get rid of middleware and a class. Things got largely simplified in general.
- Requirement: `django-classy-tags`.

0.0.8

- This version is backward **incompatible**.
- Changed syntax to follow QuerySet filtering convention.
- Sadly, tests are gone. Need to write new ones, what will not happen until 1.0.0 release.
- Example project's gone. Will be back at a different URL.
- `PERMISSIONSX_DEFAULT_URL` was renamed to `PERMISSIONSX_REDIRECT_URL`.
- New setting was added: `PERMISSIONSX_LOGOUT_IF_DENIED`.

Reference

Automatically generated documentation:

permissionsx.models

PermissionsX - Authorization for Django.

copyright Copyright (c) 2013-2014 by Robert Pogorzelski.

license BSD, see LICENSE for more details.

class `permissionsx.models.`**`Arg`**(*argument*, *request=None*)

Resolves string to an attribute of the request object.

A wrapper class used with *P* instances to change behaviour of `Permissions.rules_evaluate()` method.

If *Arg* is passed, it will be used an argument for the method defined in the rule (i.e. the last part of the keyword argument name), e.g.:

```
P(user__has_access_to=Arg('invoice'))
```

Will translate to:

```
is_authorized = request.user.has_access_to(request.invoice)
```

class `permissionsx.models.`**`Cmp`**(*argument*, *request=None*)

Resolves string to an attribute of the request object.

A wrapper class used with *P* instances to change behaviour of `Permissions.rules_evaluate()` method.

If *Cmp* is passed, it will be used to retrieve attribute of the request object, e.g.:

```
P(user__equals_to=Cmp('invoice'))
```

Will translate to:

```
is_authorized = (request.user.equals_to == request.invoice)
```

class permissionsx.models.**P**(*children=None*, *connector=None*, *negated=False*, ***kwargs*)
Base building block for defining rules.

Based on *django.db.models.query_utils.Q* mixed with *django.utils.tree.Node*.

class permissionsx.models.**Permissions**(**args*, ***kwargs*)
Base class for defining permissions. Usage:

```
permissions = SuperuserPermissions
```

Or when there is no need of reusing permissions in other parts of the code:

```
permissions = Permissions(P(user__is_superuser=True))
```

get_rules(*request=None*, ***kwargs*)
Used for overriding `rules`.

permissionsx.contrib.django

PermissionsX - Authorization for Django.

copyright Copyright (c) 2013-2014 by Robert Pogorzelski.

license BSD, see LICENSE for more details.

permissionsx.contrib.django_debug_toolbar

permissionsx.contrib.tastypie

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index