django-settings-custom Documentation

Release 1.1.0

Thomas Marques

Oct 09, 2019

Contents:

CHAPTER 1

django-settings-custom

A Django interactive command for configuration file generation.

1.1 Getting It

The project is on PyPI (https://pypi.org/project/django-settings-custom/)

```
pip install django-settings-custom
```

1.2 Installing It

To enable `django_settings_custom` in your project you need to add it to `INSTALLED_APPS` in your projects `settings.py` file:

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

1.3 Using It

Create a template for your target conf.ini like

```
[DATABASE]
NAME = { USER_VALUE }
HOST = { USER_VALUE }
PORT = { USER_VALUE }

[DATABASE_CREDENTIALS]
USER = { USER_VALUE }
PASSWORD = { ENCRYPTED_USER_VALUE }

[DJANGO]
KEY = { DJANGO_SECRET_KEY }

# A constant field
[LDAP]
URL = 'ldaps://myldap'
```

1.3.1 Configure in Django settings

Add `settings.py` file

```
SETTINGS_TEMPLATE_FILE = 'PATH_TO_YOUR_TEMPLATE_CONFIGURATION_FILE'
SETTINGS_FILE_PATH = 'TARGET_FOR_CONFIGURATION_FILE'
```

Launch in command line

```
python manage.py generate_settings
```

1.3.2 Or all in command line

```
python manage.py generate_settings path/to/template/settings.ini target/path/of/
↪settings.ini
```

1.4 Results

The command ask user to fill missing values from template:

```
[user@localhost a_project]$ ./manage.py generate_conf
** Configuration file generation: **

** Configuration file generation: **
Do you want to generate the secret key for Django ? (Y/n) : y
Django secret key generated

** Enter values for configuration file content **

Value for [DATABASE] NAME: database_name
Value for [DATABASE] HOST: database_host
Value for [DATABASE] PORT: 900
Value for [DATABASE_CREDENTIALS] USER: my_user
```

(continues on next page)

(continued from previous page)

```
Value for [DATABASE_CREDENTIALS] PASSWORD (will be encrypted):

Writing file at /home/user/a_project/conf.ini:
Configuration file successfully generated.
[user@localhost a_project]$
```

It generates the file /home/user/a_project/conf.ini:

```
[DATABASE]
NAME = database_name
HOST = database_host
PORT = 900

[DATABASE_CREDENTIALS]
USER = my_user
PASSWORD = JbAwLj5Zwz8lMrvcUZq5sP/v6eaUFY5E7U8Fmg63vxI=

# A constant field
[LDAP]
URL = 'ldaps://monldap'

[DJANGO]
KEY = w)r13ne4=id9_8xdojir)3)%%5m3r$co#jwj_)4d*_%%!0+f#sro
```

And to decrypt values in your code (in settings.py for example), you may use `django_settings_custom.encryption.decrypt` :

```
import configparser
from django_settings_custom import encryption

config = configparser.RawConfigParser()
config.read(SETTINGS_FILE_PATH)
database_password = encryption.decrypt(config.get('DATABASE_CREDENTIALS', 'PASSWORD'))
```

To decrypt values, the function uses the django SECRET_KEY (must be set before).

1.5 Miscellaneous

1.5.1 If you don't want to use Django settings

If you don't want to add specific variables to your Django settings file, you can inherit `generate_settings.Command` to specify command options :

```
from django_settings_custom.management.commands import generate_settings


class Command(generate_settings.Command):
    settings_template_file = 'The/settings/template/file_path.ini'
    settings_file_path = 'The/target/settings/file_path.ini'
```

1.5.2 Adding custom tag

To add a custom tag, you can inherit `generate_settings.Command` and override the method `get_value` :

```
import random
from django_settings_custom.management.commands import generate_settings


class Command(generate_settings.Command):

    def get_value(self, section, key, value_type):
        if value_type == 'RANDOM_VALUE':
            return random.uniform(0, 100)
        return super(Command, self).get_value(section, key, value_type)
```

Or a little more complex example :

```
from django.core.management.base import CommandError
from django_settings_custom.management.commands import generate_settings


class Command(generate_settings.Command):

    def get_value(self, section, key, value_type):
        int_less_10 = value_type == 'INT_LESS_THAN_10'
        if int_less_10:
            value_type = 'USER_VALUE'
        value = super(Command, self).get_value(section, key, value_type)
        if int_less_10:
            try:
                value = int(value)
                if value >= 10:
                    raise CommandError('This field needs an int less than 10.')
            except ValueError:
                raise CommandError('This field needs an int.')
        return value
```

1.6 Documentation

CHAPTER 2

API Documentation

2.1 Command

Documentation corresponding to Command class of generate_settings

class django_settings_custom.management.commands.generate_settings.**Command**(**argc*, ***kwargs*)

A Django interactive command for configuration file generation.

Example

python manage.py generate_settings path/to/template/settings.ini target/path/of/settings.ini'

settings_template_file
Path to the settings template file.

Type str

settings_file_path
Target path for the created settings file.

Type str

force_secret_key
Generate SECRET_KEY without asking ?

Type bool

add_arguments(*parser*)
Add custom arguments to the command. See python manage.py generate_settings –help.

get_value(*section*, *key*, *value_type*)
Get a value for the [section] key passed as parameter.

Parameters

- **section** (*str*) – Section in the configuration file.

- **key** (*str*) – Key in the configuration file.
- **value_type** (*str*) – Value type read in template, must be "DJANGO_SECRET_KEY", "USER_VALUE" or "ENCRYPTED_USER_VALUE".

Returns Value for the [section] key

Return type int or str

handle(**args*, ***options*)
Command core.

2.2 Encryption

Documentation corresponding to encryption.py

django_settings_custom.encryption.**decrypt**(*source*, *secret_key=None*)
Decrypt the source with the key passed as parameter.

Parameters

- **source** (*str or byte string*) – A string or a bytes array to decrypt.
- **secret_key** (*str*) – The key for encryption, or None if you want use the SECRET_KEY.

Returns Decrypted value.

Return type str

If the secret_key is not provided, it uses Django settings.SECRET_KEY.

django_settings_custom.encryption.**encrypt**(*source*, *secret_key=None*)
Encrypt the source with the key passed as parameter.

Parameters

- **source** (*str or byte string*) – A string or a bytes array to encrypt.
- **secret_key** (*str*) – The key for encryption, or None if you want use the SECRET_KEY.

Returns Encrypted value.

Return type str

If the secret_key is not provided, it uses Django settings.SECRET_KEY.

Python Module Index

d

e

Index