Ebury Audit Documentation

Release 0.4.0

José Antonio Perdiguero López

January 19, 2016

Contents

CHAPTER 1

Installation

1.1 Installation

To install *Audit* you need to follow the next steps:

1. Add MongoDB repository.
2. Install MongoDB.
3. Configure authentication for MongoDB.
4. Install Django Audit Tools package.

1.1.1 Install MongoDB

Add repository

Import the public key used by the package management system:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
```

Create a list file for MongoDB repository:

```
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sou
```

Reload local package database:

```
sudo apt-get update
```

Install MongoDB packages

Install the MongoDB packages:

```
sudo apt-get install mongodb-org=2.6.3 mongodb-org-server=2.6.3 mongodb-org-shell=2.6.3 mongodb-org-m
```

(OPTIONAL) Pin a specific version of MongoDB:

```
echo "mongodb-org hold" | sudo dpkg --set-selections
echo "mongodb-org-server hold" | sudo dpkg --set-selections
echo "mongodb-org-shell hold" | sudo dpkg --set-selections
echo "mongodb-org-mongos hold" | sudo dpkg --set-selections
echo "mongodb-org-tools hold" | sudo dpkg --set-selections
```

Start MongoDB:

```
sudo service mongod start
```

Configure authentication

Connect to MongoDB using client terminal:

```
mongo
```

Change to admin database:

```
use admin
```

Create admin user (replace <USER_ADMIN> and <PASSWORD_ADMIN>):

```
db.createUser({
    user: "<USER_ADMIN>",
    pwd: "<PASSWORD_ADMIN>",
    roles:
        [
            { role: "userAdminAnyDatabase", db: "admin" }
        ]
    }
)
```

Activate authentication login uncommenting the next line in /etc/mongod.conf:

```
# auth = True
```

Restart MongoDB:

```
sudo service mongod restart
```

Connect to MongoDB as admin:

```
mongo admin -u <USER_ADMIN> -p <PASSWORD_ADMIN>
```

Change to audit database (replace <DATABASE>):

```
use <DATABASE>
```

Create audit user (replace <USER> and <PASSWORD>):

```
db.createUser({
    user: "<USER>",
    pwd: "<PASSWORD>",
    roles:
        [
            { role: "readWrite", db: "<DATABASE>" }
        ]
    }
)
```

(OPTIONAL) Add environment variables appending next lines to virtualenv's activate file (replace <USER> and <PASSWORD>):

```
export DB_AUDIT_USER=<USER>
export DB_AUDIT_PASSWORD=<PASSWORD>
```

1.1.2 Install Audit

Use pip to install *Audit* from local file:

```
pip install django-audit-tools.tar.gz
```

Or install from pip repository:

```
pip install django-audit-tools
```

1.2 Configuration

To configure Audit Tools you need to follow the next steps:

1. Add *audit_tools* to your **INSTALLED_APPS** settings like this:

```
INSTALLED_APPS = (
    ...
    'audit_tools',
)
```

2. Add *audit.middleware.AuditMiddleware* to your **MIDDLEWARE_CLASSES** settings like this:

```
MIDDLEWARE_CLASSES = (
    ...
    'audit_tools.audit.middleware.AuditMiddleware',
)
```

3. Configure blacklisted URLs in **AUDIT_BLACKLIST** settings.
4. Register models that will be logged in **AUDIT_LOGGED_MODELS** settings.
5. Execute the next django command:

```
python manage.py prepare_audit
```

1.2.1 Settings

AUDIT_ACTIVATE

Activate or deactivate audit.

Default:

```
AUDIT_ACTIVATE = True
```

AUDIT_DB_ALIAS

Audit database connection alias.

Default:

```
AUDIT_DB_ALIAS = 'audit'
```

AUDIT_DB_CONNECTION

Audit database connection parameters.

Default:

```
AUDIT_DB_CONNECTION = {
    'HOST': 'localhost',
    'PORT': 27017,
    'NAME': 'audit',
    'USER': '',
    'PASSWORD': '',
}
```

AUDIT_RUN_ASYNC

Use Celery to run in async mode.

Important: Celery concurrency level must be configure to 1 (–concurrency=1 parameter in celeryd start)

Default:

```
AUDIT_RUN_ASYNC = False
```

AUDIT_CELERY_QUEUE

Celery queue name.

Default:

```
AUDIT_CELERY_QUEUE = 'audit'
```

AUDIT_LOGGED_MODELS

List of models that will be logged for audit. Each entry consists in a string that represents a model using *"<module>.<model>"* format.

Example:

```
AUDIT_LOGGED_MODELS = (
    'audit_tools.audit.models.Access',
)
```

Default:

```
AUDIT_LOGGED_MODELS = ()
```

AUDIT_BLACKLIST

Blacklisted URLs. Each application may have a tuple of regex patterns. If an URL matches a pattern will not be logged.

Example:

```
AUDIT_BLACKLIST = {
    'api': (
        r'^/api/.*',
        r'^/API/.*',
    )
}
```

Default:

```
AUDIT_BLACKLIST = {}
```

AUDIT_ACCESS_INDEXES

Custom indexes for the accesses. There is the possibility to add new custom indexes to the Audit database.

Example:

```
AUDIT_ACCESS_INDEXES = [
    'custom.pools.names',
    'custom.pools.num_polls',
    ('custom.pools.names', 'custom.pools.num_polls'),
]
```

AUDIT_PROCESS_INDEXES

Custom indexes for the processes. There is the possibility to add new custom indexes to the Audit database.

AUDIT_MODEL_ACTION_INDEXES

Custom indexes for the model actions. There is the possibility to add new custom indexes to the Audit database.

AUDIT_CUSTOM_PROVIDER

Custom data provider. Each application may add custom data to Access entries using own functions.

Default:

```
AUDIT_CUSTOM_PROVIDER = {
    'audit_tools': 'audit_tools.audit.middleware.custom_provider',
}
```

AUDIT_TRANSLATE_URLS

Translate Audit URLs:

Default:

```
AUDIT_TRANSLATE_URLS = False
```

CHAPTER 2

Audit

2.1 Signals

audit_tools.audit.signals.**register**(*model*)
Register a model to the audit code.

Parameters **model** (*object*) – Model to register.

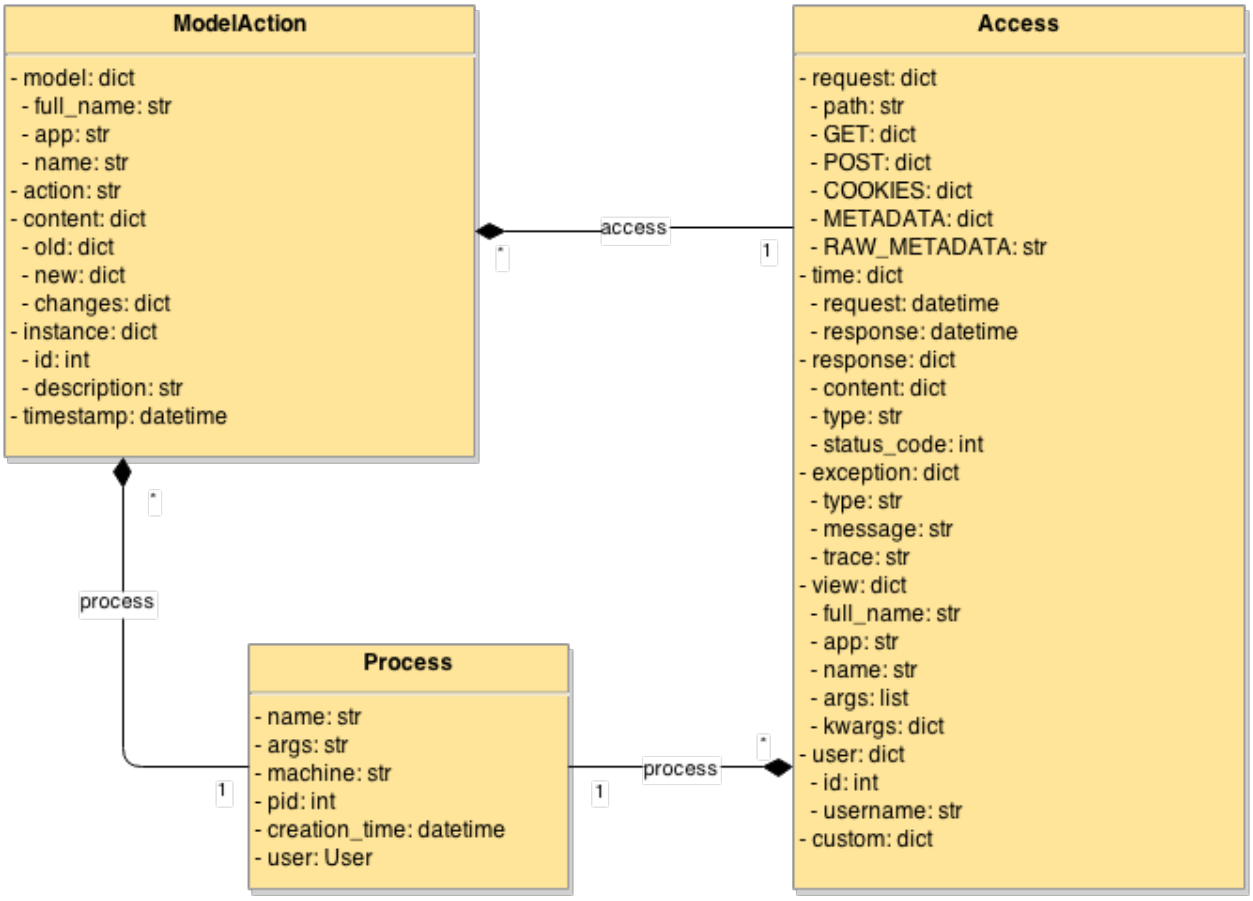
audit_tools.audit.signals.**register_models**()
Register all models listed in settings.LOGGED_MODELS.

audit_tools.audit.signals.**unregister**(*model*)
Unregister a model to the audit code.

Parameters **model** (*object*) – Model to unregister.

audit_tools.audit.signals.**unregister_models**()
Unregister all models listed in settings.LOGGED_MODELS.

2.2 Models



Ebury Audit models

class `audit_tools.audit.models.`**`Access`**(**args*, ***values*)

Information gathered from a request and response objects. Contains the following structure:

Variables

- **`interlink_id`** – Interlink id.
- **`request`** – Request.
- **`response`** – Response content, type and status code.
- **`exception`** – Exception raised.
- **`time`** – Request and response time.
- **`view`** – View full_name, app, name, args and kwargs.
- **`user`** – User id and name.
- **`custom`** – Custom providers field.
- **`process`** – Reference to Process object.

`get_user`()

Obtains User object that corresponds to this Access instance.

Returns User object.

Return type django.contrib.auth.models.User

get_view()
Obtains view object that corresponds to this Access instance.

Returns View.

Return type object

is_exception()
Check if this Access contains a exception.

Returns True if contains exception.

Return type bool

is_response()
Check if this Access contains a response.

Returns True if contains response.

Return type bool

items()
List items in form (key, value).

Returns list(tuple())

url
Property that gives access to request__path field.

Type str

verbose_str()
Verbose string representation.

Returns Verbose string representation.

Return type str

class audit_tools.audit.models.**Process**(**args*, ***values*)
Represents a process that launch a django management command. Contains the following structure:

Variables

- **name** – Process name.
- **args** – Process args.
- **machine** – Machine that launched this process.
- **user** – User that launched this process.
- **pid** – Process pid.
- **creation_time** – Time when process was launched.

items()
List items in form (key, value).

Returns list(tuple())

verbose_str()
Verbose string representation.

Returns Verbose string representation.

Return type str

class audit_tools.audit.models.**ModelAction**(**args*, ***values*)
Information from create, update or delete operations over a model. Contains the following structure:

Variables

- **model** – Model full_name, app and name.
- **action** – Create, update or delete action.
- **content** – Old and new values and changes.
- **instance** – Instance object id and description.
- **timestamp** – Time when action occurs.
- **access** – Reference to Access object.
- **process** – Reference to Process object.

changes
Property that gives access to content__changes field.

Type dict

get_instance()
Obtains instance object that corresponds to this ModelAction instance.

Returns Instance object.

Return type object

get_model()
Obtains Model class that corresponds to this ModelAction instance.

Returns Model class.

Return type object

items()
List items in form (key, value).

Returns list(tuple())

verbose_str()
Verbose string representation.

Returns Verbose string representation.

Return type str

2.3 Middleware

class audit_tools.audit.middleware.**AuditMiddleware**(**args*, ***kwargs*)
Middleware for audit logging

process_view(*self*, *request*, *view_func*, *view_args*, *view_kwargs*)
Preprocess request.

Parameters

- **request** (*django.http.HttpRequest*) – Http request.
- **view_func** (*callable*) – View.
- **view_args** (*list*) – View arguments.

- **view_kwargs** (*dict*) – View keyword arguments.

Returns None

process_response(*self*, *request*, *response*)
Postprocess response.

Parameters

- **request** (*django.http.HttpRequest*) – Http request.
- **response** (*django.http.HttpResponse*) – Response.

Returns None

process_exception(*self*, *request*, *exception*)
Postprocess exception.

Parameters

- **request** (*django.http.HttpRequest*) – Http request.
- **exception** (*Exception*) – Response.

Returns None

2.4 Managers

2.4.1 Access

The following methods are added to the default Access manager:

class audit_tools.audit.managers.**AccessQuerySet**(*document*, *collection*)
Custom manager for Access.

filter_by_exception(**args*, ***kwargs*)
Filter object by exception.

Parameters **kwargs** – exc kwarg required.

Returns QuerySet

filter_by_url(**args*, ***kwargs*)
Filtered queryset by url. Url accept all modifiers, including __regex.

Parameters **kwargs** – url kwarg required.

Returns QuerySet

filter_by_view(**args*, ***kwargs*)
Filtered queryset by view object or function.

Parameters **kwargs** – fview kwarg required.

Returns QuerySet

get_by_exception(**args*, ***kwargs*)
Get object by exception.

Parameters **kwargs** – exc kwarg required.

Returns Access

get_by_url (**args*, ***kwargs*)
Get object by url. Url accept all modifiers, including __regex.

Parameters **kwargs** – url kwarg required.

Returns Access

get_by_view (**args*, ***kwargs*)
Get object by view.

Parameters **kwargs** – fview kwarg required.

Returns Access

Examples:

```
# URL /
Access.objects.filter_by_url(url='/')
# Filter URLs using regular expression
Access.objects.filter_by_url(url=r'^/polls/\w*$')
# Accesses to /polls/ done by user with id 23
Access.objects.filter_by_url(url='/polls/', user__id=23)

# Accesses to poll's index
from polls.views import index
Access.objects.filter_by_view(fview=index)

# Accesses that raises an AttributeError exception
Access.objects.filter_by_exception(exc=AttributeError)
```

2.4.2 ModelAction

The following methods are added to the default ModelAction manager:

class audit_tools.audit.managers.**ModelActionQuerySet** (*document*, *collection*)
Custom manager for ModelAction.

filter_by_instance (**args*, ***kwargs*)
Filtered queryset by object instance.

Parameters **kwargs** – obj kwarg required.

Returns QuerySet

filter_by_model (**args*, ***kwargs*)
Filtered queryset by model.

Parameters **kwargs** – klass kwarg required.

Returns QuerySet

filter_by_model_list (**args*, ***kwargs*)
Filtered queryset by model list.

Parameters **kwargs** – klass kwarg required.

Returns QuerySet

get_by_instance (**args*, ***kwargs*)
Get object by instance.

Parameters **kwargs** – obj kwarg required.

Returns ModelAction

get_by_model(**args*, ***kwargs*)
Get object by model.

Parameters **kwargs** – klass kwarg required.

Returns ModelAction

get_by_model_list(**args*, ***kwargs*)
Get object by model list.

Parameters **kwargs** – klass kwarg required.

Returns ModelAction

Examples:

```
# Actions done over all polls
from polls.models import Poll
ModelAction.objects.filter_by_model(klass=Poll)

# Actions done over a single poll
poll = Poll.objects.get(id=1)
ModelAction.objects.filter_by_instance(obj=poll)

# Actions done over all polls and users
from django.contrib.auth.models import User
ModelAction.objects.filter_by_model_list(klass=[Poll, User])
```

2.5 Commands

Django commands for Audit application.

2.5.1 drop_audit

Drop audit database. If only initial date is given, data from that day to now. If the two dates are provided, data will be deleted in that range. Delete all data otherwise.

Syntax:

```
python manage.py drop_audit [init_date] [end_date]
```

Example:

```
python manage.py drop_audit
python manage.py drop_audit 01/01/2000
python manage.py drop_audit 01/01/2000 01/02/2000
```

2.5.2 prepare_audit

Prepare system for Audit application adding needed permissions, tables...

Syntax:

```
python manage.py prepare_audit
```

2.5.3 remove_audit

Remove Audit application from system.

Syntax:

```
python manage.py remove_audit
```

CHAPTER 3

Additional information

3.1 Custom providers

Custom provider is a mechanism that permits an application to add custom data to *Access* logs. A single provider can be specified for each app.

The next example define a provider that returns polls associated to current user:

```
def poll_provider(request):
    user = User.objects.get(pk=request.user.id)
    polls = Poll.objects.filter(user=user)
    polls_names = [p.name for p in polls]

    res = {
        'names': poll_names,
        'num_polls': len(poll_names),
    }

    return res
```

If this provider is defined inside `polls.utils` then must be set in `settings.AUDIT_CUSTOM_PROVIDER`:

```
AUDIT_CUSTOM_PROVIDER = {
    'polls': 'polls.utils.poll_provider',
}
```

This provider will result in an additional field inside *Access* named 'polls':

```
{
    ...
    custom = {
        polls = {
            names = [ "poll_1", "poll_2" ],
            num_polls = 2
        }
    }
}
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

a

A

C

F

G

I

M

P

R

U

V