django-random-filestorage Documentation

Release 0.1.0

Erik Romijn

Nov 13, 2017

Contents

Contents:

CHAPTER 1

django-random-filestorage

Django-random-filestorage is a Django storage class that assigns random filenames to all stored files.

If a user uploads a file named *foo.txt*, it will be stored as *<60 random characters>.txt*. In cases where you refer users to uploaded files or images directly, this will prevent them from finding other files, which they may not be authorised to see, by guessing the original names used by your users.

1.1 Documentation

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

1.2 Security warning

Warning: Never use django-random-filestorage for cases where the uploaded files may contain links, such as PDF files. In that case, the secrecy of your URLs can be compromised by being leaked through the referer header, as Dropbox discovered in May: https://blog.dropbox.com/2014/05/web-vulnerability-affecting-shared-links/

1.3 Quickstart

Install django-random-filestorage:

```
pip install django-random-filestorage
```

Then use it in your entire Django project:

```
DEFAULT_FILE_STORAGE="randomfilestorage.storage.RandomFileSystemStorage"
```

Or, set it on a specific field:

```
from randomfilestorage.storage import RandomFileSystemStorage

random_storage = RandomFileSystemStorage(location='/media/my_files')
class Example(models.Model):
    my_file = FileField(storage=random_storage)
```

1.4 Why would you want to do this?

Let's say you have an app that manages all ice cream recipes you sell in your shop. Some of your recipes contain secret ingredients, and are therefore only available to a small set of trusted users. We'll look at two icecreams: strawberry, which has a fairly standard and non-secret recipe, and jellyfish, which is very secret.

The recipes are stored in PDFs, which are uploaded into a Django app that uses a FileField. As Django suggests, the media directory is directly accessible through nginx or some other simple web server. So a user which is authorised to see the strawberry recipe, will be sent to a PDF like `http://example.com/media/recipes/strawberry.pdf`. They will not see jellyfish in their list of recipes, as it's too secret.

However, given that the user knows that you sell jellyfish too, they can simply find that recipe on `http://example.com/media/recipes/jellyfish.pdf`! There are many cases where names of documents, with differing access levels, are in some way predictable. Dates are another predictable example. And filenames in FileFields are derived from the original filename the user chose.

By making these filenames random, the person who can access `http://example.com/media/recipes/ZXcAoL4wmhisYlBvHLoyt3fwfMXsMiVvgQTQOb40zOQIdag7KbEU5sy9b6GW.pdf` will not be able to guess that the jellyfish recipe is available on `http://example.com/media/recipes/qPRCEVAJd1RQvkd9OTTeY4hruW0Jvy5Qq0YIVtWPrwGWMgmUogYO8aPVRCxC.pdf`.

1.5 What issues are not resolved?

Once a user knows the random string that was used to name the file, they could provide the link to others. Then again, they could just as well download the file and provide it to others in some other way.

If you would like stricter control over who accesses certain files, you'll have to prevent direct access to (part of) the media directory. You can serve those files through a Django view instead, but this comes at an additional performance cost. A more performant but more complex alternative is to use Apache sendfile or nginx X-accel.

CHAPTER 2

Installation

At the command line, ideally in a virtualenv:

```
$ pip install django-random-filestorage
```

Or if you don't have a virtualenv or pip but only setuptools:

```
$ easy_install django-random-filestorage
```

CHAPTER 3

Usage

To use django-random-filestorage, you'll have to configure it as the storage backend.

This can be done for all relevant fields that do not have an explicit storage set up, with the `DEFAULT_FILE_STORAGE` setting in Django:

```
DEFAULT_FILE_STORAGE="randomfilestorage.storage.RandomFileSystemStorage"
```

Or, you can set it on a specific field:

```
from randomfilestorage.storage import RandomFileSystemStorage

random_storage = RandomFileSystemStorage(location='/media/my_files')
class Example(models.Model):
    my_file = FileField(storage=random_storage)
```

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/erikr/django-random-filestorage/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-random-filestorage could always use more documentation, whether as part of the official django-random-filestorage 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/erikr/django-random-filestorage/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-random-filestorage* for local development.

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

```
$ git clone git@github.com:your_name_here/django-random-filestorage.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-random-filestorage
$ cd django-random-filestorage/
$ 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 randomfilestorage 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/erikr/django-random-filestorage/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_storage
```

CHAPTER 5

Credits

5.1 Development Lead

- Erik Romijn <eromijn@solidlinks.nl>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2014-02-23)

- First release on PyPI.