Django-CRUM Documentation

Release 0.7.9

Nine More Minutes, Inc.

Apr 07, 2021

Contents

Django-CRUM (Current Request User Middleware) captures the current request and user in thread local storage.

It enables apps to check permissions, capture audit trails or otherwise access the current request and user without requiring the request object to be passed directly. It also offers a context manager to allow for temporarily impersonating another user.

It provides a signal to extend the built-in function for getting the current user, which could be helpful when using custom authentication methods or user models.

It is tested against:

- Django 1.11 (Python 3.5 and 3.6)
- Django 2.0 (Python 3.5, 3.6 and 3.7)
- Django 2.1 (Python 3.5, 3.6 and 3.7)
- Django 2.2 (Python 3.5, 3.6, 3.7, 3.8 and 3.9)
- Django 3.0 (Python 3.6, 3.7, 3.8 and 3.9)
- Django 3.1 (Python 3.6, 3.7, 3.8 and 3.9)
- Django 3.2 pre-release (Python 3.6, 3.7, 3.8 and 3.9)
- Django main/4.0 (Python 3.8 and 3.9)

CHAPTER 1

Installation

Install the application from PYPI:

```
pip install django-crum
```

Add `CurrentRequestUserMiddleware` to your `MIDDLEWARE` setting:

```
MIDDLEWARE += ('crum.CurrentRequestUserMiddleware',)
```

That's it!

CHAPTER 2

Usage

The *crum* package exports three functions as its public API.

2.1 get_current_request()

`get_current_request` returns the current request instance, or `None` if called outside the scope of a request.

For example, the `Comment` model below overrides its `save` method to track the IP address of each commenter:

```
from django.db import models
from crum import get_current_request

class Comment(models.Model):
    created = models.DateTimeField(auto_now_add=True)
    comment = models.TextField()
    remote_addr = models.CharField(blank=True, default='')

    def save(self, *args, **kwargs):
        request = get_current_request()
        if request and not self.remote_addr:
            self.remote_addr = request.META['REMOTE_ADDR']
        super(Comment, self).save(*args, **kwargs)
```

2.2 get_current_user()

`get_current_user` returns the user associated with the current request, or `None` if no user is available.

If using the built-in `User` model from `django.contrib.auth`, the returned value may be the special `AnonymousUser`, which won't have a primary key.

For example, the `Thing` model below records the user who created it as well as the last user who modified it:

```
from django.db import models
from crum import get_current_user

class Thing(models.Model):
    created = models.DateTimeField(auto_now_add=True)
    created_by = models.ForeignKey('auth.User', blank=True, null=True,
                                   default=None)
    modified = models.DateTimeField(auto_now=True)
    modified_by = models.ForeignKey('auth.User', blank=True, null=True,
                                    default=None)

    def save(self, *args, **kwargs):
        user = get_current_user()
        if user and not user.pk:
            user = None
        if not self.pk:
            self.created_by = user
        self.modified_by = user
        super(Thing, self).save(*args, **kwargs)
```

2.3 impersonate(user=None)

`impersonate` is a context manager used to temporarily change the current user as returned by `get_current_user`. It is typically used to perform an action on behalf of a user or disable the default behavior of `get_current_user`.

For example, a background task may need to create or update `Thing` objects when there is no active request or user (such as from a management command):

```
from crum import impersonate

def create_thing_for_user(user):
    with impersonate(user):
        # This Thing will indicated it was created by the given user.
        user_thing = Thing.objects.create()
    # But this Thing won't have a created_by user.
    other_thing = Thing.objects.create()
```

When running from within a view, `impersonate` may be used to prevent certain actions from being attributed to the requesting user:

```
from django.template.response import TemplateResponse
from crum import impersonate

def get_my_things(request):
    # Whenever this view is accessed, trigger some cleanup of Things.
    with impersonate(None):
        Thing.objects.cleanup()
    my_things = Thing.objects.filter(created_by=request.user)
    return TemplateResponse(request, 'my_things.html',
                            {'things': my_things})
```

CHAPTER 3

Signals

(New in 0.6.0) The *crum* package provides a signal to extend the capabilities of the *get_current_user()* function.

3.1 current_user_getter

The `current_user_getter` signal is dispatched for each call to `get_current_user()`. Receivers for this signal should return a tuple of `(user, priority)`. Receivers should return `None` for the user when there is no current user set, or `False` when they can not determine the current user.

The priority value which will be used to determine which response contains the current user. The response with the highest priority will be used as long as the user returned is not `False`, otherwise lower-priority responses will be used in order of next-highest priority. Built-in receivers for this signal use priorities of -10 (current request) and +10 (thread locals); any custom receivers should usually use -10 < priority < 10.

The following example demonstrates how a custom receiver could be implemented to determine the current user from an auth token passed via an HTTP header:

```
from django.dispatch import receiver
from crum import get_current_request
from crum.signals import current_user_getter

@receiver(current_user_getter)
def (sender, **kwargs):
    request = get_current_request()
    if request:
        token = request.META.get('HTTP_AUTH_TOKEN', None)
        try:
            auth_token = AuthToken.objects.get(token=token)
            return (auth_token.user, 0)
        except AuthToken.DoesNotExist:
            return (None, 0)
    return (False, 0)
```