Django Simple User Documentation

Release 0.1.2

Valery Melou

May 13, 2019

Contents

1 Django Simple User **3**
- 1.1 Documentation . . . 3
- 1.2 Quickstart . . . 3
- 1.3 Features . . . 3
- 1.4 Usage . . . 4
- 1.5 Bonuses . . . 4
- 1.6 Note . . . 4
- 1.7 Running Tests . . . 4
- 1.8 Credits . . . 5

2 Installation **7**

3 Usage **9**
- 3.1 Default User model of Django but without `first_name` and `last_name` fields. . . . 9

4 Contributing **11**
- 4.1 Types of Contributions . . . 11
- 4.2 Get Started! . . . 12
- 4.3 Pull Request Guidelines . . . 13
- 4.4 Tips . . . 13

5 Credits **15**
- 5.1 Development Lead . . . 15
- 5.2 Contributors . . . 15

6 History **17**
- 6.1 0.1.0 (2018-05-31) . . . 17

Contents:

CHAPTER 1

Django Simple User

Provides a simple abstract User model without first and last name fields.

1.1 Documentation

The full documentation is available at https://django-simple-user.readthedocs.io.

1.2 Quickstart

Install Django Simple User:

```
pip install django-simple-user
```

1.3 Features

- Abstract User model without the fields `first_name` and `last_name`
- Abstract User model without the fields `first_name`, `last_name` and `email`
- Abstract User model without the fields `first_name`, `last_name` and `username`

1.4 Usage

If you are like me, you might have found yourself extending `django.contrib.auth.base_user.AbstractBaseUser` just to remove the `first_name` and `last_name` fields from your final User model. I was tired of doing that and I created this package. It defines an abstract User model without those fields so that you can just inherit from it to create your User model. Let's suppose you have a `users` app with a `models.py` file containing the following:

```
from simple_user.models import AbstractUser


class User(AbstractUser):
    pass
```

Then in your settings file:

```
AUTH_USER_MODEL = 'users.User'
```

This will create a User model with all the fields that happen to exists in the default Django's User model except `first_name` and `last_name` and use it for authentication in your app.

1.5 Bonuses

There are some cases where you only want `username` or `email` but not both fields in your model. Django Simple User has got you covered. You can use `SimpleUserWithUsername` or `SimpleUserWithEmail` to achieve each scenario.

For a User model with `username` as identifying field but without `first_name`, `last_name` and `email` do:

```
from simple_user.models import SimpleUserWithUsername


class User(SimpleUserWithUsername):
    pass
```

For a User model with `email` as identifying field but without `first_name`, `last_name` and `username` do:

```
from simple_user.models import SimpleUserWithUsername


class User(SimpleUserWithEmail):
    pass
```

1.6 Note

Always remember to set your extended User model as the authentication User model by setting `AUTH_USER_MODEL` in your settings file.

1.7 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.8 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-simple-user
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-simple-user
$ pip install django-simple-user
```

CHAPTER 3

Usage

You can start using Django Simple User right after installing it. No need to add it to your *INSTALLED_APPS*:

3.1 Default User model of Django but without `first_name` and `last_name` fields.

You can simply remove the first and last name fields from the default User model of Django by doing the following:

:: from simple_user.models import AbstractUser

class User(AbstractUser): pass

And set it as your `AUTH_USER_MODEL` in `settings.py`.

:: AUTH_USER_MODEL = your_users_app_label.User

You can even add your own custom fields to the new User model:

:: from django.db import models

from simple_user.models import AbstractUser

class User(AbstractUser): name = models.CharField('Name', max_length=100) birth_date = models.DateField('birth date', null=True, blank=True)

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/valerymelou/django-simple-user/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/valerymelou/django-simple-user/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django-simple-user.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-simple-user
$ cd django-simple-user/
$ 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 simple_user 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.

4.3 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/valerymelou/django-simple-user/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

CHAPTER 5

Credits

5.1 Development Lead

- Valery Melou <valerymelou@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2018-05-31)

- First release on PyPI.