django-audit-log Documentation

Release 0.8.0

Vasil Vangelovski (Atomidata)

Jun 24, 2017

Contents

Adds support for tracking who changed what models through your Django application.

- Tracking creators and modifiers of your model instances.
- Tracking full model history.

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.contrib.sessions.middleware.SessionMiddleware',
     'django.middleware.common.CommonMiddleware',
     'django.middleware.csrf.CsrfViewMiddleware',
     'django.contrib.auth.middleware.AuthenticationMiddleware',
     'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
     'django.contrib.messages.middleware.MessageMiddleware',
     'django.middleware.clickjacking.XFrameOptionsMiddleware',
     'audit_log.middleware.JWTAuthMiddleware',
     'audit_log.middleware.UserLoggingMiddleware',
)
```

For users of `django-rest-framework-jwt` you should also include a special middleware that fixes a compatibility problem with that library:

```
MIDDLEWARE_CLASSES = (
  'django.contrib.sessions.middleware.SessionMiddleware',
  'django.middleware.common.CommonMiddleware',
  'django.middleware.csrf.CsrfViewMiddleware',
  'django.contrib.auth.middleware.AuthenticationMiddleware',
  'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
  'django.contrib.messages.middleware.MessageMiddleware',
  'django.middleware.clickjacking.XFrameOptionsMiddleware',
```

```
  'audit_log.middleware.JWTAuthMiddleware',
  'audit_log.middleware.UserLoggingMiddleware',
)
```

Note that in that case `rest_framework_jwt.authentication.JSONWebTokenAuthentication` should be at the top of `DEFAULT_AUTHENTICATION_CLASSES`.

CHAPTER 2

Tracking Users that Created/Modified a Model

`AuthStampedModel` is an abstract model base class in the vein of `TimeStampedModel` from `django-extensions`. It has 4 fields used for tracking the user and the session key with which a model instance was created/modified:

```
from audit_log.models import AuthStampedModel

class WarehouseEntry(AuthStampedModel):
    product = models.ForeignKey(Product)
    quantity = models.DecimalField(max_digits = 10, decimal_places = 2)
```

This will add 4 fields to the `WarehouseEntry` model:

- `created_by` - A foreign key to the user that created the model instance.
- `created_with_session_key` - Stores the session key with which the model instance was first created.
- `modified_by` - A foreign key to the user that last saved a model instance.
- `modified_with_session_key` - Stores the session key with which the model instance was last saved.

The related names for the `created_by` and `modified_by` fields are `created_%(class)s_set` and `modified_%(class)s_set` respectively:

```
In [3]: admin = User.objects.get(username = 'admin')
In [4]: admin.created_warehouseentry_set.all()
Out[4]: [<WarehouseEntry: WarehouseEntry object>, <WarehouseEntry: WarehouseEntry␣
↪object>]
In [5]: vasil = User.objects.get(username = 'vasil')
In [6]: vasil.modified_warehouseentry_set.all()
Out[6]: [<WarehouseEntry: WarehouseEntry object>]
```

This was done to keep in line with Django's naming for the `related_name`. If you want to change that or other things you can create your own abstract base class with the proviced fields.

This is very useful when used in conjuction with `TimeStampedModel` from `django-extensions`:

```
from django_extensions.db.models import TimeStampedModel
from audit_log.models import AuthStampedModel


class Invoice(TimeStampedModel, AuthStampedModel):
    group = models.ForeignKey(InvoiceGroup, verbose_name = _("group"))
    client = models.ForeignKey(ClientContact, verbose_name = _("client"))
    currency = models.ForeignKey(Currency, verbose_name = _("currency"))
    invoice_number = models.CharField(_("invoice number"), blank = False, max_length
↪= 15)
    date_issued = models.DateField(_("date issued"))
    date_due = models.DateField(verbose_name = _("date due"))
    comment = models.TextField(_("comment"), blank = True)
    is_paid = models.BooleanField(_("is paid"), default = False)
    date_paid = models.DateField(_("date paid"), blank = True, null = True)
```

Tracking Who Created a Model

You can track user information when model instances get created with the `CreatingUserField` and `CreatingSessionKeyField`. For example:

```
from audit_log.models.fields import CreatingUserField, CreatingSessionKeyField

class ProductCategory(models.Model):
    created_by = CreatingUserField(related_name = "created_categories")
    created_with_session_key = CreatingSessionKeyField()
    name = models.CharField(max_length=15)
```

This is useful for tracking owners of model objects within your app.

Tracking Who Made the Last Changes to a Model

`LastUserField` and `LastSessionKeyField` will store the user and session key with which a model instance was last saved:

```
from django.db import models
from audit_log.models.fields import LastUserField, LastSessionKeyField

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)

    def __unicode__(self):
        return self.name

class ProductRating(models.Model):
    user = LastUserField()
    session = LastSessionKeyField()
    product = models.ForeignKey(Product)
    rating = models.PositiveIntegerField()
```

Anytime someone makes changes to the `ProductRating` model through the web interface the reference to the user that made the change will be stored in the user field and the session key will be stored in the session field.

CHAPTER 3

Tracking full model history

In order to enable historic tracking on a model, the model 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.

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.

M2M Relations

Tracking changes on M2M Relations doesn't work for now. If you really need to track changes on M2M relations with this package, explicitly define the table with another model instead of declaring the M2M relation.

Abstract Base Models

For now just attaching the `AuditLog` manager to an abstract base model won't make it automagically attach itself on the child models. Just attach it to every child separately.

Disabling/Enabling Tracking on a Model Instance

There may be times when you want a certain `save()` or `delete()` on a model instance to be ignored by the audit log. To disable tracking on a model instance you simply call:

```
modelinstance.audit_log.disable_tracking()
```

To re-enable it do:

```
modelinstance.audit_log.enable_tracking()
```

Note that this only works on instances, trying to do that on a model class will raise an exception.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search