django-audit-log Documentation

Release 0.2.2

Vasil Vangelovski (Atomidata)

December 27, 2013

Contents

Contents:

CHAPTER 1

Installation

Install from PyPI with `easy_install` or `pip`:

```
pip install django-audit-log
```

to hack on the code you can symlink the package in your site-packages from the source tree:

```
python setup.py develop
```

The package audit_log doesn't need to be in your `INSTALLED_APPS`. The only thing you need to modify in your `settings.py` is add `audit_log.middleware.UserLoggingMiddleware` to the `MIDDLEWARE_CLASSES` tupple:

```
MIDDLEWARE_CLASSES = (
     'django.middleware.common.CommonMiddleware',
     'django.contrib.sessions.middleware.SessionMiddleware',
     'django.contrib.auth.middleware.AuthenticationMiddleware',
     'audit_log.middleware.UserLoggingMiddleware',
)
```

CHAPTER 2

Usage

2.1 Tracking changes on a model

In order to enable change tracking on a model, the models needs to have a property of type `audit_log.models.managers.AuditLog` attached:

```
from django.db import models
from audit_log.models.fields import LastUserField
from audit_log.models.managers import AuditLog


class ProductCategory(models.Model):
    name = models.CharField(max_length=150, primary_key = True)
    description = models.TextField()

    audit_log = AuditLog()

class Product(models.Model):
    name = models.CharField(max_length = 150)
    description = models.TextField()
    price = models.DecimalField(max_digits = 10, decimal_places = 2)
    category = models.ForeignKey(ProductCategory)

    audit_log = AuditLog()
```

Each time you add an instance of AuditLog to any of your models you need to run `python manage.py syncdb` so that the database table that keeps the actual audit log for the given model gets created.

2.2 Querying the audit log

An instance of `audit_log.models.managers.AuditLog` will behave much like a standard manager in your model. Assuming the above model configuration you can go ahead and create/edit/delete instances of Product, to query all the changes that were made to the products table you would need to retrieve all the entries for the audit log for that particular model class:

```
In [2]: Product.audit_log.all()
Out[2]: [<ProductAuditLogEntry: Product: My widget changed at 2011-02-25 06:04:29.292363>,
```

```
        <ProductAuditLogEntry: Product: My widget changed at 2011-02-25 06:04:24.898991>,
        <ProductAuditLogEntry: Product: My Gadget super changed at 2011-02-25 06:04:15.448934>,
        <ProductAuditLogEntry: Product: My Gadget changed at 2011-02-25 06:04:06.566589>,
        <ProductAuditLogEntry: Product: My Gadget created at 2011-02-25 06:03:57.751222>,
        <ProductAuditLogEntry: Product: My widget created at 2011-02-25 06:03:42.027220>]
```

Accordingly you can get the changes made to a particular model instance like so:

```
In [4]: Product.objects.all()[0].audit_log.all()
Out[4]: [<ProductAuditLogEntry: Product: My widget changed at 2011-02-25 06:04:29.292363>,
        <ProductAuditLogEntry: Product: My widget changed at 2011-02-25 06:04:24.898991>,
        <ProductAuditLogEntry: Product: My widget created at 2011-02-25 06:03:42.027220>]
```

Instances of `AuditLog` behave like django model managers and can be queried in the same fashion.

The querysets yielded by `AuditLog` managers are querysets for models of type `[X]AuditLogEntry`, where X is the tracked model class. An instance of `XAuditLogEntry` represents a log entry for a particular model instance and will have the following fields that are of relevance:

- `action_id` - Primary key for the log entry.
- `action_date` - The point in time when the logged action was performed.
- `action_user` - The user that performed the logged action.
- `action_type` - The type of the action (Created/Changed/Deleted)
- Any field of the original `X` model that is tracked by the audit log.

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*