
django-kaio Documentation

Release 0.14.0

APSL

Dec 05, 2018

Contents

1	Installation	3
1.1	Configuration with django-configurations	3
2	How it works	5
2.1	settings.py	5
3	Management scripts	7
3.1	apsettings	7
3.2	generate_ini	7
4	Mixins	9
4.1	CachesMixin	9
4.2	CeleryMixin	10
4.3	CmsMixin	10
4.4	CompressMixin	11
4.5	DatabaseMixin	12
4.6	DebugMixin	12
4.7	EmailMixin	13
4.8	FilerMixin	13
4.9	LogsMixin	14
4.10	PathsMixin	14
4.11	SecurityMixin	15
4.12	StorageMixin	15
4.13	WhiteNoiseMixin	16
5	Application example	17
5.1	Example from scratch. The kiosk	17
6	Indices and tables	21

Django-kaio is a django-package that helps us to configure our django project. The values of the configuration can come from an .ini file or from environment settings.

The values are casted automatically, first trying to cast to *int*, then to *bool* and finally to *string*.

Also note that we can create class-based configurations settings, as [django-configurations](#) do.

Also includes:

- if the .ini file does not exist set the default values
- searches the .ini file in the current and parent directories
- managanement script to let us see the current project configuration
- management script to generate the .ini file with the default values
- uses django-configurations in order to be able to create class based settings
- mixins for standard configurations, such as Paths, Filer, Cache, Database...

To install the package

```
pip install django-kaio
```

Then you've to append `kaio` to `INSTALLED_APPS` in your settings.

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

1.1 Configuration with django-configurations

To use class based settings, we need to configure `django-configurations`. It's all explained [here](#).

1.1.1 Modifying `wsgi.py` and `manage.py`

We need to configure two files of our project: `manage.py` and `wsgi.py`

- `manage.py`

```
#!/usr/bin/env python  
  
import os  
import sys  
  
if __name__ == "__main__":  
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'main.settings')  
    os.environ.setdefault('DJANGO_CONFIGURATION', 'Base')  
  
    from configurations.management import execute_from_command_line
```

```
execute_from_command_line(sys.argv)
```

- wsgi.py

```
import os

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'main.settings')
os.environ.setdefault('DJANGO_CONFIGURATION', 'Base')

from configurations.wsgi import get_wsgi_application

application = get_wsgi_application()
```


The simplest way to get a param value is:

```
from apconf import Options

opts = Options()
APP_SLUG = opts.get('APP_SLUG', 'apsl-app')
```

We get the APP_SLUG, with the default value 'apsl-app'. Besides, *kaio* stores internally the request default value, in order to inform the management scripts. (See below).

2.1 settings.py

We configure the settings through classes, using *django-configurations*. We can use the mixins, so that the repetitive configurations rest into the mixin, centralizing the parametrization and saving code.

Important Make sure that *Settings* is the last class in the class definition:

Basic app settings sample:

```
import os
from os.path import join

from configurations import Configuration
from django.contrib.messages import constants as messages
from kaio import Options
from kaio.mixins import (CachesMixin, DatabasesMixin, CompressMixin, LogsMixin,
                        PathsMixin, SecurityMixin, DebugMixin, WhiteNoiseMixin)

opts = Options()

class Base(CachesMixin, DatabasesMixin, CompressMixin, PathsMixin, LogsMixin,
```

```
        SecurityMixin, DebugMixin, WhiteNoiseMixin, Configuration):
    """
    Project settings for development and production.
    """

    DEBUG = opts.get('DEBUG', True)

    THUMBNAIL_FORCE_OVERWRITE = True

    BASE_DIR = opts.get('APP_ROOT', None)
    APP_SLUG = opts.get('APP_SLUG', 'test-project')
    SITE_ID = 1
    SECRET_KEY = opts.get('SECRET_KEY', 'key')

    USE_I18N = True
    USE_L10N = True
    USE_TZ = True
    LANGUAGE_CODE = 'es'
    TIME_ZONE = 'Europe/Madrid'

    ROOT_URLCONF = 'main.urls'
    WSGI_APPLICATION = 'main.wsgi.application'

    INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.sites',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'kaio',
        '...',
    ]

    MIDDLEWARE = [
        'django.middleware.security.SecurityMiddleware',
        'django.middleware.locale.LocaleMiddleware',
        'django.contrib.sessions.middleware.SessionMiddleware',
        'django.middleware.common.CommonMiddleware',
        'django.middleware.csrf.CsrfViewMiddleware',
        'django.contrib.auth.middleware.AuthenticationMiddleware',
        'django.contrib.messages.middleware.MessageMiddleware',
        'django.middleware.clickjacking.XFrameOptionsMiddleware',
    ]
```

Using mixins, almost we have only to configure the `INSTALLED_APPS`. For further configurations we'll adding more mixins.

Management scripts

We have to management scripts available in order to see the current configurations values and to generate a file with default values into the standard output.

3.1 apsettings

We use it to see the current configurations values.

```
python manage.py apsettings
```

It shows the current configuration. In three columns: * final values into the settings * params into the .ini file * param default value

3.2 generate_ini

We use it to generate a file with default values into the standard output.

```
python manage.py generate_ini
```


The mixins are defined in `kaio/mixins` and inherit from **Object**. They are defined from a function that takes the name from the `.ini` (onwards `app.ini`) file section.

The params into the `app.ini` file are set without quotation marks, either are numbers, texts, strings, etc.

4.1 CachesMixin

This mixin allows us to configure the cache of our application. It is intended for use with `Redis` in production. If a cache type is not defined, it means that we have `dummy` cache.

```
from kaio.mixins import CachesMixin
```

Section: Cache

Parameters

CACHE_TYPE cache type, by default `locmem`, options: `locmem`, `redis`, `dummy`

CACHE_REDIS_DB redis database number that we'll use as cache into redis. By default, `2`.

CACHE_REDIS_PASSWORD Password for redis. By default without password.

REDIS_HOST redis host name. By default `localhost`

REDIS_PORT port of the redis server. By default `6379`

CACHE_PREFIX prefix to use in the cache keys for the project. By default is the project `SLUG`.

CACHE_TIMEOUT Cache expiration time. By default `3600` seconds, 1 hour.

CACHE_MAX_ENTRIES Maximum number of cached entries. By default `10000`.

CachesMixin also allows to configure the cache for sessions. You must set `SESSION_ENGINE = 'django.contrib.sessions.backends.cache'` or `'django.contrib.sessions.backends.cached_db'`. By default use almost same settings as default cache.

SESSION_CACHE_TYPE cache type, by default `CACHE_TYPE`, options: `redis`

SESSION_CACHE_REDIS_DB redis database number that we'll use as cache into redis. By default, 3.

SESSION_CACHE_REDIS_PASSWORD Password for redis. By default without password.

SESSION_REDIS_HOST redis host name. By default `REDIS_HOST`

SESSION_REDIS_PORT port of the redis server. By default `REDIS_PORT`

SESSION_CACHE_PREFIX prefix to use in the cache keys for the project. By default `CACHE_PREFIX_session`.

SESSION_CACHE_TIMEOUT Cache expiration time. By default `None` (no timeout).

SESSION_CACHE_MAX_ENTRIES Maximum number of cached entries. By default `1000000`.

SESSION_CACHE_ALIAS Selects the cache to use for sessions. By default `sessions`.

4.2 CeleryMixin

This mixin allows us to configure [Celery](#) in case we use it in our application.

```
from kaio.mixins import CeleryMixin
```

Section: Cache

Parameters

CELERY_RESULT_BACKEND `todo`

CELERY_ALWAYS_EAGER Default `True`.

CELERY_EAGER_PROPAGATES_EXCEPTIONS Default `True`.

CELERY_IGNORE_RESULT Default `True`.

CELERY_MAX_CACHED_RESULTS Default `5000`.

BROKER_TYPE Default `redis`.

CELERY_REDIS_BROKER_DB Default `0`.

CELERY_REDIS_RESULT_DB Default `0`.

RABBITMQ_HOST Default `localhost`.

RABBITMQ_PORT Default `5672`.

RABBITMQ_USER Default `guest`.

RABBITMQ_PASSWD Default `guest`.

RABBITMQ_VHOST Default `/`.

BROKER_URL `todo`

4.3 CmsMixin

Warning: Deprecated mixin

Mixin that helps us to get the languages configured on the project.

```
from kaio.mixins import CMSMixin
```

Section: Compress

Parameters

4.4 CompressMixin

django-compressor configuration.

```
from kaio.mixins import CompressMixin
```

Section: Compress

Parameters

COMPRESS_DEBUG_TOGGLE by default `nocompress` in `DEBUG` mode.

COMPRESS_ENABLED by default `False`.

COMPRESS_CSS_HASHING_METHOD by default `content`.

COMPRESS_LESSC_ENABLED by default `True`.

COMPRESS_SASS_ENABLED by default `True`.

COMPRESS_BABEL_ENABLED by default `False`.

COMPRESS_LESSC_PATH by default `lessc`.

COMPRESS_SASS_PATH by default `node-sass`.

COMPRESS_BABEL_PATH by default `babel`.

COMPRESS_PRECOMPILERS by default includes automatically `less`, `babel` and `coffeescript` if they are active.

COMPRESS_OUTPUT_DIR by default `CACHE/`.

COMPRESS_OFFLINE by default `False`.

COMPRESS_OFFLINE_TIMEOUT by default `31536000` (1 year in seconds).

COMPRESS_OFFLINE_MANIFEST by default `manifest.json`.

Static offline compression

In order to be able to use it you have to follow two steps:

- add `COMPRESS_OFFLINE = True` to `app.ini` file
- the `{% compress js/css %}` can not have any django logic, no vars, no templatetags, no subblocks...

This last step is advisable to follow it as a good practice just in case in any future moment we want the **COMPRESS_OFFLINE** feature.

Example of the [Compress] section with compress activated and compress offline activated. **LESS**, **SASS** and **BABEL** support are active by default:

```
...
[Compress]
COMPRESS_ENABLED = True
COMPRESS_OFFLINE = True
...
```

The idea is to have `COMPRESS_OFFLINE = False` in development environment and to have `COMPRESS_OFFLINE = True` once we deploy the project to production environment.

In order to test it in development environment you have to execute

```
python manage.py collectstatic
```

and then

```
python manage.py compress
```

4.5 DatabaseMixin

Database access configuration.

```
from kaio.mixins import DatabasesMixin
```

Section: Database

Parameters

DATABASE_ENGINE by default `sqlite3`, allow `sqlite3`, `postgresql_psycopg2`, `mysql`, `oracle`

DATABASE_NAME default name, if we use `sqlite3` it will be `db.sqlite`

DATABASE_USER user to use

DATABASE_PASSWORD password

DATABASE_HOST host name

DATABASE_PORT port number

DATABASE_CONN_MAX_AGE by default 0.

DATABASE_OPTIONS_OPTIONS string to add to database options setting. Empty by default. Example to change the postgresql schema: `DATABASE_OPTIONS_OPTIONS = -c search_path=some_schema`

4.6 DebugMixin

This mixin allows us to define and work with the debug parameters and configure `django-debug-toolbar` to be used in our application. Therefore its use depends on whether this module is configured in the `requirements.txt` of the project, otherwise we will not have activated the option of the debug toolbar.

```
from kaio.mixins import DebugMixin
```

Section: Debug

Parameters

DEBUG by default `False`.

TEMPLATE_DEBUG by default same as **DEBUG**.

ENABLE_DEBUG_TOOLBAR by default same as **DEBUG**. `False` if the module is not installed.

INTERNAL_IPS Debug Toolbar is shown only if your IP is listed in the `INTERNAL_IPS` setting. CSV of IPs, by default `127.0.0.1`.

4.7 EmailMixin

Set the basic parameters by default to configure the mail. In its configuration by default allows us to operate with `django-yubin`, leaving its final configuration for the production environment.

```
from kaio.mixins import EmailMixin
```

Section: Email

Parameters

DEFAULT_FROM_EMAIL by default Example <info@example.com>.

EMAIL_BACKEND by default `django.core.mail.backends.smtp.EmailBackend` or `django_yubin.smtp_queue.EmailBackend` if `django_yubin` is installed.

EMAIL_FILE_PATH by default None.

EMAIL_HOST by default localhost.

EMAIL_HOST_PASSWORD by default ''.

EMAIL_HOST_USER by default ''.

EMAIL_PORT by default 25.

EMAIL_SUBJECT_PREFIX Prefix to add to Django's subject. By default *[Django]*

EMAIL_USE_TLS by default False.

MAILER_PAUSE_SEND by default False.

MAILER_USE_BACKEND by default `django.core.mail.backends.smtp.EmailBackend`.

MAILER_MAIL_ADMINS_PRIORITY by default None.

MAILER_MAIL MANAGERS_PRIORITY by default None.

MAILER_EMPTY_QUEUE_SLEEP by default 30.

MAILER_LOCK_WAIT_TIMEOUT by default 0.

MAILER_LOCK_PATH by default `os.path.join(self.APP_ROOT, "send_mail")`.

Recall that in order to use `django_yubin` we must configure the **cron**.

4.8 FilerMixin

Todo: FilerMixin - Complete description

```
from kaio.mixins import FilerMixin
```

Section: Filer

Parameters

FILER_IS_PUBLIC_DEFAULT Default True.

FILER_ENABLE_PERMISSIONS Default False.

FILER_DEBUG Default False.

FILER_ENABLE_LOGGING Default `False`.

FILER_0_8_COMPATIBILITY_MODE Default `False`.

THUMBNAIL_DEBUG Default `False`.

THUMBNAIL_QUALITY Default `85`.

FILER_CUSTOM_NGINX_SERVER Default `False`.

DEFAULT_FILE_STORAGE Default `django.core.files.storage.FileSystemStorage`.

FILER_CUSTOM_SECURE_MEDIA_ROOT Default `filer_private`.

4.9 LogsMixin

Mixin that handles the configuration the Django logs. Established some default configurations that we use in our development and production environments for the project configuration.

```
from kaio.mixins import LogsMixin
```

Section: Logs

Parameters

LOG_LEVEL sets the project logging level. By default: `DEBUG`

DJANGO_LOG_LEVEL sets the django logging level. By default: `ERROR`

LOG_FILE name of the log file. No established by default, usually specified in production.

EXTRA_LOGGING parameter that sets the log level at module level in a easy way. It does not have a default value. As a parameter we have to set a module list with the differents levels to log each separated by comma in the followinf format: `<module>:log_value` E.g.:

```
[Logs]
EXTRA_LOGGING = oscar.paypal:DEBUG, django.db:INFO
```

LOG_FORMATTER_FORMAT by default `[%(asctime)s] %(levelname)s %(name)s-%(lineno)s %(message)s`. This option is not interpolated, see <https://docs.python.org/3/library/configparser.html#interpolation-of-values>

LOG_FORMATTER_CLASS custom formatter class. By default no formatter class is used.

LOG_FORMATTER_EXTRA_FIELDS optional extra fields passed to the logger formatter class.

SENTRY_ENABLED by default `False`.

SENTRY_DSN by default `' '`.

SENTRY_TRANSPORT by default `'raven.transport.threaded.ThreadedHTTPTransport'`.

4.10 PathsMixin

Paths base settings.

```
from kaio.mixins import PathsMixin
```

Section: Paths

Parameters

APP_ROOT By default the current directory, `abspath('')`.

MEDIA_ROOT By default the current `APP_ROOT + /media`.

STATIC_URL By default `/static/`.

MEDIA_URL By default `/media/`.

STATIC_ROOT By default `abspath(join("/tmp", "{}-static".format(self.APP_SLUG)))`.

4.11 SecurityMixin

Security base settings.

```
from kaio.mixins import SecurityMixin
```

Section: Security

Parameters

SECRET_KEY A secret key for a particular Django installation. This is used to provide cryptographic signing, and should be set to a unique, unpredictable value. By default `' '`.

ALLOWED_HOSTS A list of strings representing the host/domain names that this Django site can serve. By default `[]`.

SECURE_PROXY_SSL_HEADER_NAME user to use The name of the header to configure the proxy ssl. By default `HTTP_X_FORWARDED_PROTO`

SECURE_PROXY_SSL_HEADER_VALUE The value of the header to configure the proxy ssl. By default `https`

SECURE_PROXY_SSL_HEADER A tuple representing a HTTP header/value combination that signifies a request is secure. This controls the behavior of the request object's `is_secure()` method. By default returns the tuple of the combination of the `SECURE_PROXY_SSL_HEADER_NAME` and `SECURE_PROXY_SSL_HEADER_VALUE`. <https://docs.djangoproject.com/en/1.10/ref/settings/#secure-proxy-ssl-header>

4.12 StorageMixin

Mixin that provides settings for `django-storages`. Currently only supports AWS S3. Look at <http://django-storages.readthedocs.io/en/latest/backends/amazon-S3.html> for details.

```
from kaio.mixins import StorageMixin
```

Section: Storage

Parameters

DEFAULT_FILE_STORAGE By default: `storages.backends.s3boto3.S3Boto3Storage`. For tests it might be convenient to change it by `django.core.files.storage.FileSystemStorage`.

AWS_S3_SIGNATURE_VERSION By default `s3v4`.

AWS_S3_REGION_NAME By default `None`. Example: `eu-west-1`.

AWS_STORAGE_BUCKET_NAME By default `' '`.

AWS_LOCATION By default `' '`.

AWS_ACCESS_KEY_ID By default `' '`.

AWS_SECRET_ACCESS_KEY By default ''.

AWS_QUERYSTRING_AUTH By default True.

AWS_DEFAULT_ACL By default private.

4.13 WhiteNoiseMixin

Automatic configuration for static serving using `whitenoise`. You must have version 3 installed.

```
from kaio.mixins import WhiteNoiseMixin
```

Parameters

ENABLE_WHITENOISE by default False. False if the module is not installed.

WHITENOISE_AUTOREFRESH by default True.

WHITENOISE_USE_FINDERS by default True.

5.1 Example from scratch. The kiosk

1. We execute

```
django-admin.py startproject kiosk
```

Since we do not want the project and the application to be called the same we will rename the main directory of *kiosk* to 'prj_kiosk' and we move all within the `src` directory of the project. We will change the name of the `src` folder to `main` so that 'kiosko' will be free if we want to create there the data model.

2. We create the requirements file in the project directory and create the requirements to proceed to create the virtual environment.

```
# requirements.txt
Django==1.10.7
django-appconf==1.0.2
django-compressor==2.1
django-extensions==1.7.2
django-kaio==0.7.1
django-logentry-admin==1.0.2
django-redis==4.4.4
django-robots==2.0
django-storages==1.5.2
django-yubin==0.3.1
psycopg2==2.6.2
pytz==2016.6.1
redis==2.10.5
requests==2.17.3
```

with the versions we need

3. Modify `manage.py` and `wsgi.py` as explained in the *Modifying wsgi.py and manage.py* section.
4. Replace the `settings.py` by our custom version of it. E.g.:

```

import os
from os.path import join

from configurations import Configuration
from django.contrib.messages import constants as messages
from kaio import Options
from kaio.mixins import (CachesMixin, DatabasesMixin, CompressMixin, LogsMixin,
                        PathsMixin, SecurityMixin, DebugMixin, WhiteNoiseMixin)

opts = Options()

class Base(CachesMixin, DatabasesMixin, CompressMixin, PathsMixin, LogsMixin,
          SecurityMixin, DebugMixin, WhiteNoiseMixin, Configuration):
    """
    Project settings for development and production.
    """

    DEBUG = opts.get('DEBUG', True)

    THUMBNAIL_FORCE_OVERWRITE = True

    BASE_DIR = opts.get('APP_ROOT', None)
    APP_SLUG = opts.get('APP_SLUG', 'kiosk')
    SITE_ID = 1
    SECRET_KEY = opts.get('SECRET_KEY', 'key')

    USE_I18N = True
    USE_L10N = True
    USE_TZ = True
    LANGUAGE_CODE = 'es'
    TIME_ZONE = 'Europe/Madrid'

    ROOT_URLCONF = 'main.urls'
    WSGI_APPLICATION = 'main.wsgi.application'

    INSTALLED_APPS = [
        # django
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.sites',
        'django.contrib.messages',
        'django.contrib.staticfiles',

        # apps
        'kiosk',
        'main',

        # 3rd parties
        'compressor',
        'constance',
        'cookielaaw',
        'constance.backends.database',
        'django_extensions',
        'django_yubin',
        'kaio',
    ]

```

```

        'logentry_admin',
        'robots',
        'sorl.thumbnail',
        'bootstrap3',
        'storages',
        'django_tables2',
    ]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

# SecurityMiddleware options
SECURE_BROWSER_XSS_FILTER = True

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'sfc_test_portal/templates/'),
        ],
        'OPTIONS': {
            'context_processors': [
                'django.contrib.auth.context_processors.auth',
                'django.template.context_processors.debug',
                'django.template.context_processors.i18n',
                'django.template.context_processors.media',
                'django.template.context_processors.static',
                'django.contrib.messages.context_processors.messages',
                'django.template.context_processors.tz',
                'django.template.context_processors.request',
                'constance.context_processors.config',
            ],
            'loaders': [
                'django.template.loaders.filesystem.Loader',
                'django.template.loaders.app_directories.Loader',
            ]
        },
    },
]

if not DEBUG:
    TEMPLATES[0]['OPTIONS']['loaders'] = [
        ('django.template.loaders.cached.Loader', TEMPLATES[0]['OPTIONS']['loaders
↪']),
    ]

# Email
EMAIL_BACKEND = 'django_yubin.smtp_queue.EmailBackend'
DEFAULT_FROM_EMAIL = opts.get('DEFAULT_FROM_EMAIL', 'Example <info@example.com>')
MAILER_LOCK_PATH = join(BASE_DIR, 'send_mail')
```

```
# Bootstrap 3 alerts integration with Django messages
MESSAGE_TAGS = {
    messages.ERROR: 'danger',
}

# Constance
CONSTANCE_BACKEND = 'constance.backends.database.DatabaseBackend'
CONSTANCE_DATABASE_CACHE_BACKEND = 'default'
CONSTANCE_CONFIG = {
    'GOOGLE_ANALYTICS_TRACKING_CODE': ('UA-XXXXX-Y', 'Google Analytics tracking_
↪code.'),
}
```

5. Generate the .ini file in the src directory executing:

```
python manage.py generate_ini > app.ini
```

and then modify the default parameters we have. In particular we will have to modify the database connection and put the application in debug mode.

6. Execute the migrations:

```
python manage.py syndb --all
```

And we proceed as always.

7. We need to modify main/urls.py to be able to serve the static content while we are in debug mode.

```
from django.conf.urls import patterns, include, url
from django.conf import settings

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    url(r'^$', 'kiosk.views.home', name='home'),
    url(r'^kiosk/', include('kiosk.foo.urls')),
    url(r'^admin/', include(admin.site.urls)),
)

if settings.DEBUG:
    from django.conf.urls.static import static
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

And finally we run

```
python manage.py apsettings
```

to check the **settings** of our application.

If we need to add an application settings we have two options:

1. Generate a mixin for the particular module, if it has to be reusable.
2. Add such configuration in our settings.py base class.

CHAPTER 6

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

Todo: FilerMixin - Complete description

(The original entry is located in `/home/docs/checkouts/readthedocs.org/user_builds/django-kaio/checkouts/latest/docs/mixins.rst`, line 385.)