django-field-history Documentation

Release 0.6.0

Grant McConnaughey

Sep 27, 2017

Contents

Contents:

CHAPTER 1

django-field-history

A Django app to track changes to a model field. For Python 2.7/3.2+ and Django 1.7+.

Documentation

The full documentation is at https://django-field-history.readthedocs.org.

Features

- Keeps a history of all changes to a particular model's field.
- Stores the field's name, value, date and time of change, and the user that changed it.
- Works with all model field types (except `ManyToManyField`).

Quickstart

Install django-field-history:

```
pip install django-field-history
```

Be sure to put it in INSTALLED_APPS.

```
INSTALLED_APPS = [
    # other apps...
    'field_history',
]
```

Then add it to your models.

```python
from field_history.tracker import FieldHistoryTracker

class PizzaOrder(models.Model):
    STATUS_CHOICES = (
        ('ORDERED', 'Ordered'),
        ('COOKING', 'Cooking'),
        ('COMPLETE', 'Complete'),
    )
    status = models.CharField(max_length=64, choices=STATUS_CHOICES)

    field_history = FieldHistoryTracker(['status'])
```

Now each time you change the order's status field information about that change will be stored in the database.

```python
from field_history.models import FieldHistory

# No FieldHistory objects yet
assert FieldHistory.objects.count() == 0

# Creating an object will make one
pizza_order = PizzaOrder.objects.create(status='ORDERED')
assert FieldHistory.objects.count() == 1

# This object has some fields on it
history = FieldHistory.objects.get()
assert history.object == pizza_order
assert history.field_name == 'status'
assert history.field_value == 'ORDERED'
assert history.date_created is not None

# You can query FieldHistory using the get_{field_name}_history()
# method added to your model
histories = pizza_order.get_status_history()
assert list(FieldHistory.objects.all()) == list(histories)

# Or using the custom FieldHistory manager
histories2 = FieldHistory.objects.get_for_model_and_field(pizza_order, 'status')
assert list(histories) == list(histories2)

# Updating that particular field creates a new FieldHistory
pizza_order.status = 'COOKING'
pizza_order.save()
assert FieldHistory.objects.count() == 2

updated_history = histories.latest()
assert updated_history.object == pizza_order
assert updated_history.field_name == 'status'
assert updated_history.field_value == 'COOKING'
assert updated_history.date_created is not None
```

Management Commands

django-field-history comes with a few management commands.

createinitialfieldhistory

This command will inspect all of the models in your application and create `FieldHistory` objects for the models that have a `FieldHistoryTracker`. Run this the first time you install django-field-history.

```
python manage.py createinitialfieldhistory
```

renamefieldhistory

Use this command after changing a model field name of a field you track with `FieldHistoryTracker`:

```
python manage.py renamefieldhistory --model=app_label.model_name --from_field=old_
↪field_name --to_field=new_field_name
```

For instance, if you have this model:

```
class Person(models.Model):
    username = models.CharField(max_length=255)

    field_history = FieldHistoryTracker(['username'])
```

And you change the `username` field name to `handle`:

```
class Person(models.Model):
    handle = models.CharField(max_length=255)

    field_history = FieldHistoryTracker(['handle'])
```

You will need to also update the `field_name` value in all `FieldHistory` objects that point to this model:

```
python manage.py renamefieldhistory --model=myapp.Person --from_field=username --to_
↪field=handle
```

Storing Which User Changed the Field

There are two ways to store the user that changed your model field. The simplest way is to use **the logged in user** that made the request. To do this, add the `FieldHistoryMiddleware` class to your `MIDDLEWARE` setting (in Django 1.10+) or your `MIDDLEWARE_CLASSES` setting (in Django 1.7-1.9).

```
MIDDLEWARE = [
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'field_history.middleware.FieldHistoryMiddleware',
]
```

Alternatively, you can add a `_field_history_user` property to the model that has fields you are tracking. This property should return the user you would like stored on `FieldHistory` when your field is updated.

```
class Pizza(models.Model):
    name = models.CharField(max_length=255)
    updated_by = models.ForeignKey('auth.User')
    field_history = FieldHistoryTracker(['name'])

    @property
    def _field_history_user(self):
        return self.updated_by
```

Working with MySQL

If you're using MySQL, the default configuration will throw an exception when you run migrations. (By default, `FieldHistory.object_id` is implemented as a `TextField` for flexibility, but indexed columns in MySQL InnoDB tables may be a maximum of 767 bytes.) To fix this, you can set `FIELD_HISTORY_OBJECT_ID_TYPE` in settings.py to override the default field type with one that meets MySQL's constraints. `FIELD_HISTORY_OBJECT_ID_TYPE` may be set to either:

1. the Django model field class you wish to use, or
2. a tuple `(field_class, kwargs)`, where `field_class` is a Django model field class and `kwargs` is a dict of arguments to pass to the field class constructor.

To approximate the default behavior for Postgres when using MySQL, configure `object_id` to use a `CharField` by adding the following to settings.py:

```
from django.db import models
FIELD_HISTORY_OBJECT_ID_TYPE = (models.CharField, {'max_length': 100})
```

`FIELD_HISTORY_OBJECT_ID_TYPE` also allows you to use a field type that's more efficient for your use case, even if you're using Postgres (or a similarly unconstrained database). For example, if you always let Django auto-create an `id` field (implemented internally as an `AutoField`), setting `FIELD_HISTORY_OBJECT_ID_TYPE` to `IntegerField` will result in efficiency gains (both in time and space). This would look like:

```
from django.db import models
FIELD_HISTORY_OBJECT_ID_TYPE = models.IntegerField
```

Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements-test.txt
(myenv) $ python runtests.py
```

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-field-history
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-field-history
$ pip install django-field-history
```

CHAPTER 3

Usage

To use django-field-history in a project:

```
import field_history
```

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/grantmcconnaughey/django-field-history/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.

Fix Bugs

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

Implement Features

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

Write Documentation

django-field-history could always use more documentation, whether as part of the official django-field-history docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/grantmcconnaughey/django-field-history/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 :)

Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django-field-history.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-field-history
$ cd django-field-history/
$ 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:

```
$ flake8 field_history
$ python runtests.py tests
```

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.

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 2.7 and 3.2+, and for Django 1.8+. Check https://travis-ci.org/grantmcconnaughey/django-field-history/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

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

CHAPTER 5

Credits

Development Lead

- Grant McConnaughey <grantmcconnaughey@gmail.com>

Contributors

- Boris Shifrin <https://github.com/ramusus>

Background

The `FieldHistoryTracker` class in this project is based off of the `FieldTracker` class from django-model-utils. The following authors contributed to `FieldTracker`:

- Trey Hunner
- Matthew Schinckel
- Mikhail Silonov
- Carl Meyer
- @bboogaard

CHAPTER 6

History

0.6.0 (December 22, 2016)

- Added Django 1.10 compatibility.
- Added MySQL compatibility.
- Fixed issue that would duplicate tracked fields.

0.5.0 (April 16, 2016)

- Added the ability to track field history of parent models.
- Added Django 1.7 compatibility.

0.4.0 (February 24, 2016)

- Added a way to automatically store the logged in user on `FieldHistory.user`.

0.3.0 (February 20, 2016)

- `FieldHistory` objects are now created using `bulk_create`, which means only one query will be executed, even when changing multiple fields at the same time.
- Added a way to store which user updated a field.
- Added `get_latest_by` to `FieldHistory` Meta options so `.latest()` and `.earliest()` can be used.
- Added `createinitialfieldhistory` management command.

- Added `renamefieldhistory` management command.

0.2.0 (February 17, 2016)

- First release on PyPI.