django-uwsgi-admin Documentation

Release 2.0.1

django-uwsgi-admin maintainers

Jan 13, 2023

CONTENTS

CHAPTER

ONE

OVERVIEW

docs	
tests	
package	

Django related examples/tricks/modules for uWSGI.

- Free software: MIT license

1.1 Installation

```
pip install django-uwsgi-admin
```

You can also install the in-development version with:

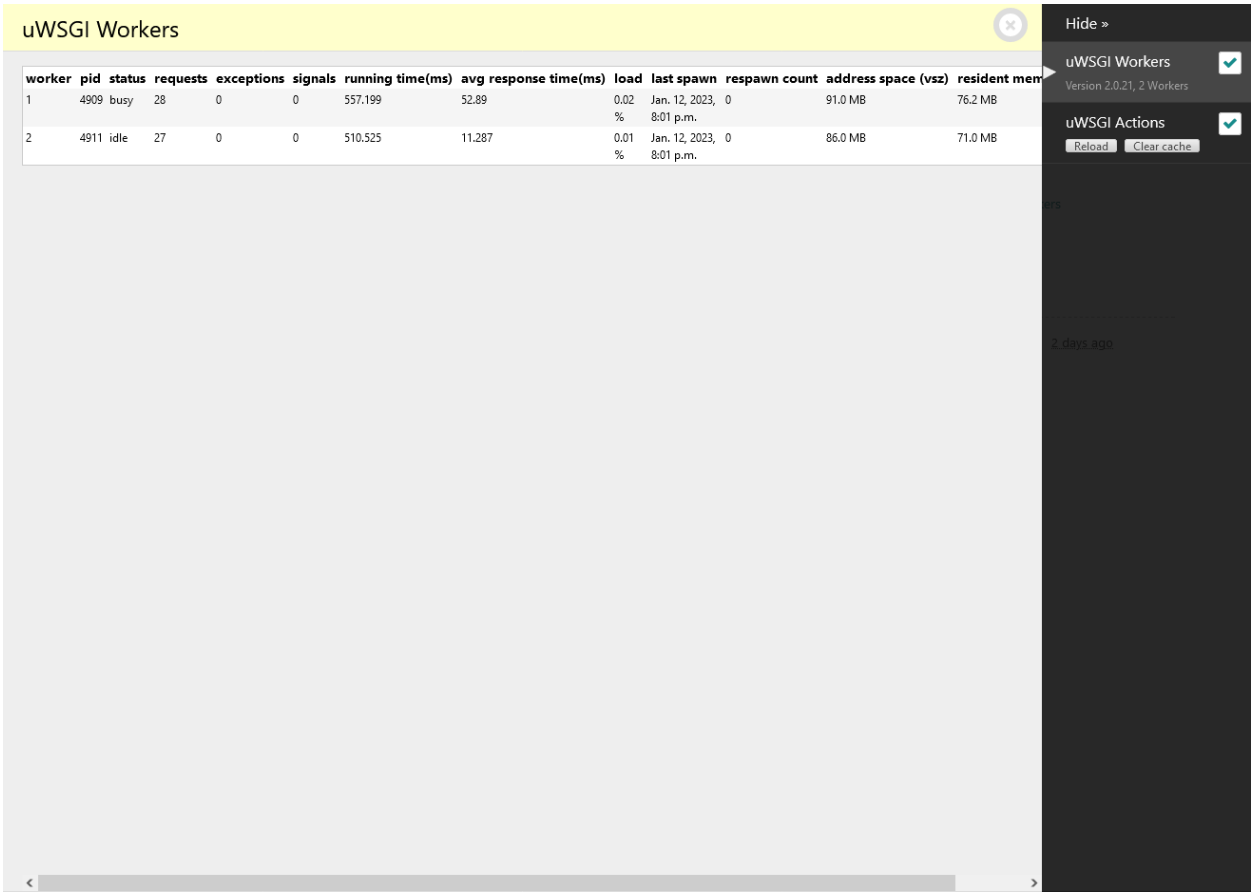
```
pip install https://github.com/ionelmc/django-uwsgi-admin/archive/master.zip
```

1.2 Documentation

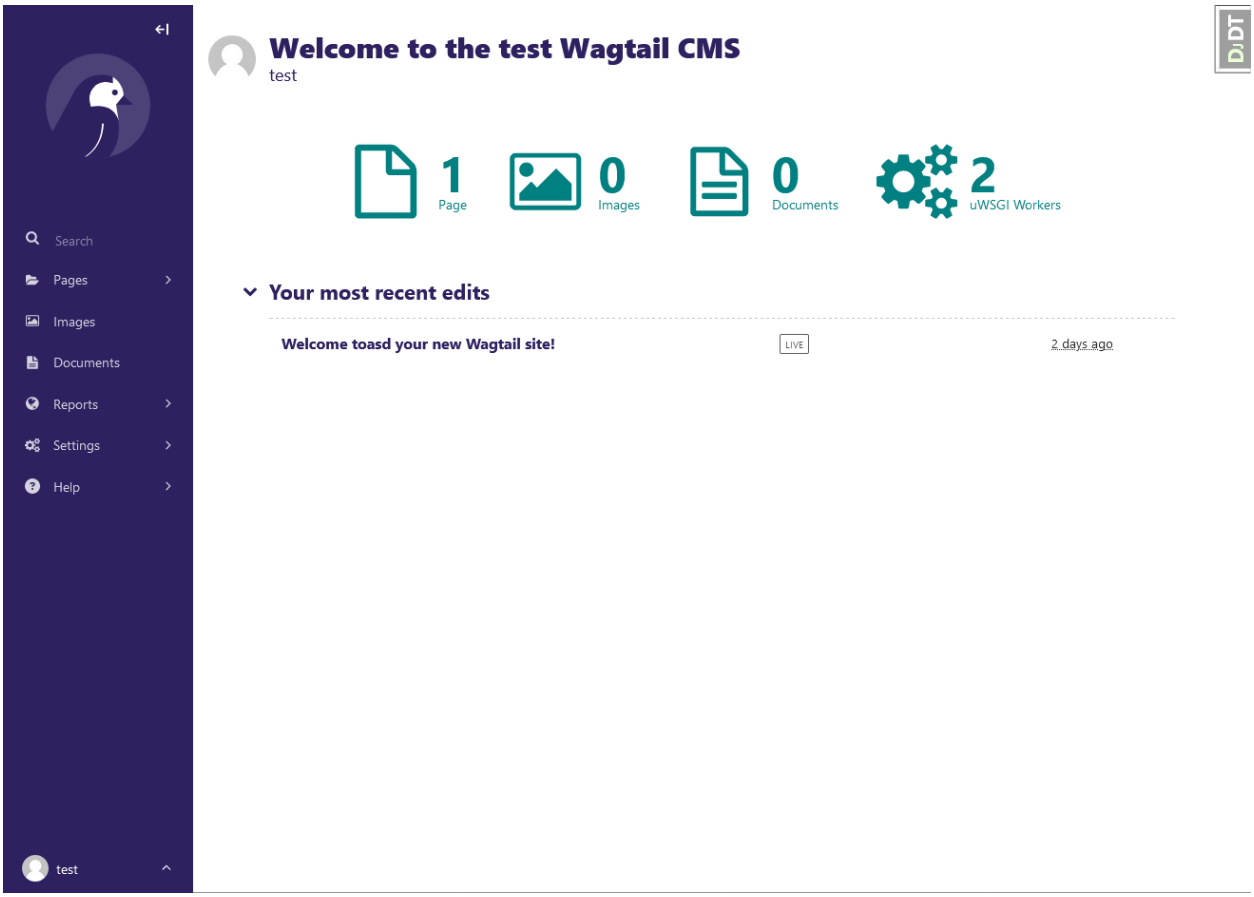
https://django-uwsgi-admin.readthedocs.io/

1.3 Screenshots

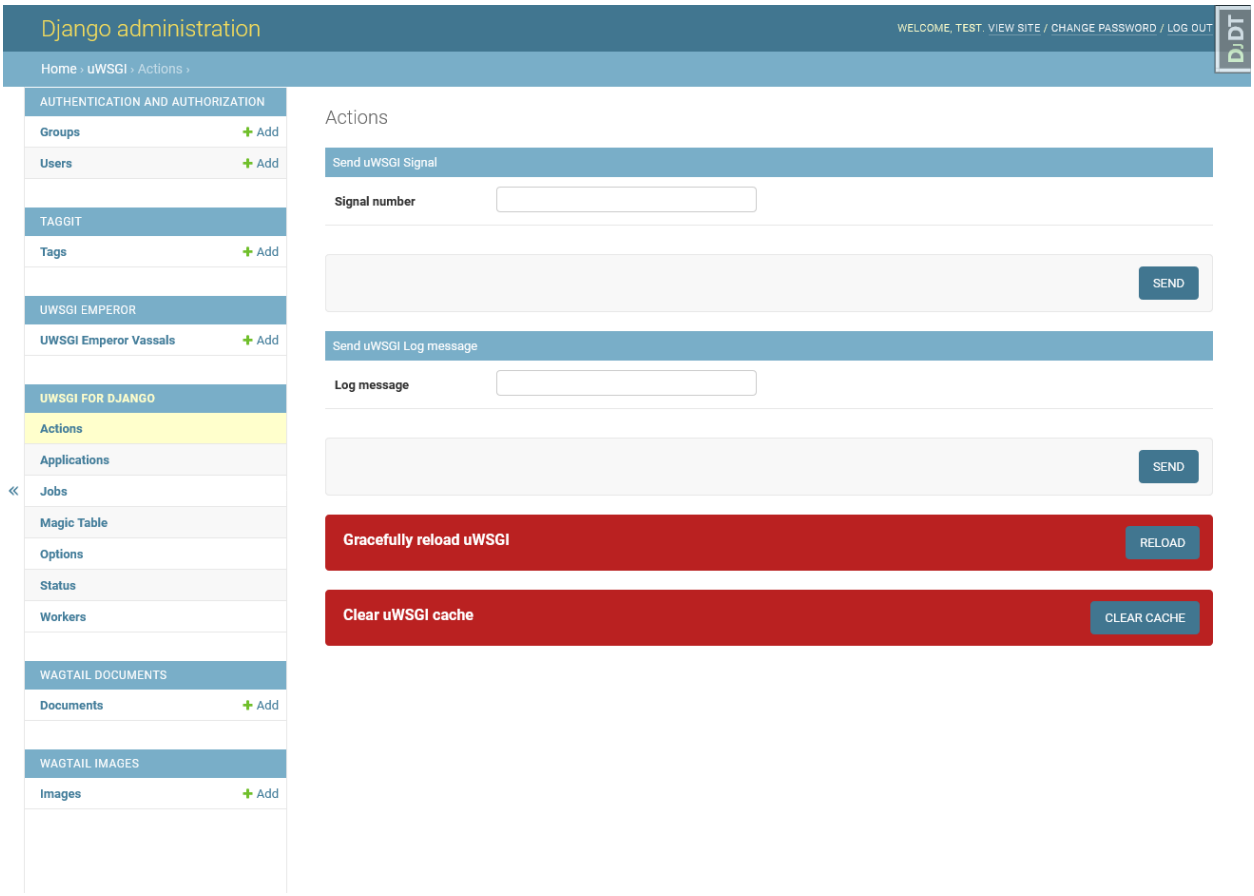
django-debug-toolbar panel

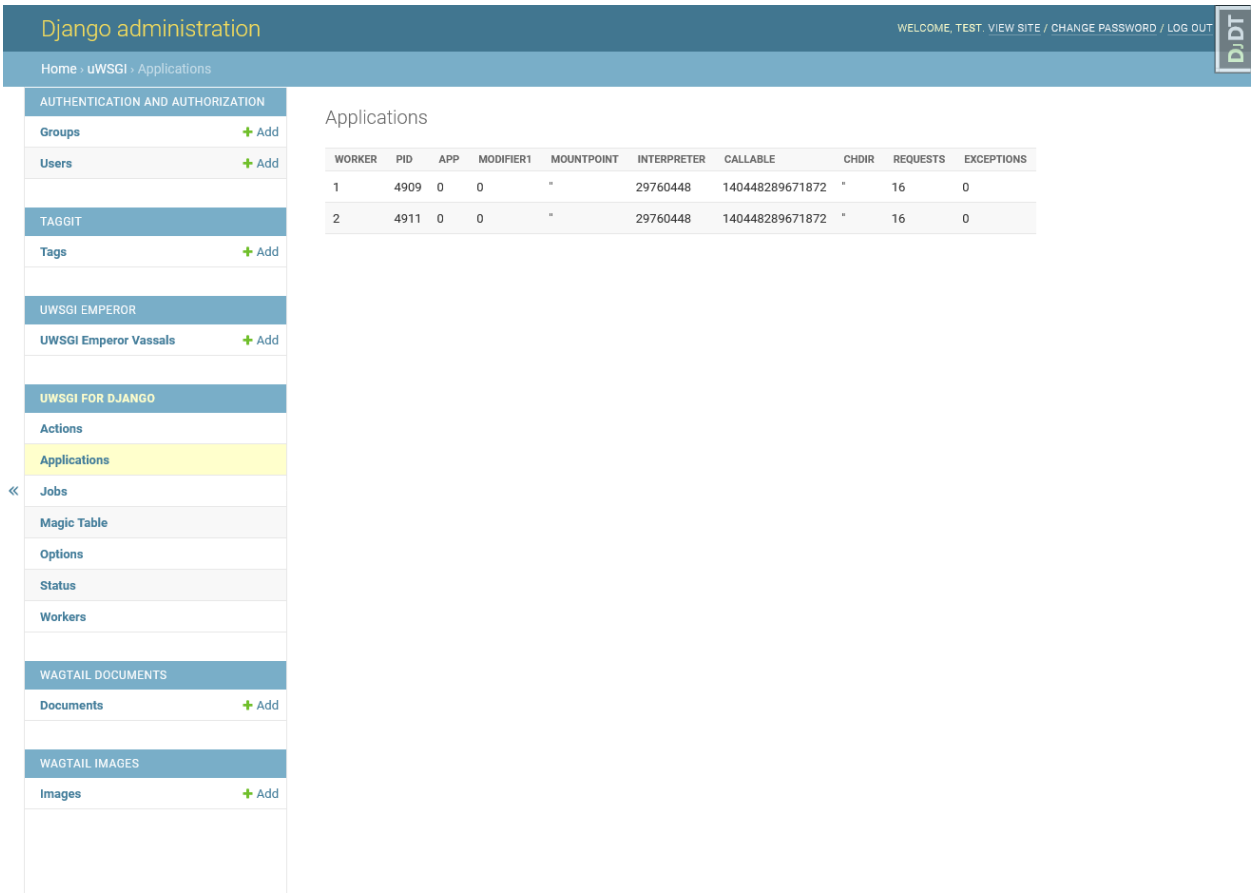


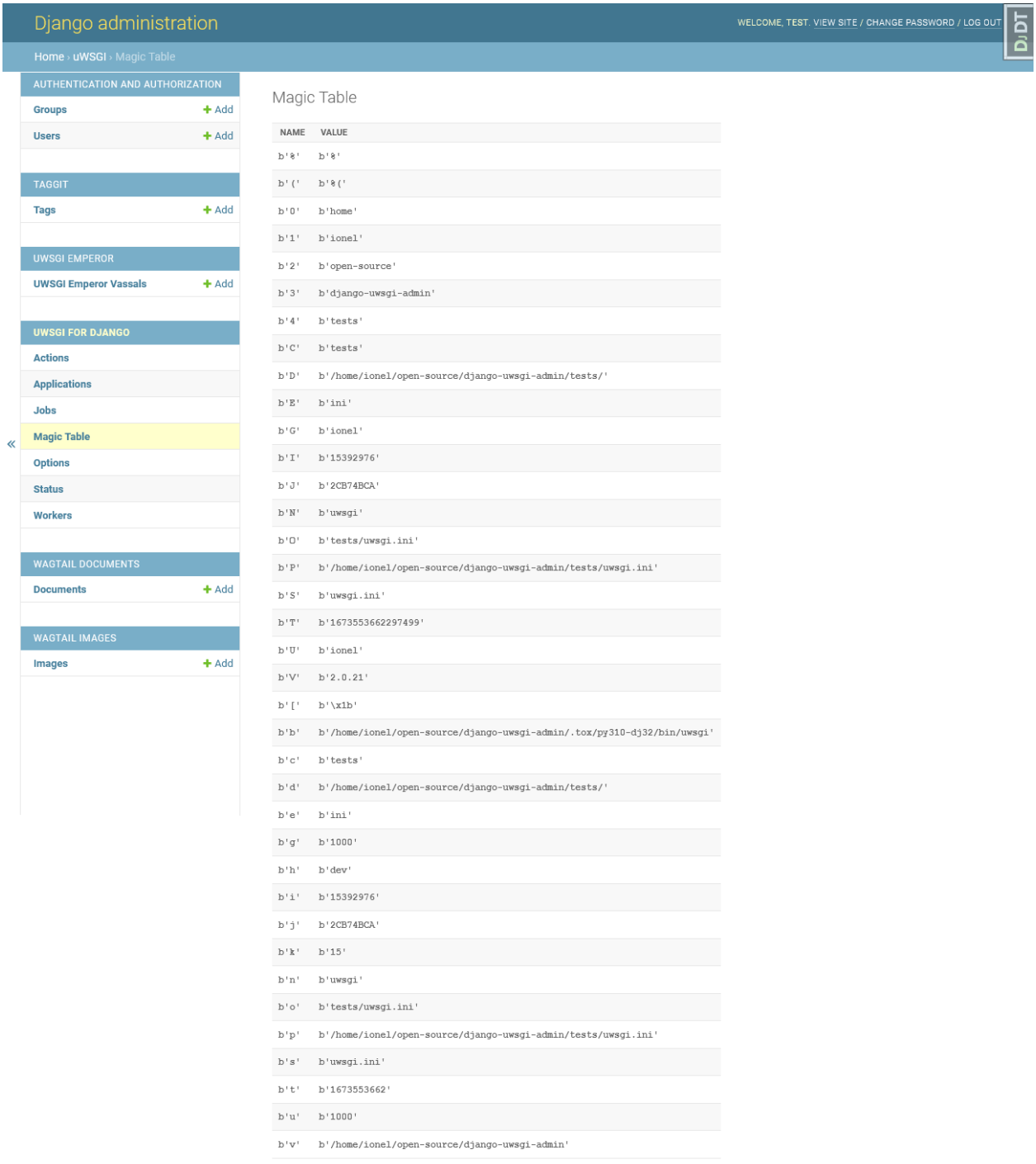
Wagtail admin interface:

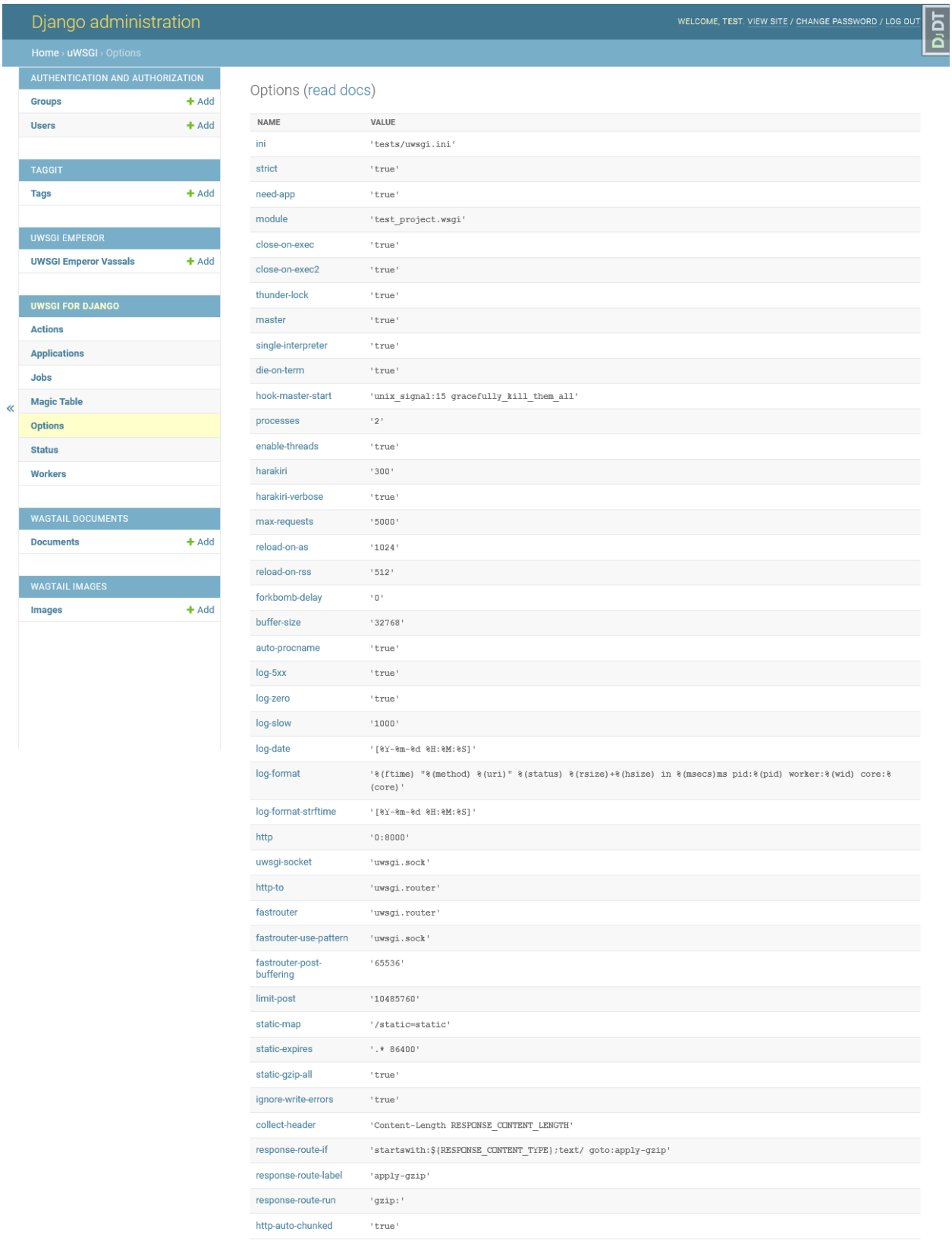


django.contrib.admin interface

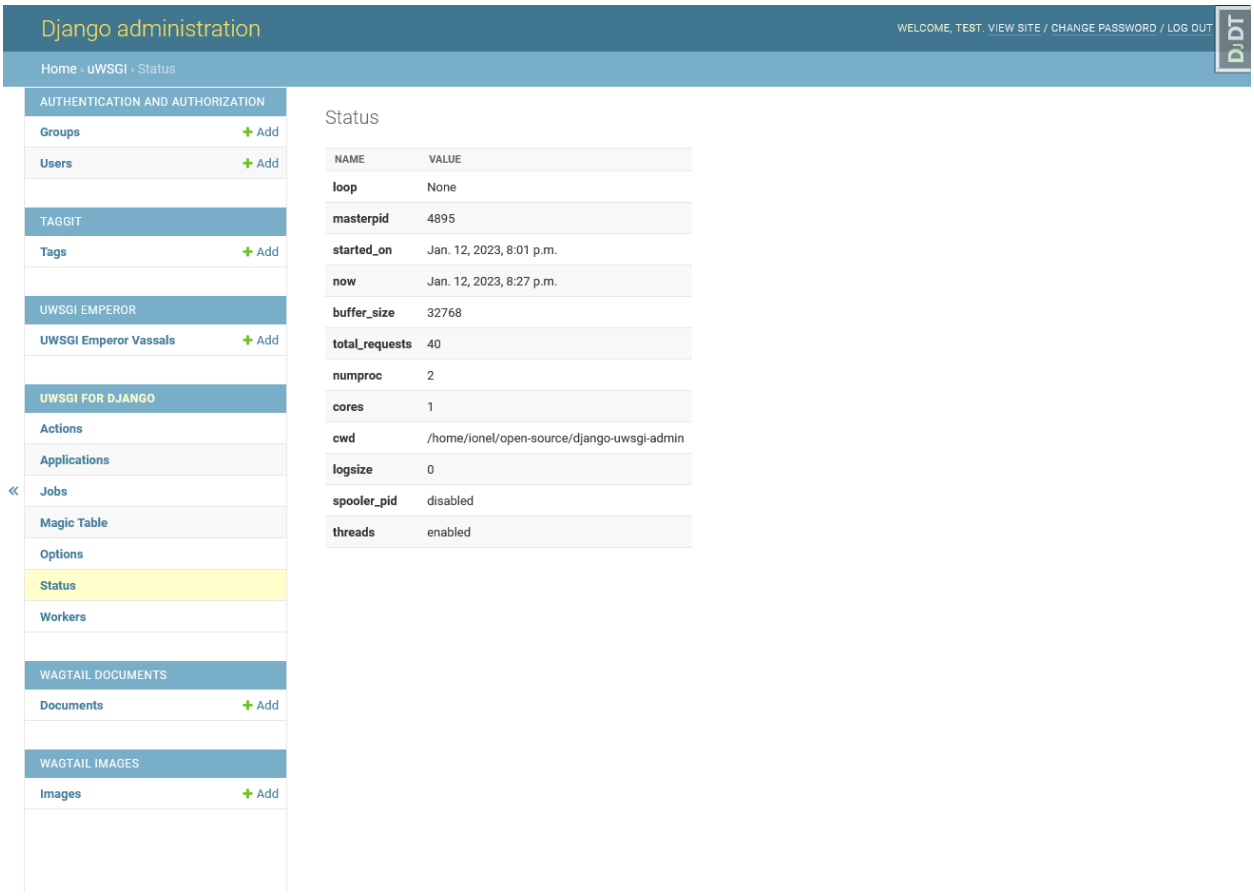

Django administration
WELCOME, TEST. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › uWSGI › Actions ›
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
TAGGIT
Tags + Add
UWSGI EMPEROR
UWSGI Emperor Vassals + Add
UWSGI FOR DJANGO
Actions
Applications
Jobs
Magic Table
Options
Status
Workers
WAGTAIL DOCUMENTS
Documents + Add
WAGTAIL IMAGES
Images + Add
Actions
Send uWSGI Signal
Signal number
SEND
Send uWSGI Log message
Log message
SEND
Gracefully reload uWSGI
RELOAD
Clear uWSGI cache
CLEAR CACHE

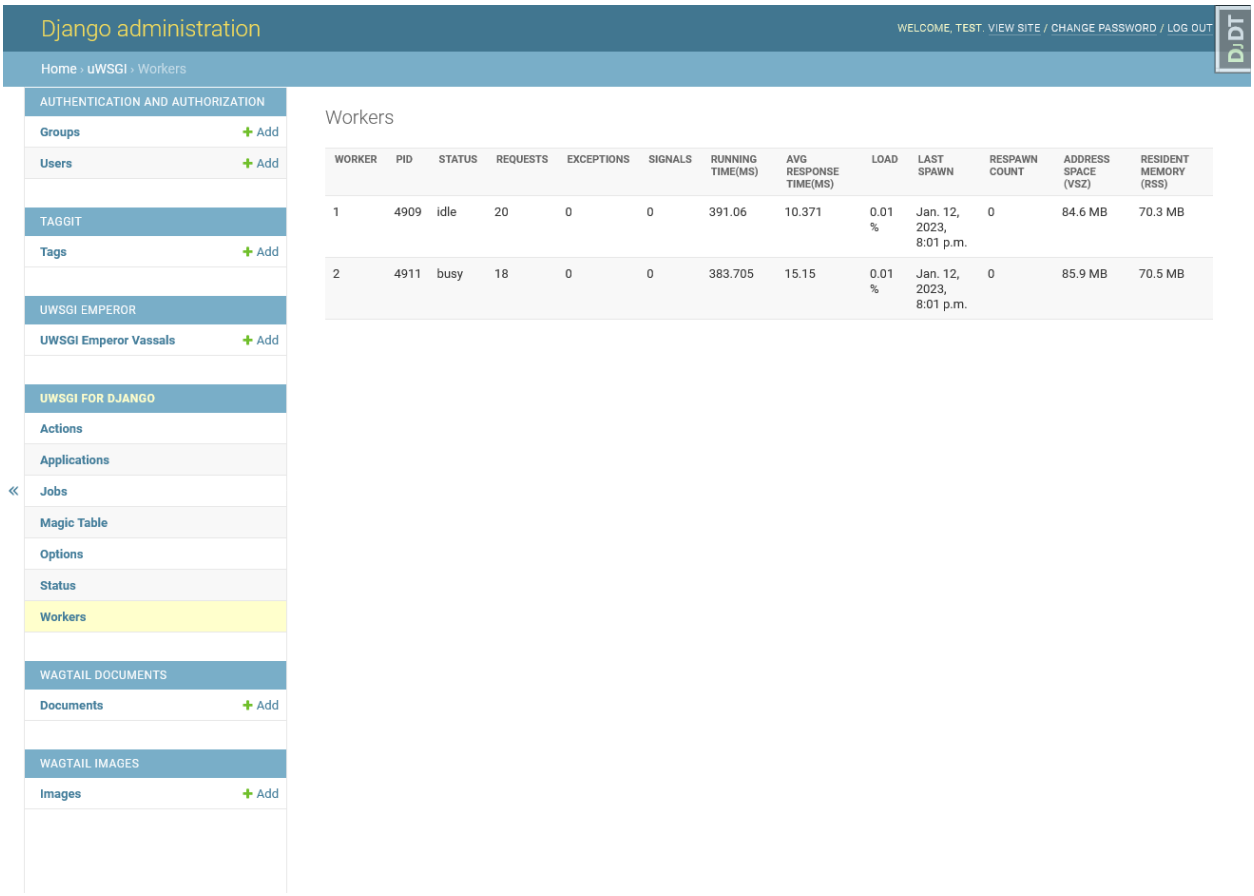

Django administration
WELCOME, TEST. VIEW SITE / CHANGE PASSWORD / LOG OUT
DjDT
Home › uWSGI › Applications
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
TAGGIT
Tags + Add
UWSGI EMPEROR
UWSGI Emperor Vassals + Add
UWSGI FOR DJANGO
Actions
Applications
Jobs
Magic Table
Options
Status
Workers
WAGTAIL DOCUMENTS
Documents + Add
WAGTAIL IMAGES
Images + Add
Applications
WORKER PID APP MODIFIER1 MOUNTPOINT INTERPRETER CALLABLE CHDIR REQUESTS EXCEPTIONS
1 4909 0 0 '' 29760448 140448289671872 '' 16 0
2 4911 0 0 '' 29760448 140448289671872 '' 16 0


Django administration
WELCOME, TEST. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › uWSGI › Magic Table
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
TAGGIT
Tags + Add
UWSGI EMPEROR
UWSGI Emperor Vassals + Add
UWSGI FOR DJANGO
Actions
Applications
Jobs
Magic Table
Options
Status
Workers
WAGTAIL DOCUMENTS
Documents + Add
WAGTAIL IMAGES
Images + Add
Magic Table
NAME VALUE
b'%' b'%'
b'(' b'%('
b'0' b'home'
b'1' b'ionel'
b'2' b'open-source'
b'3' b'django-uwsgi-admin'
b'4' b'tests'
b'C' b'tests'
b'D' b'/home/ionel/open-source/django-uwsgi-admin/tests/'
b'E' b'ini'
b'G' b'ionel'
b'I' b'15392976'
b'J' b'2CB74BCA'
b'N' b'uwsgi'
b'O' b'tests/uwsgi.ini'
b'P' b'/home/ionel/open-source/django-uwsgi-admin/tests/uwsgi.ini'
b'S' b'uwsgi.ini'
b'T' b'1673553662297499'
b'U' b'ionel'
b'V' b'2.0.21'
b'[' b'\x1b'
b'b' b'/home/ionel/open-source/django-uwsgi-admin/.tox/py310-dj32/bin/uwsgi'
b'c' b'tests'
b'd' b'/home/ionel/open-source/django-uwsgi-admin/tests/'
b'e' b'ini'
b'g' b'1000'
b'h' b'dev'
b'i' b'15392976'
b'j' b'2CB74BCA'
b'k' b'15'
b'n' b'uwsgi'
b'o' b'tests/uwsgi.ini'
b'p' b'/home/ionel/open-source/django-uwsgi-admin/tests/uwsgi.ini'
b's' b'uwsgi.ini'
b't' b'1673553662'
b'u' b'1000'
b'v' b'/home/ionel/open-source/django-uwsgi-admin'



NAME	VALUE
ini	'tests/uwsgi.ini'
strict	'true'
need-app	'true'
module	'test_project.wsgi'
close-on-exec	'true'
close-on-exec2	'true'
thunder-lock	'true'
master	'true'
single-interpreter	'true'
die-on-term	'true'
hook-master-start	'unix_signal:15 gracefully_kill_them_all'
processes	'2'
enable-threads	'true'
harakiri	'300'
harakiri-verbose	'true'
max-requests	'5000'
reload-on-as	'1024'
reload-on-rss	'512'
forkbomb-delay	'0'
buffer-size	'32768'
auto-procname	'true'
log-5xx	'true'
log-zero	'true'
log-slow	'1000'
log-date	'[%Y-%m-%d %H:%M:%S]'
log-format	'%(ftime) "%(method) %(uri)" %(status) %(rsize)+%(hsize) in %(msecs)ms pid:%(pid) worker:%(wid) core:%(core)'
log-format-strftime	'[%Y-%m-%d %H:%M:%S]'
http	'0:8000'
uwsgi-socket	'uwsgi.sock'
http-to	'uwsgi.router'
fastrouter	'uwsgi.router'
fastrouter-use-pattern	'uwsgi.sock'
fastrouter-post-buffering	'65536'
limit-post	'10485760'
static-map	'/static=static'
static-expires	'.* 86400'
static-gzip-all	'true'
ignore-write-errors	'true'
collect-header	'Content-Length RESPONSE_CONTENT_LENGTH'
response-route-if	'startswith:${RESPONSE_CONTENT_TYPE};text/ goto:apply-gzip'
response-route-label	'apply-gzip'
response-route-run	'gzip:'
http-auto-chunked	'true'


Django administration
WELCOME, TEST. VIEW SITE / CHANGE PASSWORD / LOG OUT
DjDT
Home › uWSGI › Status
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
TAGGIT
Tags + Add
UWSGI EMPEROR
UWSGI Emperor Vassals + Add
UWSGI FOR DJANGO
Actions
Applications
Jobs
Magic Table
Options
Status
Workers
WAGTAIL DOCUMENTS
Documents + Add
WAGTAIL IMAGES
Images + Add
Status
NAME VALUE
loop None
masterpid 4895
started_on Jan. 12, 2023, 8:01 p.m.
now Jan. 12, 2023, 8:27 p.m.
buffer_size 32768
total_requests 40
numproc 2
cores 1
cwd /home/ionel/open-source/django-uwsgi-admin
logsize 0
spooler_pid disabled
threads enabled



1.4 Development

To run all the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

FEATURES

- Admin page with uWSGI stats (options to reload/stop uWSGI, clear uWSGI cache)
- uWSGI *Cache Backend* for Django
- uWSGI *Email Backend* for Django(send emails via uWSGI's spooler)
- Debug Panel for django-debug-toolbar (offers same functions as admin page)
- Django template loader for embedded into uWSGI files
- Django *Management Command* `runuwsgi` (with live autoreload when DEBUG is True)
- uWSGI config generator
- Django CBV Mixins based on uWSGI decorators
- Forms to send log messages and signals to uWSGI via admin interface

Some features are not added into repo or not yet implemented(See *Todo*)

CHAPTER

THREE

INSTALLATION

At the command line:

```
pip install django-uwsgi-admin
```

By default `django-uwsgi` doesn't require uWSGI as requirement. And here are a few known reasons why:

- Django project is installed into virtualenv and ran in Emperor mode. In this case uWSGI is installed system-wide or into some other virtualenv.
- Some devs love to use system package managers like apt and prefer to install uwsgi other way.
- You need to build uWSGI with custom profile ex: `UWSGI_PROFILE=gevent pip install uwsgi`

You can install django-uwsgi with uWSGI by appending `[uwsgi]` to the install command:

```
pip install 'django-uwsgi-admin[uwsgi]'
```

CHAPTER

FOUR

CONFIGURATION

4.1 Adding django-uwsgi to your project

Add `'django_uwsgi'`, to your `INSTALLED_APPS` in `settings.py`:

```
INSTALLED_APPS += ['django_uwsgi',]
```

CHAPTER

FIVE

DECORATORS

The uWSGI API is very low-level, as it must be language-independent.

That said, being too low-level is not a Good Thing for many languages, such as Python.

Decorators are, in our humble opinion, one of the more kick-ass features of Python, so in the uWSGI source tree you will find a module exporting a bunch of decorators that cover a good part of the uWSGI API.

5.1 Notes

Signal-based decorators execute the signal handler in the first available worker. If you have enabled the spooler you can execute the signal handlers in it, leaving workers free to manage normal requests. Simply pass `target='spooler'` to the decorator.

```
@timer(3, target='spooler')
def hello(signum):
    print("hello")
```

5.2 Example: a Django session cleaner and video encoder

Let's define a `tasks.py` module and put it in the Django project directory.

```
import os
from django.contrib.sessions.models import Session
from django_uwsgi.decorators import cron, spool

@cron(40, 2, -1, -1, -1)
def clear_django_session(num):
    print("it's 2:40 in the morning: clearing django sessions")
    Session.objects.all().delete()

@spool
def encode_video(arguments):
    os.system("ffmpeg -i \"%s\" image%%d.jpg" % arguments['filename'])
```

The session cleaner will be executed every day at 2:40, to enqueue a video encoding we simply need to spool it from somewhere else.

```
from tasks import encode_video

def index(request):
    # launching video encoding
    encode_video.spool(filename=request.POST['video_filename'])
    return render_to_response('enqueued.html')
```

Now run uWSGI with the spooler enabled:

```
[uwsgi]
; a couple of placeholder
django_projects_dir = /var/www/apps
my_project = foobar
; chdir to app project dir and set pythonpath
chdir = %(django_projects_dir)/%(my_project)
pythonpath = %(django_projects_dir)
; load django
module = django.core.handlers:WSGIHandler()
env = DJANGO_SETTINGS_MODULE=%(my_project).settings
; enable master
master = true
; 4 processes should be enough
processes = 4
; enable the spooler (the mytasks dir must exist!)
spooler = %(chdir)/mytasks
; load the task.py module
import = task
; bind on a tcp socket
socket = 127.0.0.1:3031
```

The only especially relevant option is the `import` one. It works in the same way as `module` but skips the WSGI callable search. You can use it to preload modules before the loading of WSGI apps. You can specify an unlimited number of '"import"' directives.

5.3 django_uwsgi.decorators API reference

`django_uwsgi.decorators.`**`postfork`**(*func*)

uWSGI is a preforking (or "fork-abusing") server, so you might need to execute a fixup task after each `fork()`. The `postfork` decorator is just the ticket. You can declare multiple `postfork` tasks. Each decorated function will be executed in sequence after each `fork()`.

```
@postfork
def reconnect_to_db():
    myfoodb.connect()

@postfork
def hello_world():
    print("Hello World")
```

`django_uwsgi.decorators.`**`spool`**(*func*)

The uWSGI spooler can be very useful. Compared to Celery or other queues it is very "raw". The `spool` decorator will help!

```
@spool
def a_long_long_task(arguments):
    print(arguments)
    for i in xrange(0, 10000000):
        time.sleep(0.1)

@spool
def a_longer_task(args):
    print(args)
    for i in xrange(0, 10000000):
        time.sleep(0.5)

# enqueue the tasks
a_long_long_task.spool(foo='bar',hello='world')
a_longer_task.spool({'pippo':'pluto'})
```

The functions will automatically return `uwsgi.SPOOL_OK` so they will be executed one time independently by their return status.

`django_uwsgi.decorators.`**spoolforever**(*func*)

Use `spoolforever` when you want to continuously execute a spool task. A `@spoolforever` task will always return `uwsgi.SPOOL_RETRY`.

```
@spoolforever
def a_longer_task(args):
    print(args)
    for i in xrange(0, 10000000):
        time.sleep(0.5)

# enqueue the task
a_longer_task.spool({'pippo':'pluto'})
```

`django_uwsgi.decorators.`**spoolraw**(*func*)

Advanced users may want to control the return value of a task.

```
@spoolraw
def a_controlled_task(args):
    if args['foo'] == 'bar':
        return uwsgi.SPOOL_OK
    return uwsgi.SPOOL_RETRY

a_controlled_task.spool(foo='bar')
```

`django_uwsgi.decorators.`**rpc**(*"name"*, *func*)

uWSGI RPC is the fastest way to remotely call functions in applications hosted in uWSGI instances. You can easily define exported functions with the @rpc decorator.

```
@rpc('helloworld')
def ciao_mondo_function():
    return "Hello World"
```

`django_uwsgi.decorators.`**signal**(*num)(func*)

You can register signals for the signal framework in one shot.

```
@signal(17)
def my_signal(num):
    print("i am signal %d" % num)
```

django_uwsgi.decorators.**timer**(*interval*, *func*)

Execute a function at regular intervals.

```
@timer(3)
def three_seconds(num):
    print("3 seconds elapsed")
```

django_uwsgi.decorators.**rbtimer**(*interval*, *func*)

Works like @timer but using red black timers.

django_uwsgi.decorators.**cron**(*min*, *hour*, *day*, *mon*, *wday*, *func*)

Easily register functions for the Cron.

```
@cron(59, 3, -1, -1, -1)
def execute_me_at_three_and_fiftynine(num):
    print("it's 3:59 in the morning")
```

Since 1.2, a new syntax is supported to simulate `crontab`-like intervals (every Nth minute, etc.). `*/5 * * * *` can be specified in uWSGI like thus:

```
@cron(-5, -1, -1, -1, -1)
def execute_me_every_five_min(num):
    print("5 minutes, what a long time!")
```

django_uwsgi.decorators.**filemon**(*path*, *func*)

Execute a function every time a file/directory is modified.

```
@filemon("/tmp")
def tmp_has_been_modified(num):
    print("/tmp directory has been modified. Great magic is afoot")
```

django_uwsgi.decorators.**erlang**(*process_name*, *func*)

Map a function as an *Erlang <http://uwsgi-docs.readthedocs.org/en/latest/Erlang.html>* process.

```
@erlang('foobar')
def hello():
    return "Hello"
```

django_uwsgi.decorators.**thread**(*func*)

Mark function to be executed in a separate thread.

Important: Threading must be enabled in uWSGI with the `enable-threads` or `threads <n>` option.

```
@thread
def a_running_thread():
    while True:
        time.sleep(2)
```

(continues on next page)

(continued from previous page)

```
        print("i am a no-args thread")

@thread
def a_running_thread_with_args(who):
    while True:
        time.sleep(2)
        print("Hello %s (from arged-thread)" % who)

a_running_thread()
a_running_thread_with_args("uWSGI")
```

You may also combine `@thread` with `@postfork` to spawn the postfork handler in a new thread in the freshly spawned worker.

```
@postfork
@thread
def a_post_fork_thread():
    while True:
        time.sleep(3)
        print("Hello from a thread in worker %d" % uwsgi.worker_id())
```

`django_uwsgi.decorators.`**lock**(*func*)

This decorator will execute a function in fully locked environment, making it impossible for other workers or threads (or the master, if you're foolish or brave enough) to run it simultaneously. Obviously this may be combined with @postfork.

```
@lock
def dangerous_op():
    print("Concurrency is for fools!")
```

`django_uwsgi.decorators.`**mulefunc**([*mulespec*], *func*)

Offload the execution of the function to a *mule <http://uwsgi-docs.readthedocs.org/en/latest/Mules.html>*. When the offloaded function is called, it will return immediately and execution is delegated to a mule.

```
@mulefunc
def i_am_an_offloaded_function(argument1, argument2):
    print argument1,argument2
```

You may also specify a mule ID or mule farm to run the function on. Please remember to register your function with a uwsgi import configuration option.

```
@mulefunc(3)
def on_three():
    print "I'm running on mule 3."

@mulefunc('old_mcdonalds_farm')
def on_mcd():
    print "I'm running on a mule on Old McDonalds' farm."
```

`django_uwsgi.decorators.`**harakiri**(*time*, *func*)

Starting from uWSGI 1.3-dev, a customizable secondary harakiri subsystem has been added. You can use this decorator to kill a worker if the given call is taking too long.

```
@harakiri(10)
def slow_function(foo, bar):
    for i in range(0, 10000):
        for y in range(0, 10000):
            pass

# or the alternative lower level api

uwsgi.set_user_harakiri(30) # you have 30 seconds. fight!
slow_func()
uwsgi.set_user_harakiri(0) # clear the timer, all is well
```

CHAPTER

SIX

EMAIL BACKEND

A Django backend for e-mail delivery using uWSGI Spool to queue deliveries.

6.1 Usage

First, add uWSGI backend in your settings file.

```
EMAIL_BACKEND = 'django_uwsgi.mail.EmailBackend'
```

And send your e-mails normally.

```
from django.core.mail import send_mail

send_mail('Subject here', 'Here is the message.', 'from@example.com',
    ['to@example.com'], fail_silently=False)
```

6.2 Note

You must setup uwsgi spooler. Example ini:

```
plugin = spooler
spooler = /tmp
spooler-import = django_uwsgi.tasks
```

or use built in management command *runuwsgi*

6.3 Changing the backend

By default the 'django.core.mail.backends.smtp.EmailBackend' is used for the real e-mail delivery. You can change that using:

```
UWSGI_EMAIL_BACKEND = 'your.backend.EmailBackend'
```

6.4 django-configurations

If you're using django-configurations in your project, you must setup importer as mentioned in django-configurations docs for celery

CHAPTER

SEVEN

CACHE BACKEND

7.1 Installation

change settings to:

```
CACHES = {
    'default': {
        'BACKEND': 'django_uwsgi.cache.UwsgiCache',

        # and optionally, if you used a different cache name
        'LOCATION': 'foobar'
    }
}
```

7.2 django-confy

if you're using django-confy:, you can use url like:

```
CACHE_URL=uwsgi://foobar
```

7.3 Settings

`UWSGI_CACHE_FALLBACK`

- `False` - raises Exception if `uwsgi` cannot be imported.
- `True` (default) - if uwsgi is not importable this cache backend will alias to LocMemCache. Note that south or other management commands might try to load the cache backend so this is why it's the default.

CHAPTER

EIGHT

MANAGEMENT COMMAND

8.1 runuwsgi

```
python manage.py runuwsgi
```

8.2 runuwsgi options:

8.3 http

```
python manage.py runuwsgi http=8000
```

8.4 socket

```
python manage.py runuwsgi socket=/tmp/projectname-uwsgi.sock
```

8.5 Other options

Any other options can be passed via environment variables, prefixed with *UWSGI_* and converted to upper-case, or as key-value pairs in a dictionary named *UWSGI* in settings.

Options from *UWSGI* in settings are passed to uwsgi as INI, which allows passing multi-value options. Example:

```
UWSGI = {
    "module": "my.project.wsgi",
    "attach-daemon": ["memcached -p 11311", "celery -A my.project.tasks worker"]
}
```

CHAPTER

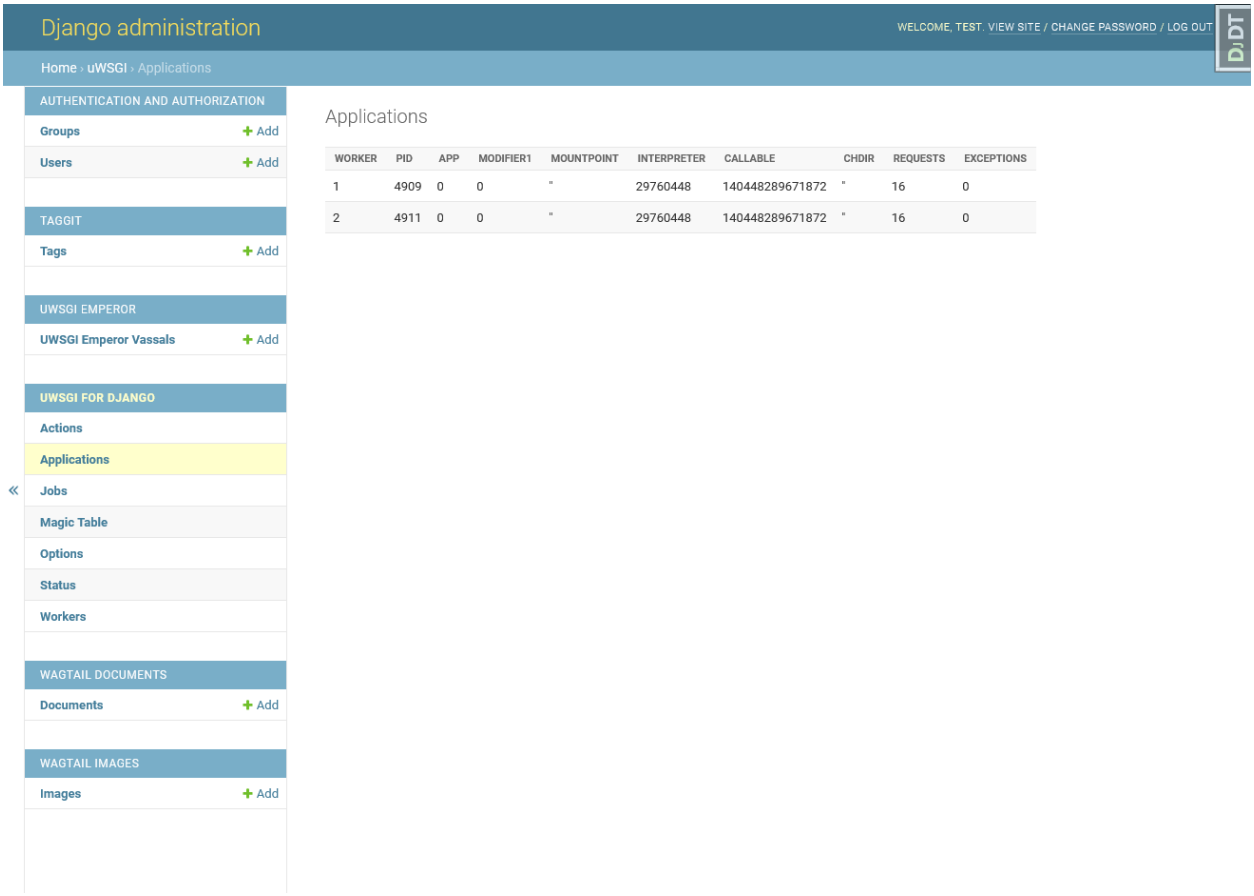
NINE

EMPEROR

you can use *django_uwsgi.emperor* module if you want to store vassals configs in PostgreSQL database.

Simply add *'django_uwsgi.emperor',* into *INSTALLED_APPS*

```
INSTALLED_APPS += ['django_uwsgi.emperor',]
```



Populate vassals via django admin interface and start uwsgi with command like:

```
uwsgi --plugin emperor_pg --emperor "pg://host=127.0.0.1 user=foobar dbname=emperor;
↪SELECT name,config,ts FROM vassals WHERE enabled = True"
```

Each time vassal added, removed, updated, enabled or disabled - uwsgi will start/stop it or reload.

CHAPTER

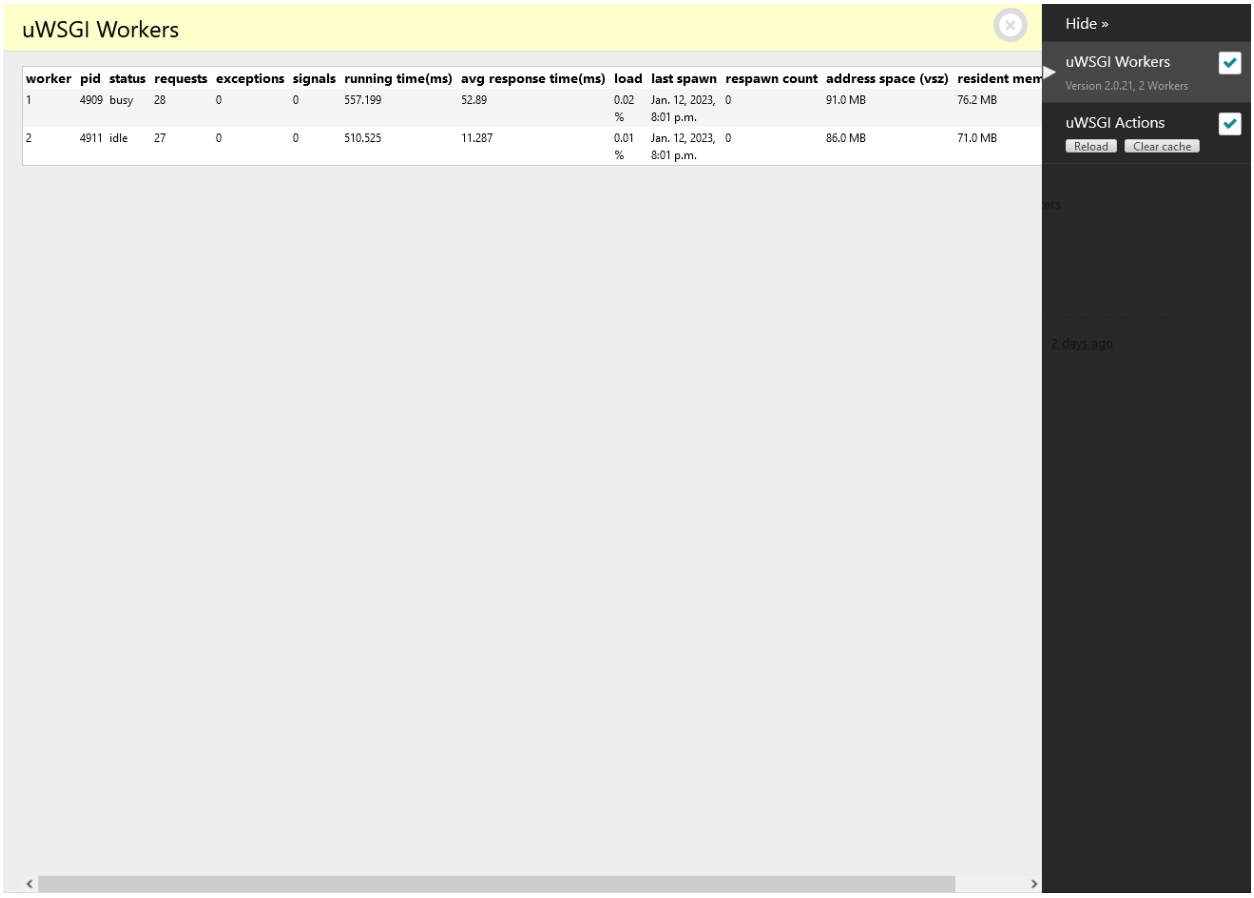
TEN

INTEGRATIONS

10.1 Django-Debug-Toolbar

If you're using django-debug-toolbar, you can add:

```
DEBUG_TOOLBAR_PANELS += ['django_uwsgi.panels.UwsgiPanel',]
```



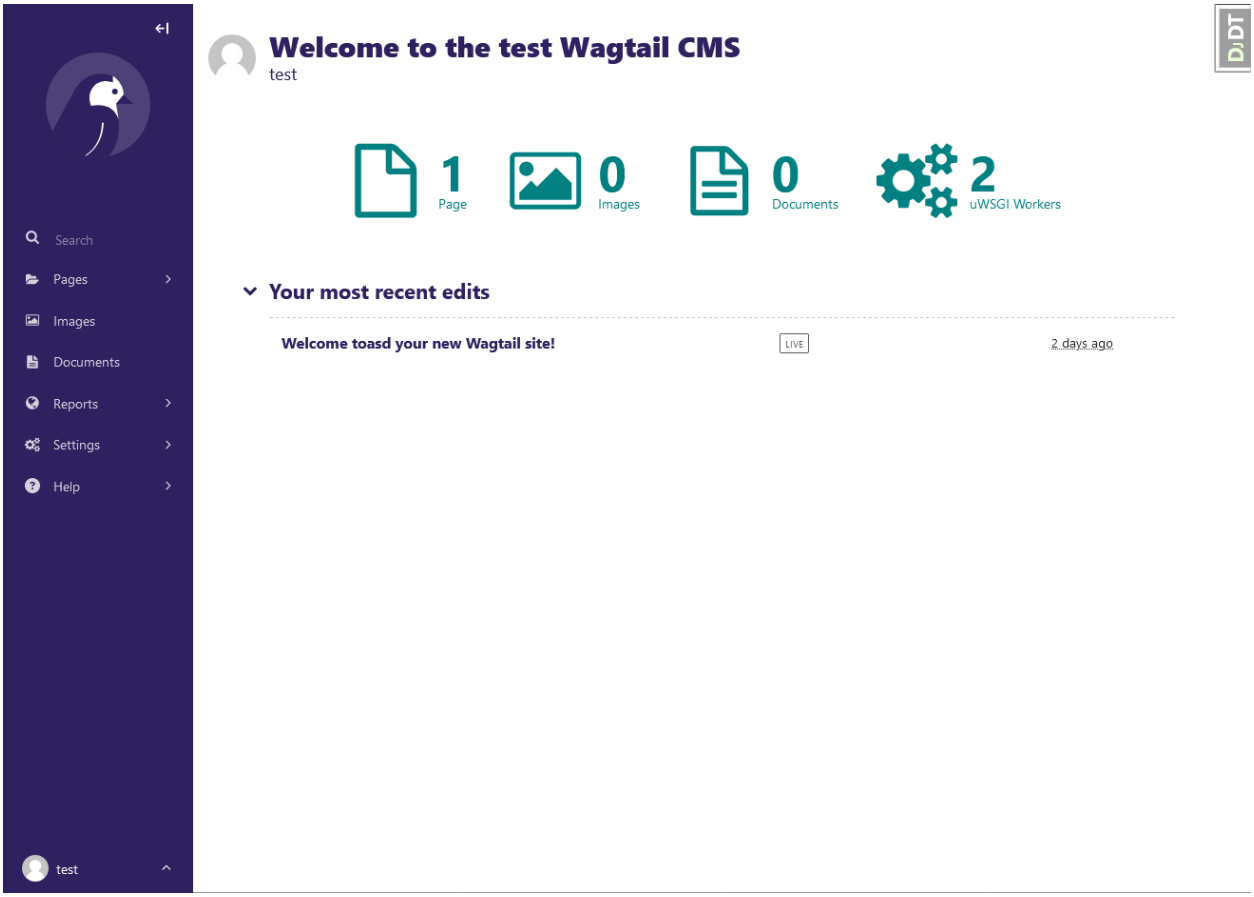
10.2 Wagtail

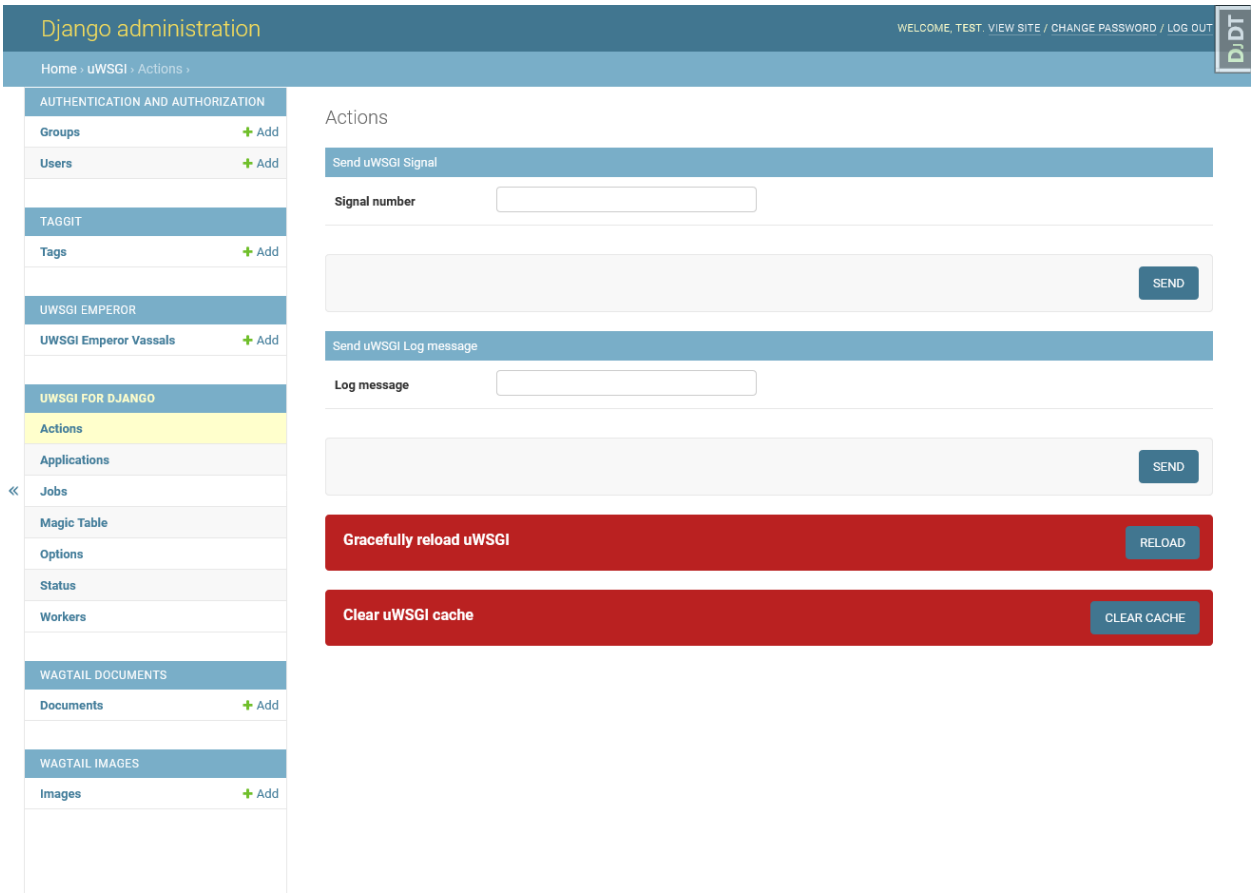
If you're using Wagtail:

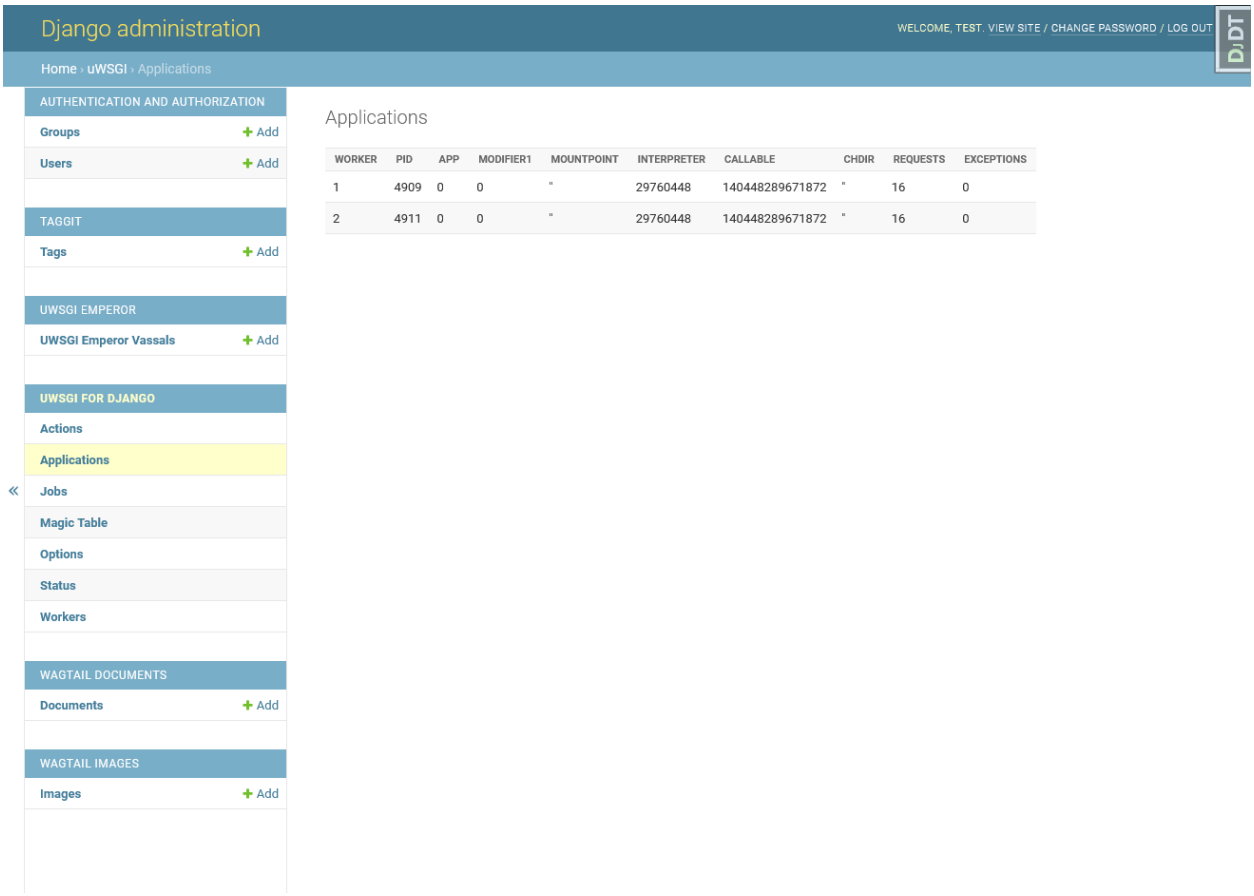
There is *wagtail_hooks.py* file available and Wagtail will read it automatically

And you don't have to add *django_uwsgi* into urls.py

Wagtail admin interface:




Django administration
WELCOME, TEST. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › uWSGI › Actions ›
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
TAGGIT
Tags + Add
UWSGI EMPEROR
UWSGI Emperor Vassals + Add
UWSGI FOR DJANGO
Actions
Applications
Jobs
Magic Table
Options
Status
Workers
WAGTAIL DOCUMENTS
Documents + Add
WAGTAIL IMAGES
Images + Add
Actions
Send uWSGI Signal
Signal number
SEND
Send uWSGI Log message
Log message
SEND
Gracefully reload uWSGI
RELOAD
Clear uWSGI cache
CLEAR CACHE


Django administration
WELCOME, TEST. VIEW SITE / CHANGE PASSWORD / LOG OUT
DjDT
Home › uWSGI › Applications
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
TAGGIT
Tags + Add
UWSGI EMPEROR
UWSGI Emperor Vassals + Add
UWSGI FOR DJANGO
Actions
Applications
Jobs
Magic Table
Options
Status
Workers
WAGTAIL DOCUMENTS
Documents + Add
WAGTAIL IMAGES
Images + Add
Applications
WORKER PID APP MODIFIER1 MOUNTPOINT INTERPRETER CALLABLE CHDIR REQUESTS EXCEPTIONS
1 4909 0 0 " 29760448 140448289671872 " 16 0
2 4911 0 0 " 29760448 140448289671872 " 16 0

CHAPTER

ELEVEN

TODO

- Tests
- uWSGI config generator
- Improve Docs
- Translations?
- Good cache panel
- Ability to add cronjobs/filemonitors via admin interface
- Options for sendfile if uwsgi serving files

Some code is borrowed from projects I did earlier and some code is still not added yet, but does exists in my projects.

CHAPTER

TWELVE

REFERENCE

12.1 django_uwsgi

CHAPTER

THIRTEEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

13.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

13.2 Documentation improvements

django-uwsgi-admin could always use more documentation, whether as part of the official django-uwsgi-admin docs, in docstrings, or even on the web in blog posts, articles, and such.

13.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/ionelmc/django-uwsgi-admin/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

13.4 Development

To set up *django-uwsgi-admin* for local development:

1. Fork django-uwsgi-admin (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/django-uwsgi-admin.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

13.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

13.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

CHAPTER

FOURTEEN

AUTHORS

Roberto De Ioris, Unbit, <roberto@unbit.it> Eugene MechanisM, MechanisM, <eugene@mechanism.name> Ionel Cristian Mărieș, <contact@ionelmc.ro> Jayson Reis, jaysonsantos, <santosdosreis@gmail.com> Alan Justino da Silva, alanjds, <alan.justino@yahoo.com.br> Michael Fladischer, fladi, <michael@fladi.at> Paul Bailey, pizzapanther Arkadiusz Adamski, ar4s Dominik George, Natureshadow, <nik@naturalnet.de>

CHAPTER

FIFTEEN

CHANGELOG

15.1 2.0.1 (2023-01-13)

- UwsgiWorkersPanel no longer tries to generate stats if there's no uwsgi.

15.2 2.0.0 (2023-01-12)

- Removed the decorators module, something that only existed to avoid installing a separate package. Instead you should install the updated uwsgidecorators package.
- Removed `django_uwsgi.template.Loader` (and the whole module) as it was broken and pretty hard to test without a custom build of uWSGI.
- Split all sections in the Status page into seperate admin pages: Actions, Applications, Jobs, Magic Table, Options, Status and Workers.
- Removed the old django debug toolbar and replaced with 2 new panes:
  - `django_uwsgi.panels.UwsgiWorkersPanel`
  - `django_uwsgi.panels.UwsgiActionsPanel`

15.3 1.0.0 (2023-01-10)

- Removed the `runuwsgi` management command as it was very broken. Yes, I've looked at django-uwsgi-ng (another fork, which has lots of changes for that command) and it's still pretty unusable in general (expects a certain project layout, and still generates weird if not broken configuration).

  Instead you should own your uWSGI configuration and not lets some tool generate it for you as some of the options have high impact on the behavior and performance of uWSGI.
- Fixed stats page title.
- Made clear cache and reload actions be performed safely over POST requests (previously they were GET requests).

15.4 0.3.0 (2023-01-09)

Forked from https://github.com/unbit/django-uwsgi this adds:

- Support for latest Django releases (3.2+).
- A basic integration test suite.
- Removed lots of old compat cruft.
- Integrated the uWSGI stats pane directly in the Django admin. Adding urls manually is no longer necessary.
- Removed the old wagtail-styled admin page (it was broken anyway).

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX