Django Mqueue Documentation

Release 0.4.6

synw

Sep 27, 2017

Usage

1 Record events **3**
- 1.1 Event creation . . . 3
- 1.2 Fields autoguess . . . 4

2 Registered models **5**
- 2.1 Autoregister a model . . . 5
- 2.2 Manualy register a model . . . 5

3 Watchers **7**

4 Retrieve events **9**

5 Hooks **11**
- 5.1 Postgresql . . . 11
- 5.2 Influxdb . . . 12
- 5.3 Redis . . . 12
- 5.4 Centrifugo . . . 12
- 5.5 Custom hook . . . 12

6 Logs handler **13**

7 Graphical settings **15**
- 7.1 Event classes . . . 15
- 7.2 Event Extra html . . . 16
- 7.3 Event Icons . . . 16

8 Graphql Api **17**
- 8.1 Settings . . . 17
- 8.2 Event scope . . . 18
- 8.3 Queries . . . 18

To install: `pip install django-mqueue`

Add to INSTALLED_APPS:

```
"django_extensions",
"mqueue",
```

Run the migrations.

CHAPTER 1

Record events

You can plug mqueue into your app by creating an event whenever you need. It can be in the save method of a model, a form_valid method of a view or in a signal for example.

Event creation

```
from mqueue.models import MEvent
from myapp.models import MyModel

# simpliest event
MEvent.objects.create(name = 'Something happened!')

# full event
MEvent.objects.create(
   name = obj.title,
   model = MyModel,
   obj_pk =obj.pk,
   instance = obj,
   user = request.user,
   url ='/anything/'+obj.slug+'/',
   admin_url ='/admin/app/model/'+str(obj.pk)+'/',
   notes = 'Object X was saved!',
   event_class = 'Info',
   request = request,
   bucket = "bucket_name",
   data = {"foo": "bar"},
   scope = "users"
   )
```

The only required field is `name`

The `instance` parameter will not be recorded: it is only used for auto guessing some fields.

The `scope` parameter is used by the Api to query the database. Possible values are: `public`, `users`, `staff` and `superuser` (default). It controls who can view an event in the Api.

Note this method will return the event instance just created. There is an option for not to save it immediately:

```
from mqueue.models import MEvent
from myapp.models import MyModel

# initiate event
mevent = MEvent.objects.create(name='No commit', commit=False)
# do things ...
# update event
mevent.notes='Some stuff'
# and save it
mevent.save()
```

Fields autoguess

If you provided an instance or a content_type and a model mqueue will guess the following fields unless you provided arguments for:

- `user`: checks if you model has a `user` field or an `editor` field and populates from it
- `url`: checks for a `get_event_object_url()` method in your model, and then check for a `get_absolute_url()` method and populates from it. Write your own `get_event_object_url()` method in your model to manage which url will be associated to the object.
- `admin_url`: will be reversed from the instance

Example:

```
from mqueue.models import MEvent

MEvent.objects.create(name = 'Something happened!', instance=my_obj)
```

So that `user`, `url` and `admin_url` will be auto guessed

CHAPTER 2

Registered models

Models can be registered. They will be automatically monitored.

Autoregister a model

In `settings.py`:

```
MQUEUE_AUTOREGISTER = (
     ('django.contrib.auth.models.User', ["c", "d", "u"]),
     ('emailmodule.models.Email', ["c", "d"])
     )
```

The registered models will be monitored according to the chosen monitoring level:

`c`: create

`d`: delete

`u`: upgrade

Manualy register a model

In any installed app `apps.py` :

```
from django.apps import AppConfig

class MyappConfig(AppConfig):
    name = "myapp"
    verbose_name = "My app"

    def ready(self):
        from mqueue.tracking import mqueue_tracker
        from myapp.models import TheModel
```

```
        mqueue_tracker.register(TheModel)
```

By default this will records create and delete events. To change it set a parameter:

```
mqueue_tracker.register(TheModel, ["c", "u", "d"])
```

CHAPTER 3

Watchers

Some watchers connected to signals are available. Declare the ones you want to use in settings:

```
MQUEUE_WATCH = ["login", "logout", "login_failed"]
```

The events will be fired accordingly, same way than registering models

CHAPTER 4

Retrieve events

Events for a model:

```
from mqueue.models import MEvent
from myapp.models import MyModel

# get all events for the model
events_for_mymodel = MEvents.objects.get_for_model(MyModel)

# count all events for the model
events_for_mymodel = MEvents.objects.count_for_model(MyModel)
```

Events for an object:

```
from mqueue.models import MEvent

# get all events for the model
events_for_myobject = MEvents.objects.get_for_object(any_model_instance)
```

CHAPTER 5

Hooks

New in 0.7.1 (inspired by Logrus)

Optional hooks can be used to perform extra actions on events. Available hooks:

- **Postgresql**: record the events in a postgresql database (go)
- **Influxdb**: record the events in an influxdb database (go)
- **Redis**: record the events in Redis (python)
- **Centrifugo**: push events as messages in the Centrifugo websockets server (python)

Postgresql

In `settings.py`

```
MQUEUE_HOOKS = {
 "postgresql": {
     "path": "mqueue.hooks.postgresql",
     "addr": "localhost",
     "user": "user",
     "password": "pwd",
     "database": "mydomain_events",
     "table": "events"
 }
}
```

Create the database in postgresql and migrate it with a management command:

```
python3 manage.py mqueue_migrate_pg
```

Influxdb

```
"influxdb": {
     "path": "mqueue.hooks.influxdb",
     "addr": "http://localhost:8086",
     "user": "admin",
     "password": "admin",
     "database": "events"
 }
```

Create the database in Influxdb

Redis

```
"redis": {
     "path": "mqueue.hooks.redis",
     "host": "localhost",
     "port": 6379,
     "db": 0,
 }
```

Centrifugo

Install Django Instant to manage the websockets

```
"centrifugo": {
     "path": "mqueue.hooks.centrifugo",
     "channel": "$events"
 }
```

Note: for the Go based hooks you might need to make the binary (`mqueue/hooks/<hookname>/run`) executable with chmod

Custom hook

Create a file : `mymodule/mqueue_hook.py`

Declare your hook and config in settings:

```
MQUEUE_HOOKS = {
 "myhook": {
     "path": "mymodule.mqueue_hook",
     "myparam": "myvalue",
 }
}
```

Create a `save` function in your hook that takes and event object as parameter and the hook config. Example:

```
def save(event, conf):
    print(event, conf["myparam"])
```

CHAPTER 6

Logs handler

Mqueue has a log handler that to stores the django logs into the db as events.

To enable it add this to `settings.py`

```
from mqueue.logging import LOGGING
```

This will enable logging on ERROR level when `DEBUG` is `False`. To log on WARNING level (which also handles the the 404 errors and friends) do this:

```
from mqueue.logging import LOGGING_WARNING as LOGGING
```

To enable logging in dev mode, when `DEBUG` is `True` or `False` (useful to debug some ajax for example):

```
from mqueue.logging import DEV_LOGGING as LOGGING
```

CHAPTER 7

Graphical settings

Event classes

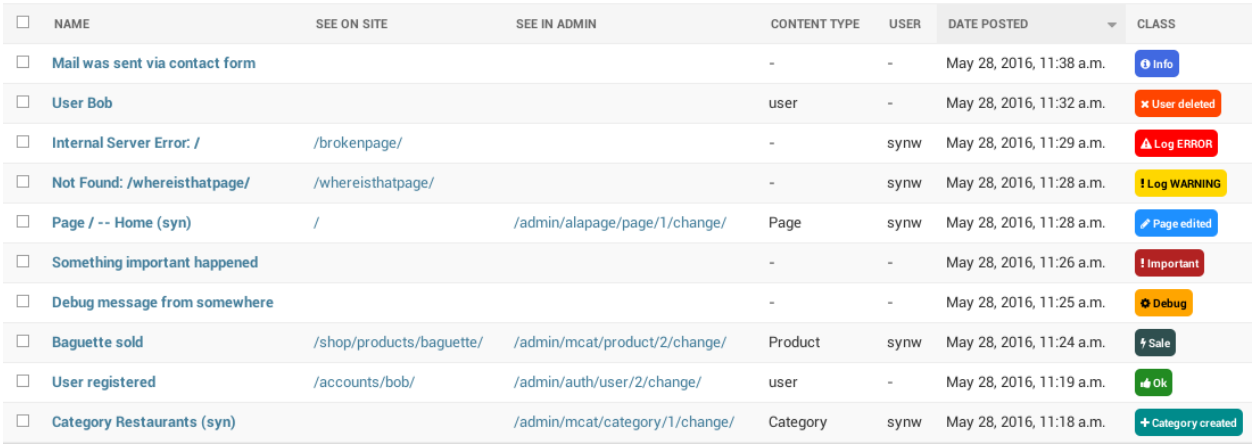
You can define your custom set of event classes and the corresponding css classes to display in the admin. The default values are (check `mqueue/static/mqueue.css`):

```
MQUEUE_EVENT_CLASSES = {
     #~ 'Event class label' : 'css class to apply',
     'Default' : 'mq-label mq-default',
     'Important' : 'mq-label mq-important',
     'Ok' : 'mq-label mq-ok',
     'Info' : 'mq-label mq-info',
     'Debug' : 'mq-label mq-debug',
     'Warning' : 'mq-label mq-warning',
     'Error' : 'mq-label mq-error',
     'Object created' : 'mq-label mq-created',
     'Object edited' : 'mq-label mq-edited',
     'Object deleted' : 'mq-label mq-deleted',
     }
```

Note: if the `event_class` field value is not in MQUEUE_EVENT_CLASSES, the display will fallback to 'Default'.

To use your own event classes set a `MQUEUE_EVENT_CLASSES` setting. Ex:

```
MQUEUE_EVENT_CLASSES = {
#~ 'Event class label' : 'css class to apply',
'Default' : 'mydefaultcssclass',
'User registered' : 'mycssclass',
'Post reviewed' : 'mycssclass mycssclass2',
'Error in some process' : 'mycssclass mycssclass2',
# ...
}
```



NAME	SEE ON SITE	SEE IN ADMIN	CONTENT TYPE	USER	DATE POSTED	CLASS
Mail was sent via contact form			-	-	May 28, 2016, 11:38 a.m.	Info
User Bob			user	-	May 28, 2016, 11:32 a.m.	User deleted
Internal Server Error: /	/brokenpage/		-	synw	May 28, 2016, 11:29 a.m.	Log ERROR
Not Found: /whereisthatpage/	/whereisthatpage/		-	synw	May 28, 2016, 11:28 a.m.	Log WARNING
Page / -- Home (syn)	/	/admin/alapage/page/1/change/	Page	synw	May 28, 2016, 11:28 a.m.	Page edited
Something important happened			-	-	May 28, 2016, 11:26 a.m.	Important
Debug message from somewhere			-	-	May 28, 2016, 11:25 a.m.	Debug
Baguette sold	/shop/products/baguette/	/admin/mcat/product/2/change/	Product	synw	May 28, 2016, 11:24 a.m.	Sale
User registered	/accounts/bob/	/admin/auth/user/2/change/	user	-	May 28, 2016, 11:19 a.m.	Ok
Category Restaurants (syn)		/admin/mcat/category/1/change/	Category	synw	May 28, 2016, 11:18 a.m.	Category created

Event Extra html

You can add some extra html that will display after the event_class label:

```
EVENT_EXTRA_HTML = {
        #~ 'Event class label' : 'html to apply',
        'My event' : ' <blink>!!</blink>',
        }
```

Event Icons

You can provide html to display icons in your `event_class`. The defaults are the font-awesome ones (embeded in mqueue):

```
EVENT_ICONS_HTML = {
     #~ 'Event class label' : 'icon html',
     'Default' : '<i class="fa fa-flash"></i>',
     'Important' : '<i class="fa fa-exclamation"></i>',
     'Ok' : '<i class="fa fa-thumbs-up"></i>',
     'Info' : '<i class="fa fa-info-circle"></i>',
     'Debug' : '<i class="fa fa-cog"></i>',
     'Warning' : '<i class="fa fa-exclamation"></i>',
     'Error' : '<i class="fa fa-exclamation-triangle"></i>',
     'Object edited' : '<i class="fa fa-pencil"></i>',
     'Object created' : '<i class="fa fa-plus"></i>',
     'Object deleted' : '<i class="fa fa-remove"></i>',
     }
```

If you don't want any icons set it empty.

CHAPTER 8

Graphql Api

New in 0.9

It is possible to query for public events, users events and staff events.

Settings

Install with `pip install django-graphql-utils django-filters`

```
INSTALLED_APPS += ("graphene_django", "graphql_utils", "django_filters",)

GRAPHENE = {
   'SCHEMA': 'mqueue.schema.schema'
}

if DEBUG is True:
   GRAPHENE['MIDDLEWARE'] = ['graphene_django.debug.DjangoDebugMiddleware']

# optional: limit number of events to be retrieved in one query (default 100):
MQUEUE_API_MAX_EVENTS = 50
```

Urls:

```
from django.conf import settings
from graphene_django.views import GraphQLView
from graphql_utils.views import TGraphQLView

urlpatterns = [
 # ...
 url(r'^graphql', TGraphQLView.as_view()),
]
```

```
if settings.DEBUG:
   urlpatterns += [url(r'^graphiql', GraphQLView.as_view(graphiql=True)), ]
```

Note: the `/graphql` endpoint is protected by a Django csrf token

Event scope

Use the `scope` parameter in your events to make them available to the Api:

```
MEvent.objects.create(name="Test public event", scope="public")
```

This event will be available for all users.

Possible scope values are: `public`, `users`, `staff`, `superuser` (default)

Queries

Available queries are:

`publicEvents`, `usersEvents`, `staffEvents`, `allEvents` (for the superuser)

Example: public events:

```
{
   publicEvents(first:10) {
      edges {
         node {
            name
            event_class
         }
      }
   }
}
```

Get all error events (must be logged in as superuser):

```
{
   allEvents(eventClass_Icontains: "error") {
      edges {
         node {
            name
            eventClass
            request
            notes
         }
      }
   }
}
```

Available filters: `Icontains`, `Iexact`, `Istartswith`

Filterable fields: `name`, `eventClass`, `datePosted`