eventlog Documentation

Release 0.11.0

Eldarion

May 04, 2015

Contents

1 Development **3**
1.1 Contents 3

`eventlog` is a simple app that provides an easy and clean interface for logging diagnostic as well as business intelligence data about activity that occurs in your site.

Out of the box using this does write to the database. For small sites, it should be good enough to use inline but you might at some point want to consider wrapping calls to the `log()` method and queue them in a job manager like `celery` or `pyres` so that the calls become asyncronous.

CHAPTER 1

Development

The source repository can be found at https://github.com/eldarion/eventlog/

1.1 Contents

1.1.1 ChangeLog

0.11.0

- added the ability to link content objects you are logging about

0.10.0

0.9.0

0.8.0

- removed non-working templatetag
- update setup to work with Python 3.3+

0.7.0

- remove pusher integration
- support for custom user model

0.6.7

- added the *event_logged* signal
- corrected typo in usage documentation

0.6.6

- attempts at fixing admin performance

0.6.5

- attempts at fixing admin performance

0.6.4

- attempts at fixing admin performance with an index on action

0.6.3

- attempts at fixing admin performance with an index on timestamp

0.6.2

- update setup.py to use install_requires instead of setup_requires

0.6.1

- made the extra argument optional

0.6.0

- improve the admin

0.5.5

- use *django.utils.timezone.now* instead of *datetime.datetime.now* for timestamp

0.5.4

- when a user is deleted set FK to null instead of losing data

0.5.3

- bumped version on django-jsonfield

0.5.2

- added docs

0.5.1

- initial release

1.1.2 Installation

- To install

  ```
  pip install eventlog
  ```

- Add 'eventlog' to your INSTALLED_APPS setting:

  ```
  INSTALLED_APPS = (
      # other apps
      "eventlog",
  )
  ```

1.1.3 Signals

There is a signal that you are setup a receiver for to enable you to trigger other actions when an event has been logged:

event_logged provides an *event* object as an argument that is the event that was just logged.

1.1.4 Usage

Using eventlog is pretty simple. Throughout your site, you just call a single function, log() to record whatever information you want to log. If you are wanting to log things from third party apps, your best bet is to use signals. Hopefully the app in question provides some useful signals, but if not, perhaps some of the built in model signals will be enough (e.g. pre_save, post_delete, etc.)

Example:

```
from eventlog.models import log

def some_view(request):
    # stuff is done in body of view
    # then at the end before returning the response:
    log(
        user=request.user,
        action="CREATED_FOO_WIDGET",
        extra={
            "pk": foo.pk,
            "title": foo.title
        }
    )
    return HttpResponse()
```

The action parameter can be any string you choose. By convention, we always use all caps. Take note, however, whatever you choose, will be the label that appears in the admin's list filter, so give it some thought on naming conventions in your site so that the admin interface makes sense when you have 50,000 log records you want to filter down and analyze.

The extra parameter can be anything that will serialize to JSON. Results become easier to manager if you keep it at a single level. Also, keep in mind that this is displayed in the admin's list view so if you put too much it can take up a lot of space. A good rule of thumb here is put enough identifying data to get a sense for what is going on and a key or keys that enable you to dig deeper if you want or need to.

Mixin

You can also easily make your class based views auto-logged by using the *eventlog.mixins.EventLogMixin*. The only requirement is defining an *action_kind* property on the view. But you can also override a number of properties to customize what is logged.