# django-usersettings2 Documentation

***Release 0.1.5***

**Mishbah Razzaque**

**Sep 27, 2017**

# Contents


**1 django-usersettings2** **3**
- 1.1 Why would you use usersettings? . . . 3
- 1.2 Example Usage . . . 3
- 1.3 Hooking the 'usersettings' to the admin site . . . 4
- 1.4 Hooking into the current usersettings from views . . . 5
- 1.5 Custom Middleware . . . 5
- 1.6 Caching the current `UserSettings` object . . . 5
- 1.7 Install . . . 6
- 1.8 Dependencies . . . 6
- 1.9 DJANGO-CMS >= 3.0 Toolbar . . . 7
- 1.10 Documentation . . . 7

**2 Installation** **9**
- 2.1 Dependencies . . . 10

**3 Usage** **11**
- 3.1 Hooking the 'usersettings' to the admin site . . . 12
- 3.2 Hooking into the current usersettings from views . . . 12
- 3.3 Custom Middleware . . . 13
- 3.4 Caching the current `UserSettings` object . . . 13

**4 Contributing** **15**
- 4.1 Types of Contributions . . . 15
- 4.2 Get Started! . . . 16
- 4.3 Pull Request Guidelines . . . 17
- 4.4 Tips . . . 17

**5 Credits** **19**
- 5.1 Development Lead . . . 19
- 5.2 Contributors . . . 19

**6 History** **21**
- 6.1 0.1.0 (2014-09-05) . . . 21

Contents:

CHAPTER 1

# django-usersettings2

A reusable app for django, provides the ability to configure site settings via the admin interface, rather than by editing settings.py

## Why would you use usersettings?

This project is the missing extension to the Django "sites" framework, use it to store additional information for your Django-powered sites. The project structure is heavily inspired by django `sites` app, with a `one-to-one` relationship to the `Site` model.

It's best explained through examples.

## Example Usage

For example, suppose you're developing a multi-site django project i.e. using single Django installation that powers more than one site and you need to differentiate between those sites in some way.

(e.g. Site Title, Physical Location, Contact Details... etc)

Of course, you could hardcode the information in the templates and use different templates for each site. Alternatively you could configure details in your *settings.py* for each site.

A better solution would be to use `django-usersettings2`. This project accomplishes several things quite nicely:

- It lets the site producers edit all settings – for multiple sites – in a single interface (the Django admin).
- It lets the site developers use the same Django views/templates for multiple sites.

To get started, create a class that inherits from `usersettings.models.UserSettings`. Make sure to import the `UserSettings` model. Your class should live in one of your apps' models.py (or module).

Since `UserSettings` model inherit from `django.db.models.Model`, you are free to add any field you want.

Here's a simple example:

```
from django.db import models
from django.utils.translation import ugettext_lazy as _

from usersettings.models import UserSettings

class SiteSettings(UserSettings):
    site_title = models.CharField(_('Site Title'), max_length=100)
    tag_line = models.CharField(_('Tag Line'), max_length=150, blank=True)
    site_description = models.TextField(_('Site Description'), blank=True)

    ...

    class Meta:
        verbose_name = 'Site Settings'
        verbose_name_plural = 'Site Settings'
```

If you followed the Django tutorial, this shouldn't look too new to you. The only difference to normal models is that you subclass `usersettings.models.UserSettings` rather than `django.db.models.base.Model`.

## Hooking the 'usersettings' to the admin site

To make your new model editable in the admin interface, you must first create an admin class that subclasses `usersettings.admin.SettingsAdmin`. Continuing with the example model above, here's a simple corresponding `SiteSettingsAdmin` class:

```
from django.contrib import admin
from django.utils.translation import ugettext_lazy as _

from usersettings.admin import SettingsAdmin

from .models import SiteSettings


class SiteSettingsAdmin(SettingsAdmin):

    fieldsets = (
        (_('Site Title / Tag Line'), {
            'description': '...',
            'fields': ('site_title', 'tag_line',)
        }),
        ...
    )

    ...

admin.site.register(SiteSettings, SiteSettingsAdmin)
```

Since `SettingsAdmin` inherits from ModelAdmin, you'll be able to use the normal set of Django ModelAdmin properties, as appropriate to your circumstance.

Once you've registered your admin class, a new model will appear in the top-level admin list.

## Hooking into the current usersettings from views

You can use the `usersettings` in your Django views to do particular things based on the `usersettings` for the site.

Here's an example of what the a view looks like:

```
from usersettings.shortcuts import get_current_usersettings

def home(request):
    ...

    current_usersetting = get_current_usersettings()

    context = {
        'title': current_usersetting.site_title,
    }

    ...
```

## Custom Middleware

To avoid the repetitions of having to import `current_usersetting` for every view. Add `usersettings.middleware.CurrentUserSettingsMiddleware` to `MIDDLEWARE_CLASSES` The middleware sets the `usersettings` attribute on every request object, so you can use `request.usersettings` to get the current usersettings:

```
MIDDLEWARE_CLASSES=(
    ...
    'usersettings.middleware.CurrentUserSettingsMiddleware',
    ...
)
```

## Caching the current `UserSettings` object

As the `usersettings` are stored in the database, each call to `UserSettings.objects.get_current()` could result in a database query.

But just like the Django sites framework, on the first request the current usersettings is cached, and any subsequent call returns the cached data instead of hitting the database.

If for any reason you want to force a database query, you can tell Django to clear the cache using `UserSettings.objects.clear_cache()`:

```
from usersettings.shortcuts import get_usersettings_model

UserSettings = get_usersettings_model()

# First call; current usersettings fetched from database.
current_usersetting = UserSettings.objects.get_current()

# Second call; current usersettings fetched from cache.
current_usersetting = UserSettings.objects.get_current()
```

```
# Force a database query for the third call.
UserSettings.objects.clear_cache()
current_usersetting = UserSettings.objects.get_current()
```

## Install

1. Install `django-usersettings`:

   ```
   pip install django-usersettings2
   ```

2. Add `usersettings` to `INSTALLED_APPS`:

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

4. Specify the custom `UserSettings` model as the default usersettings model for your project using the `USERSETTINGS_MODEL` setting in your settings.py (required):

   ```
   USERSETTINGS_MODEL='config.SiteSettings'
   ```

5. Add `usersettings.middleware.CurrentUserSettingsMiddleware` to `MIDDLEWARE_CLASSES` (optional).

The middleware sets the `usersettings` attribute on every request object, so you can use `request.usersettings` to get the current usersettings:

```
MIDDLEWARE_CLASSES=(
    ...
    'usersettings.middleware.CurrentUserSettingsMiddleware',
    ...
)
```

6. The current usersettings are made available in the template context when your `TEMPLATE_CONTEXT_PROCESSORS` setting contains `usersettings.context_processors.usersettings`:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    'usersettings.context_processors.usersettings',
    ...
)
```

## Dependencies

django-usersettings2 requires The "sites" framework to be installed.

To enable the sites framework, follow these steps:

1. Add *django.contrib.sites* to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    ...
    'django.contrib.sites'
    ...
)
```

2. Define a `SITE_ID` setting:

```
SITE_ID = 1
```

3. Run migrate.

## DJANGO-CMS >= 3.0 Toolbar

djangocms-usersettings2 integrates `django-usersettings2` with django-cms>=3.0

This allows a site editor to add/modify all usersettings in the frontend editing mode of django CMS and provide your users with a streamlined editing experience.

## Documentation

The full documentation is at https://django-usersettings2.readthedocs.org.

CHAPTER 2

# Installation

1. Install `django-usersettings`:

```
pip install django-usersettings2
```

2. Add `usersettings` to `INSTALLED_APPS`:

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

4. Specify the custom `UserSettings` model as the default usersettings model for your project using the `USERSETTINGS_MODEL` setting in your settings.py (required):

```
USERSETTINGS_MODEL='config.SiteSettings'
```

5. Add `usersettings.middleware.CurrentUserSettingsMiddleware` to `MIDDLEWARE_CLASSES` (optional).

The middleware sets the `usersettings` attribute on every request object, so you can use `request.usersettings` to get the current usersettings:

```
MIDDLEWARE_CLASSES=(
    ...
    'usersettings.middleware.CurrentUserSettingsMiddleware',
    ...
),
```

6. The current usersettings are made available in the template context when your `TEMPLATE_CONTEXT_PROCESSORS` setting contains `usersettings.context_processors.usersettings`:

```
TEMPLATE_CONTEXT_PROCESSORS = (
    ...
```

```
    'usersettings.context_processors.usersettings',
    ...
)
```

## Dependencies

django-usersettings2 requires The "sites" framework to be installed.

To enable the sites framework, follow these steps:

1. Add *django.contrib.sites* to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    ...
    'django.contrib.sites'
    ...
)
```

2. Define a `SITE_ID` setting:

```
SITE_ID = 1
```

3. Run migrate.

CHAPTER 3

# Usage

For example, suppose you're developing a multi-site django project i.e. using single Django installation that powers more than one site and you need to differentiate between those sites in some way.

(e.g. Site Title, Physical Location, Contact Details... etc)

Of course, you could hardcode the information in the templates and use different templates for each site. Alternatively you could configure details in your *settings.py* for each site.

A better solution would be to use `django-usersettings2`. This project accomplishes several things quite nicely:

- It lets the site producers edit all settings – for multiple sites – in a single interface (the Django admin).
- It lets the site developers use the same Django views/templates for multiple sites.

To get started, create a class that inherits from `usersettings.models.UserSettings`. Make sure to import the `UserSettings` model. Your class should live in one of your apps' models.py (or module).

Since `UserSettings` model inherit from `django.db.models.Model`, you are free to add any field you want.

Here's a simple example:

```
from django.db import models
from django.utils.translation import ugettext_lazy as _

from usersettings.models import UserSettings

class SiteSettings(UserSettings):
    site_title = models.CharField(_('Site Title'), max_length=100)
    tag_line = models.CharField(_('Tag Line'), max_length=150, blank=True)
    site_description = models.TextField(_('Site Description'), blank=True)

    ...

    class Meta:
        verbose_name = 'Site Settings'
        verbose_name_plural = 'Site Settings'
```

If you followed the Django tutorial, this shouldn't look too new to you. The only difference to normal models is that you subclass `usersettings.models.UserSettings` rather than `django.db.models.base.Model`.

## Hooking the 'usersettings' to the admin site

To make your new model editable in the admin interface, you must first create an admin class that subclasses `usersettings.admin.SettingsAdmin`. Continuing with the example model above, here's a simple corresponding `SiteSettingsAdmin` class:

```
from django.contrib import admin
from django.utils.translation import ugettext_lazy as _

from usersettings.admin import SettingsAdmin

from .models import SiteSettings


class SiteSettingsAdmin(SettingsAdmin):

    fieldsets = (
        (_('Site Title / Tag Line'), {
            'description': '...',
            'fields': ('site_title', 'tag_line',)
        }),
        ...
    )

    ...

admin.site.register(SiteSettings, SiteSettingsAdmin)
```

Since `SettingsAdmin` inherits from ModelAdmin, you'll be able to use the normal set of Django ModelAdmin properties, as appropriate to your circumstance.

Once you've registered your admin class, a new model will appear in the top-level admin list.

## Hooking into the current usersettings from views

You can use the `usersettings` in your Django views to do particular things based on the `usersettings` for the site.

Here's an example of what the a view looks like:

```
from usersettings.shortcuts import get_current_usersettings

def home(request):
    ...

    current_usersetting = get_current_usersettings()

    context = {
        'title': current_usersetting.site_title,
    }

    ...
```

## Custom Middleware

To avoid the repetitions of having to import `current_usersetting` for every view. Add `usersettings.middleware.CurrentUserSettingsMiddleware` to `MIDDLEWARE_CLASSES` The middleware sets the `usersettings` attribute on every request object, so you can use `request.usersettings` to get the current usersettings:

```
MIDDLEWARE_CLASSES=(
    ...
    'usersettings.middleware.CurrentUserSettingsMiddleware',
    ...
)
```

## Caching the current `UserSettings` object

As the `usersettings` are stored in the database, each call to `UserSettings.objects.get_current()` could result in a database query.

But just like the Django sites framework, on the first request the current usersettings is cached, and any subsequent call returns the cached data instead of hitting the database.

If for any reason you want to force a database query, you can tell Django to clear the cache using `UserSettings.objects.clear_cache()`:

```
# First call; current usersettings fetched from database.
current_usersetting = UserSettings.objects.get_current()
# ...

# Second call; current usersettings fetched from cache.
current_usersetting = UserSettings.objects.get_current()
# ...

# Force a database query for the third call.
UserSettings.objects.clear_cache()
current_usersetting = UserSettings.objects.get_current()
```

CHAPTER 4

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## Types of Contributions

### Report Bugs

Report bugs at https://github.com/mishbahr/django-usersettings2/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

### Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

### Write Documentation

django-usersettings2 could always use more documentation, whether as part of the official django-usersettings2 docs, in docstrings, or even on the web in blog posts, articles, and such.

### Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mishbahr/django-usersettings2/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## Get Started!

Ready to contribute? Here's how to set up *django-usersettings2* for local development.

1. Fork the *django-usersettings2* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-usersettings2.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-usersettings2
$ cd django-usersettings2/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 usersettings tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/mishbahr/django-usersettings2/pull_requests and make sure that the tests pass for all supported Python versions.

## Tips

To run a subset of tests:

```
$ python -m unittest tests.test_usersettings
```

CHAPTER 5

# Credits

## Development Lead

- Mishbah Razzaque <mishbahx@gmail.com>

## Contributors

- Basil Shubin <basil.shubin@gmail.com>

CHAPTER 6

# History

## 0.1.0 (2014-09-05)

- First release on PyPI.