Django Stuff Documentation

Release 0.7.1

Rafael Henter

Sep 30, 2020

Contents

Django Stuff is a collection of tools and utilities to make your development with Django simpler.

CHAPTER 1

Features

- TimeStamp and History models to giving you information like when your record wore created/updated and History Changes
- UUID Model as primary key or not instead of sequence ID.
- Serializer model to return a dict with all data of your django instance.
- Signals model to add any task before or after you save, update or delete your model
- And many other stuff. For more information, see our documentation following the links below.
- Backend to Login using email or username

CHAPTER 2

Quickstart

2.1 Installation

2.1.1 Requirements

- Python 3.x
- Django 1.11 or later

2.1.2 Install

You can get Django Stuff by using pip:

```
$ pip install django-stuff
```

If you want to install it from source, grab the git repository from Gitlab and run setup.py:

```
$ git clone git@github.com:rhenter/django_stuff.git
$ cd django_stuff
$ python setup.py install
```

2.1.3 Enable

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

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

CHAPTER 3

User Guide

3.1 Backends

3.1.1 EmailOrUsernameModelBackend

Backend to login using email or username.

Usage:

Add before the default ModelBackend

```
AUTHENTICATION_BACKENDS = (
    'django_stuff.backends.auth.EmailOrUsernameModelBackend',
    'django.contrib.auth.backends.ModelBackend'
)
```

3.2 Fields

3.2.1 UUIDPrimaryKeyField

Field to use UUID as Primary Key of your model instead of the sequential

usage:

```
from django_stuff.fields import UUIDPrimaryKeyField

...
class TestModel(models.Model):
    id = UUIDPrimaryKeyField()
    ...
```

3.2.2 CharFieldDigitsOnly

Field to use only digits but with zero on left.

usage:

```
from django_stuff.fields import CharFieldDigitsOnly

...
class TestModel(models.Model):
    code = CharFieldDigitsOnly(max_length=10)
    ...
```

3.3 Forms

Like django-localflavors the CPFField and CNPJField are validators to Brazilian CPF and CNPJ with a big difference, It removes repeated sequences and undue values.

Usage:

Your Just need to add your form class. Example using ModelForm:

3.3.1 CPFField

```
from django_stuff.forms import CPFField

...

class TestForm(forms.ModelForm):
    class Meta:
        model = YourModel

    fields = [..., 'cpf']
    widgets = {
        'cpf': CNPJField(),
    }
```

3.3.2 CNPJField

```
from django_stuff.forms import CNPJField

...

class TestForm(forms.ModelForm):
    class Meta:
        model = YourModel

    fields = [..., 'cnpj']
    widgets = {
        'cnpj': CNPJField(),
    }
```

3.4 Models

3.4.1 Generic Models

You just need to add to your template to get the behaviors below. Use as many models as you want.

SlugModel

Model with a slugField already implemented

Usage:

```
class YourModel(SlugModel)
    ...
```

SerializerModel

Model with serialize method making possible serializer your instance data returning a dict.

Usage:

```
from django_stuff.models import SerializerModel
...

class YourModel(SerializerModel)
    ...
```

Example of a instance from a Model using the SerializerModel

```
instance.serialize()
{
    'id': 1,
    'name': 'Test'
}
```

TimestampedModel

Model with created_at and updated_at fields to let you know when your instance wore created and updated

Usage:

```
from django_stuff.models import TimestampedModel
...

class YourModel(TimestampedModel)
    ...
```

UUIDModel

Model with UUIDPrimaryKeyField already implemented

Usage:

```
from django_stuff.models import UUIDModel
...

class YourModel(UUIDModel)
    ...
```

3.4.2 History Model

Model to save changes to specific fields any time the template is saved

Required fields

- history_model

  Field with the model where will save the changes

- history_parent_field_name

  Field used in the foreign key in the template where the changes will be stored. The default value is **parent**

Example

```
from django_stuff.models import HistoryModel

class HistoryTestModel(models.Model):
    parent = models.ForeignKey('TestHistoryModel', related_name='history', on_
↪delete=models.CASCADE)
    name = models.CharField(max_length=128)
    email = models.CharField(max_length=32, default='example@example.com')

class TestHistoryModel(HistoryModel):
    history_model = HistoryTestModel
    name = models.CharField(max_length=128)
    email = models.CharField(max_length=32, default='example@example.com')
    description = models.CharField(max_length=32,blank=True)
```

Note: In this example, only the **name** and **email** will only be saved in the change history.

3.4.3 Signals Model

Model that makes it possible to add any task before or after you save, update or delete your model, just adding a method in your model

Usage:

Pre-save

Note: This will be made before you save your model

```
from django_stuff.models import SignalsModel
...

class YourModel(SignalsModel)
    ...
    def pre_save(self):
        do_something()
```

Pos-save

Note: This will be made after you save your model

```
from django_stuff.models import SignalsModel
...

class YourModel(SignalsModel)
    ...
    def pos_save(self):
        do_something()
```

Pre-delete

Note: This will be made before you delete your model

```
from django_stuff.models import SignalsModel
...

class YourModel(SignalsModel)
    ...
    def pre_delete(self):
        do_something()
```

Pos-delete

Note: This will be made after you delete your model

```
from django_stuff.models import SignalsModel
...

class YourModel(SignalsModel)
    ...
    def pos_delete(self):
        do_something()
```

3.5 Utils

3.5.1 generate_md5_hashcode

Generates an MD5 hash code from a key word

Usage:

```
In[0]: from django_stuff.utils import generate_md5_hashcode
In[1]: generate_md5_hashcode('test')
Out[1]: '39df5136522809aafdf726f1a8a7d00d'
```

3.5.2 generate_random_code

Generates an random code with a specific length

Usage:

```
In[0]: from django_stuff.utils import generate_random_code
In[1]: generate_random_code(16)     # with 16 length
Out[1]: 'U7Q2M1FW79WIGSW0'
In[2]: generate_random_code(10)     # with 10 length
Out[2]: 'D2U2PHUSBD'
```

3.5.3 generate_datetime

Generates an random datetime

Usage:

```
In[0]: from django_stuff.utils import generate_datetime
In[1]: generate_datetime()
Out[1]: datetime.datetime(1995, 4, 6, 21, 42, 32, 955163)
In[2]: generate_datetime(min_year=2019)     # with min_year
Out[2]: datetime.datetime(2019, 9, 26, 1, 17, 38, 303408)
```

3.5.4 is_equal

Checks if the word are formed with the same character

Usage:

```
In[0]: from django_stuff.utils import is_equal
In[1]: is_equal('asdfgh')
Out[1]: False
In[2]: is_equal('aaaaaaaaaa')
Out[2]: True
```

3.5.5 sanitize_digits

Removes all non digits characters from a string

Usage:

```
In[0]: from django_stuff.utils import sanitize_digits
In[1]: sanitize_digits('123.123.123-23')
Out[1]: '12312312323'
In[2]: sanitize_digits('123ASDF')
Out[2]: '123'
```

3.5.6 validate_cpf

Validates CPF code and return the original number

Ps: doesn't matter if you use a masked number or not

Usage:

```
In[0]: from django_stuff.utils import validate_cpf
In[1]: validate_cnpj('01212312312')   # Invalid
Out[1]: False
In[2]: validate_cpf('062.265.326-10') # Valid
Out[2]: '062.265.326-10'
```

3.5.7 validate_cnpj

Validates CNPJ code and return the original number

Ps: doesn't matter if you use a masked number or not

Usage:

```
In[0]: from django_stuff.utils import validate_cnpj
In[1]: validate_cnpj('12345123/000000')      # Invalid
Out[1]: False
In[2]: validate_cnpj('61.553.678/0001-96')   # Valid
Out[2]: '61.553.678/0001-96'
```

CHAPTER 4

Development

4.1 Development Installation

4.1.1 Requirements

- Python 3.x
- Django 1.11 or later

4.1.2 Development install

After forking or checking out:

```
$ cd django-stuff/
$ pip install -r requirements-dev.txt
$ pre-commit install
```

The requirements-dev are only used for development, so we can easily install/track dependencies required to run the tests using continuous integration platforms.

The official entrypoint for distritubution is the `requirements.txt` which contains the minimum requirements to execute the tests.

4.1.3 Running tests

```
$ make test
```

or:

```
$ cd test-django-project/
$ py.test -vv -s
```

4.1.4 Generating documentation

```
$ cd docs/
$ make html
```

4.2 Release

To release a new version, a few steps are required:

- Add entry to `CHANGES.rst` and documentation
- Review changes in test requirements `requirements.txt`
- Test build with `make build`
- Commit and push changes
- Release with `make release`

CHAPTER 5

Downloads

- PDF
- Epub

CHAPTER 6

Other

6.1 Changelog

6.1.1 0.7.2

- Update delete with force hard_delete

6.1.2 0.7.1

- Rename SoftDeleteManager to SoftDeleteSignalsManager

6.1.3 0.7.0

- Add restore method to Model and Manager to restore a record

6.1.4 0.6.2

- Add Missing Argument to delete on softdelete manager

6.1.5 0.6.1

- Fix tests with Signal using create method

6.1.6 0.6.0

- Add SoftDeleteSignalModel and Manager to apply Soft delete

6.1.7 0.5.1

- Change ugettext_lazy to gettext_lazy cause django 3.0 support

6.1.8 0.5.0

- Add sequential values on CPF to set as invalid on CPF validation

6.1.9 0.4.3

- Add CPF and CNPJ validation when the values wore equals

6.1.10 0.4.2

- Remove unnecessary code

6.1.11 0.4.1

- Fix tests and remove Manager.

6.1.12 0.4.0

- Add trigger options on save

6.1.13 0.3.0

- Refactor Utils
- Add CPF and CNPJ generators

6.1.14 0.2.1

- Remove Swagger render.

6.1.15 0.2.0

- Add swagger render

6.1.16 0.1.12

- Add remove special characters function

6.1.17 0.1.12

- Add random datetime generator

6.1.18 0.1.11

- BugFix: Change how to get a new version from changes

6.1.19 0.1.10

- BugFix: Get version from changes

6.1.20 0.1.9

- Finish all base documentation

6.1.21 0.1.8

- Update Readme
- Increase test coverage

6.1.22 0.1.7

- Updates functions names on utlils to be more clear
- Documentation: Add forms, fields and utils

6.1.23 0.1.6

- BugFix: Models SignalsModel and HistoryModel
- Fix tests
- Increase tests coverage

6.1.24 0.1.5

- BugFix: Models SignalsModel and HistoryModel
- Fix tests
- Increase tests coverage

6.1.25 0.1.5

- Add a centered version command
- Add Sphinx docs base
- Update dev requirements with Sphinx

6.1.26 0.1.4

- Update documentation

6.1.27 0.1.3

- Add noqa imports to models

6.1.28 0.1.2

- Add license on setup.py

6.1.29 0.1.1

- Refactor Structure
- Fix CI
- Add new requirements

6.1.30 0.1.0

- Initial release