Django Livesettings Documentation

Release 1.4.9

Bruce Kroeze

Jul 01, 2017

Contents

Contents:

CHAPTER 1

About

Django-Livesettings is a project split from the Satchmo Project. It provides the ability to configure settings via an admin interface, rather than by editing `settings.py`. In addition, livesettings allows you to set sane defaults so that your site can be perfectly functional without any changes. Livesettings uses caching to make sure this has minimal impact on your site's performance.

Finally, if you wish to lock down your site and disable the settings, you can export your livesettings and store them in your `settings.py`. This allows you have flexibility in deciding how various users interact with your app.

Livesettings supports several types of input choices:

- Boolean
- Decimal
- Duration
- Float
- Integer
- Positive Integer (non negative)
- String
- Long string
- Multiple strings
- Long multiple strings
- Module values
- Password

Livesettings has been used for many years in the satchmo project and is considered stable and production ready.

CHAPTER 2

Installation

Requirements

- Python 2.5+, 2.6+ or 2.7+
- Django 1.4+ or 1.5+
- Django-Keyedcache

Installing Livesettings

After the dependencies have been installed, you can install the latest livesettings, using:

```
pip install -e hg+http://bitbucket.org/bkroeze/django-livesettings/#egg=django-
↪livesettings
```

Add livesettings to your installed apps in `settings.py`:

```
INSTALLED_APPS = (
    ...
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    'livesettings',
    'myapp'
    ...
)
```

It is high recommended to configure a global cache (like *MemcachedCache*) for multiprocess servers! Otherwise the processes would not be notified about new values with the default *LocMemCache*. The default configuration is safe for a debug server (manage.py runserver).

Add it to your `urls.py`:

```
urlpatterns = patterns('',
    ...
    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
    url(r'^settings/', include('livesettings.urls')),
    ...
)
```

Execute a syncdb to create the required tables:

```
python manage.py syncdb
```

CHAPTER 3

Usage

An example project is in the directory `test-project`. It's beginning isidentical to the following description and is a useful example for integrating livesettings into your app.

Creating Config.py

In order to use livesettings, you will need to create a `config.py` in your django application. For this example, we will create a `config.py` file in the 'test-project/localsite' directory.

Example: "For this specific app, we want to allow an admin user to control how many images are displayed on the front page of our site." We will create the following `config.py`:

```
from livesettings.functions import config_register
from livesettings.values import ConfigurationGroup, PositiveIntegerValue,␣
↪MultipleStringValue
from django.utils.translation import ugettext_lazy as _

# First, setup a grup to hold all our possible configs
MYAPP_GROUP = ConfigurationGroup(
    'MyApp',               # key: internal name of the group to be created
    _('My App Settings'),  # name: verbose name which can be automatically translated
    ordering=0             # ordering: order of group in the list (default is 1)
    )

# Now, add our number of images to display value
# If a user doesn't enter a value, default to 5
config_register(PositiveIntegerValue(
    MYAPP_GROUP,           # group: object of ConfigurationGroup created above
        'NUM_IMAGES',      # key:   internal name of the configuration value to be␣
↪created
        description = _('Number of images to display'),              # label for the␣
↪value
        help_text = _("How many images to display on front page."),  # help text
        default = 5        # value used if it have not been modified by the user␣
↪interface
```

```
    ))

# Another example of allowing the user to select from several values
config_register(MultipleStringValue(
        MYAPP_GROUP,
        'MEASUREMENT_SYSTEM',
        description=_("Measurement System"),
        help_text=_("Default measurement system to use."),
        choices=[('metric',_('Metric')),
                    ('imperial',_('Imperial'))],
        default="imperial"
    ))
```

In order to activate this file, add the following line to your `models.py`:

```
import config
```

You can now see the results of your work by running the dev server and going to settings

```
python manage.py runserver
```

Dislayed values can be limited to a configuration group by the url. For example we want to do experiments with configuration group *MyApp* only: group settings :: where *MyApp* is the key name of the displayed group.

More examples for all implemented types of ..Values can be found in `test-project/localsite/config.py`:: including configuration groups which are enabled or disabled based on modules selected in the form. You can review examples by:

cd test-project python manage.py runserver

and browse <*http://127.0.0.1:8000/settings/*>.

Accessing your value in a view

Now that you have been able to set a value and allow a user to change it, the next step is to access it from a view.

In a `views.py`, you can use the config_value function to get access to the value. Here is a very simple view that passes the value to your template:

```
from django.shortcuts import render_to_response
from livesettings import config_value

def index(request):
    image_count = config_value('MyApp','NUM_IMAGES')
    # Note, the measurement_system will return a list of selected values
    # in this case, we use the first one
    measurement_system = config_value('MyApp','MEASUREMENT_SYSTEM')
    return render_to_response('myapp/index.html',
                              {'image_count': image_count,
                              'measurement_system': measurement_system[0]})
```

Using the value in your `index.html` is straightforward:

```
<p>Test page</p>
<p>You want to show {{image_count}} pictures and use the {{measurement_system}}␣
↪system.</p>
```

Security and Restricting Access to Livesettings

In order to give non-superusers access to the /settings/ views, open Django Admin Auth screen and give the user or to its group the permission *livesettings|setting|Can change settting*. The same permission is needed to view the form and submit. Permissions for insert or delete and any permissions for "long setting" are ignored.

Note: Superusers will have access to this setting without enabling any specific permissions.

Note: Because of the security significance of livesettings, all views in livesettings support CSRF regardless of whether or not the CsrfViewMiddleware is enabled or disabled.

If you want to save a sensitive information to livesettings on production site (e.g. a password for logging into other web service) it is recommended not to grant permissions to livesettings to users which are logging in everyday. The most secure method is to export the settings and disable web access to livesettings as described below. Exporting settings itself is allowed only by the superuser.

Password values should be declared by *PasswordValue(... render_value=False)* that replaces password characters by asterisks in the browser. (Though hidden to a human observer, password is still accessible by attacker's javascripts or by connection eavesdropping.)

Exporting Settings

Settings can be exported by the http://127.0.0.1:8000/settings/export/ . After exporting the file, the entire output can be manually copied and pasted to `settings.py` in order to deploy configuration to more sites or to entirely prevent further changes and reading by web browser. If you restrict DB access to the settings, all of the livesettings_* tables will be unused.

Here is a simple example of what the extract will look like:

```
LIVESETTINGS_OPTIONS = \
{   1: {   'DB': False,
           'SETTINGS': {   u'MyApp': {   u'DECIMAL_TEST': u'34.0923443',
                                         u'MEASUREMENT_SYSTEM': u'["metric"]',
                                         u'STRING_TEST': u'Orange'}}}}
```

In order to restrict or enable DB access, use the following line in your settings:

```
'DB': True,     # or False
```

If you have multiple sites, they can be manually combined in the file as well, where "1:" is to be repeatedly replaced by site id.

Exporting settings requires to be a superuser in Django.

Notes

If you use logging with the level DEBUG in your application, prevent increasing of logging level of keyedcache by configuring it in settings.py:

```
import logging
logging.getLogger('keyedcache').setLevel(logging.INFO)
```

Next Steps

The rest of the various livesettings types can be used in a similar manner. You can review the satchmo code for more advanced examples.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search