django-adminjournal

Nov 16, 2018

Contents

This library added extended capabilities to log access to Django ModelAdmins.

CHAPTER 1

Features

- Log additions, changes, deletions of models via the Django admin
- Log read access to change lists and model instances (unsaved change views)
- Log calls to actions in changelists of ModelAdmins

CHAPTER 2

Requirements

django-adminjournal supports Python 3 only and requires at least Django 1.11. The package uses Django's JSONField. Therefore, PostgreSQL database backend is required.

CHAPTER 3

Prepare for development

A Python 3.6 interpreter is required in addition to pipenv.

```
$ pipenv install --python 3.6 --dev
$ pipenv shell
$ pip install -e .
```

Now you're ready to run the tests:

```
$ pipenv run py.test
```

CHAPTER 4

Resources

- Documentation
- Bug Tracker
- Code

Contents:

4.1 Installation

- Install with pip:

```
pip install django-adminjournal
```

- Your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    # ...
    'adminjournal',
)
```

4.1.1 Configuration options

- `ADMINJOURNAL_PERSISTENCE_BACKEND` defines the backend that is used to store/persist the journal entries. Default is a database backend.
- `ADMINJOURNAL_MODEL_WHITELIST` defines the models to automatically activate the ModelAdmin mixin. The settings should be a list of Django models (e.g. `auth.User`) or the string `'__all__'` to activate the admin journal for all models.
- `ADMINJOURNAL_ENTRY_EXPIRY_DAYS` defines the number of days after which the journal entries are deleted when calling the management command `clearadminjournal`. The default is `365` days.

4.2 Usage

After adding `adminjournal` to `INSTALLED_APPS`, the journal is activated for all model admins added to Django's default AdminSite (`django.contrib.admin.site`).

4.2.1 Cleanup

Journal entries can be deleted automatically after a given amount of time (see configuration option `ADMINJOURNAL_ENTRY_EXPIRY_DAYS`). To do so, run the `clearadminjournal` regulary using a cron daemon or some other trigger tool.

If you use uwsgi, you might even us the built in cron helper:

```
[uwsgi]
# ...

# Django session cleanup
cron = 30 4 -1 -1 -1 django-admin clearsessions

# Adminjournal cleanup
cron = 15 4 -1 -1 -1 django-admin clearadminjournal
```

This would run the cleanup command every day at 4:15 am.

4.3 Changelog

4.3.1 0.1.0 (2018-11-16)

- Add management command to clean up old journal entries

4.3.2 0.0.1 (2018-11-12)

- Initial release of *django-adminjournal*

Api documentation:

4.4 API Reference

4.4.1 adminjournal package

Subpackages

adminjournal.persistence_backends package

Submodules

adminjournal.persistence_backends.base module

class adminjournal.persistence_backends.base.**BaseBackend**
Bases: object

Base backend to persist journal entries.

Every backend must provide at least a *persist* method.

persist(*entry*)
The *persist* method is able to persist instances of *adminjournal.entry.Entry* classes. The method will return *True* or *False* to signal success.

adminjournal.persistence_backends.db module

class adminjournal.persistence_backends.db.**Backend**
Bases: *adminjournal.persistence_backends.base.BaseBackend*

Database-backed persistence layer for journal entries. Uses adminjournal.Entry model to store entries to database.

persist(*entry*)
The *persist* method is able to persist instances of *adminjournal.entry.Entry* classes. The method will return *True* or *False* to signal success.

adminjournal.persistence_backends.log module

class adminjournal.persistence_backends.log.**Backend**
Bases: *adminjournal.persistence_backends.base.BaseBackend*

Simple backend that uses Python-logging to "persist" a given entry.

The log backend has two settings:

- ADMINJOURNAL_BACKEND_LOG_LOGGER: Name of the logger to use
- ADMINJOURNAL_BACKEND_LOG_LEVEL: Valid logging level name to use

persist(*entry*)
The *persist* method is able to persist instances of *adminjournal.entry.Entry* classes. The method will return *True* or *False* to signal success.

Submodules

adminjournal.admin module

class adminjournal.admin.**EntryAdmin**(*model*, *admin_site*)
Bases: django.contrib.admin.options.ModelAdmin

search_fields = ('user_repr',)

list_display = ('__str__', 'action', 'content_type_repr', 'user_repr', 'object_id', 'l

list_filter = ('action', 'content_type__app_label', 'user_repr')

date_hierarchy = 'timestamp'

readonly_fields = ('timestamp', 'action', 'user', 'user_repr', 'content_type', 'conten

has_add_permission(*request*, *obj=None*)
has_add_permission is overwritten to ensure no entries can be added.

has_delete_permission(*request*, *obj=None*)
has_delete_permission is overwritten to ensure nobody can remove entries.

lazy_description(*obj*)
Helper to return the human readable entry description. If no description is available, the payload will be returned.

object_repr(*obj*)
Helper to get the str-representation of the logged object.

media

adminjournal.apps module

class adminjournal.apps.**AdminjournalConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

name = 'adminjournal'

ready()
When loading the adminjournal app, we patch the Django admin site to ensure every model admin is hooked to the admin journal mixin if the setting ADMINJOURNAL_PATCH_ADMINSITE is set to True (default).

adminjournal.entry module

class adminjournal.entry.**Entry**(*action*, *user*, *model_class=None*, *model=None*, *description=None*, *timestamp=None*, *payload=None*)
Bases: object

This class represents a journal entry and provides methods to get information about the action which was tracked.

ACTION_VIEW = 'view'

ACTION_ADD = 'add'

ACTION_CHANGE = 'change'

ACTION_DELETE = 'delete'

__init__(*action*, *user*, *model_class=None*, *model=None*, *description=None*, *timestamp=None*, *payload=None*)
Create a new entry instance.

The constructor handles the provided data and does some basis validation.

The parameters *action* and *user* are always required. It is possible to override the timestamp which used for that event.

In addition, *model_class* and/or *model* needs to be provided. If both are given, the constructor will ensure that the *model_class* fits the *provided model*.

It is allowed to provide one of the following as *model_class*:

- Python class of a Django model
- ContentType model instance
- None (if none is provided, the model_class will be derived from the given model).

You don't have to provide a *model* if you already have the *model_class* on hand.

The parameter *description* is useful to provide a human-readable representation of what happened.

timestamp = None
Point in time when the event happend.

payload = None
Dict-like object holding any other information related to the event.

user_repr
Returns a human readable version of the user object.

content_type_repr
Returns a human readable version of the content type object.

object_id

persist()
Triggers the persisting of the instance.

adminjournal.mixins module

class adminjournal.mixins.**JournaledModelAdminMixin**
Bases: object

Mixin for ModelAdmin classes to issue journal entries on various actions via the model admin.

Tracked actions:

- View changelist (w/ and w/o filters)
- View object change view
- Change object
- Add object
- Delete object
- Changelist actions (selected action and selected objects)

log_to_adminjournal(*action*, *user*, *message*, *model=None*, *payload=None*)
The log_to_adminjournal method requires at least the action type and the issuing user together with a human readable message or a change_message-style list from Django's LogEntry.

The method might use change_message-style lists to generate a human readable version of the data.

If a change_message-style input is provided, the payload is ignored.

If a str message is provided and the payload is a dictionary, the data is passed to the persistence layer.

log_addition(*request*, *model*, *message*)
In addition to the Django LogEntry, add another entry to the adminjournal.

log_change(*request*, *model*, *message*)
In addition to the Django LogEntry, add another entry to the adminjournal.

log_deletion(*request*, *model*, *object_repr*)
In addition to the Django LogEntry, add another entry to the adminjournal.

render_change_form(*request*, **args*, ***kwargs*)
If a object change view is requested (GET request on change view), a ACTION_VIEW entry is generated to track read access to single objects.

response_action(*request*, *queryset*)
Actions on change lists are tracked too. To achieve this, this method parses the POST request and checks for various action fields to

- Learn what action was triggered
- What obejects are included in the action call
- Are all objects involved or just some selected ones

We don't know what the action does, therefore all actions are tracked as ACTION_VIEW.

changelist_view(*request*, **args*, ***kwargs*)
GET requests on the changelist are tracked as ACTION_VIEW entries. The changelist view might also return redirects. This case is handled by ensuring that a proper TemplateResponse is available and the "cl" context (which is the ChangeList instance) is present.

Parameters passed to the change list are tracked too. This allows to reconstruct the subset of objects a user viewed.

adminjournal.models module

class adminjournal.models.**Entry**(*id*, *timestamp*, *action*, *user*, *user_repr*, *content_type*, *content_type_repr*, *object_id*, *description*, *payload*)
Bases: django.db.models.base.Model

timestamp
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

action
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

user
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

user_repr
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

content_type
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

content_type_repr
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

object_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

description
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

payload
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception MultipleObjectsReturned
Bases: `django.core.exceptions.MultipleObjectsReturned`

content_type_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

user_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

adminjournal.monkeypatch module

adminjournal.monkeypatch.**patch_admin_site**(*site*)
This helper patches the default Django admin site to ensure the JournaledModelAdminMixin is added to the model admins.

After patching the admin site, this helper checks all already registered model admins to be adminjournal enabled.

adminjournal.persistence module

adminjournal.persistence.**persist**(*entry*, *backend=None*)
The *persist* function is the abstract entrypoint to persist journal entries. The method receives a *adminjournal.entry.Entry* instance and an optional *backend* parameter to override the default persistence backend.

The return value is either *True* of *False*, to signal if the entry was saved.

adminjournal.persistence.**get_persistence_backend**(*path=None*)
Load a persistence backend and return a instance. If a path is provided, the backend is imported from that path. By default, `adminjournal.persistence_backends.db.Backend` is used.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

Symbols

A

B

C

D

E

G

H

I

J

L

M

N

O

P

R

S

T

U