Django App Helper Documentation

Release 3.3.4

Iacopo Spalletti

Jan 28, 2024

CONTENTS

Warning: Starting from 3.3 django-app-helper only supports Django 3.2+ and django CMS 3.11. If you need support for older (unsupported) versions, use django-app-helper<3.3

Starting from 3 django-app-helper only supports Django 2.2+ and django CMS 3.7+. If you need support for older (unsupported) versions, use django-app-helper 2.

CHAPTER

ONE

HELPER FOR DJANGO APPLICATIONS DEVELOPMENT

django-app-helper is a set of commands and helper methods to make developing and testing reusable Django applications easier.

Being born in the django CMS ecosystem, it provides a lot of utility functions to develop, run and test django CMS applications.

It's a modified version of django CMS's own `develop.py` script, modified to handle generic application development process.

It supports both tests writted using Django `TestCase` and pytest ones (see pytest support).

1.1 Supported versions

Python: 3.8, 3.9, 3.10, 3.11, 3.12

Django: 3.2, 4.0, 4.1, 4.2, 5.0

django CMS: 3.11

Newer versions might work but are not tested yet.

1.2 Common options

- `--cms`: Loads configuration to properly run a django CMS-based application;
- `--extra-settings`: Path to a helper file to set extra settings; see Project settings with Django App Helper for details;

CHAPTER

TWO

USAGE

The command **must** be executed in the main plugin directory (i.e. in the same directory as the `setup.py` file) as it needs to import files relative to the current directory.

The basic command structure is:

```
django-app-helper <application> <command> [options ...]
```

where **<application>** is the django application name and **<command>** is one of the available commands. Options vary for each command.

2.1 Base test class

A base test class is available to provide helpers and methods that implements repetitive tasks during development or compatibility shims (especially for django CMS).

CHAPTER

THREE

BOOTSTRAPPING

To bootstrap a project using `django-app-helper` you may want to have a look at cookiecutter-djangopackage-helper, a cookiecutter template for `django-app-helper`.

To use it follows usage

CHAPTER

FOUR

RUNNER

By using the integrated runned in the settings file you'll be able to run the commands without invoking `django-app-helper`: see Integrated runner for reference.

CHAPTER

FIVE

ASGI / CHANNELS

ASGI / Channels are supported by installing the project with `django-app-helper[async]`.

With Daphne / Channels installed you can run `django-app-helper server --use-daphne|--use-channels` to run the project on ASGI.

See ASGI / Channels support

Pure ASGI support is available only for Django 3.0+.

CHAPTER

SIX

INSTALLATION

Installing from pip:

```
pip install django-app-helper
```

Installing from source:

```
pip install git+https://github.com/nephila/django-app-helper#egg=django-app-helper
```

6.1 Requirements

- django CMS optional; required only to work with `--cms` option
- docopt
- tox
- dj-database-url

CHAPTER

SEVEN

DOCUMENTATION

Documentation is available on readthedocs.

CHAPTER

EIGHT

AUTHORS

`django-app-helper` was written by Iacopo Spalletti with help from other contributors.

8.1 Thanks

The general logic and part of the code of the whole application is heavily taken from django CMS's own `develop.py` so all the contributors deserve a huge thanks for their work.

8.1.1 How to use Django App Helper

We'll assume that you have an application for django CMS that you're working on.

Once you have django CMS installed, it'll be available using `django-app-helper` command.

`cd` into the root directory of your application (that is, the outer directory containing its `setup.py`). You need to be here to run the `django-app-helper` command.

Running Django App Helper command

Try it:

```
django-app-helper <myapp> test --cms  # change <myapp> to your application's actual name
```

It'll spawn its virtual project and run your tests in it. You should see some output along these lines (there may well be some other output before it gets to this stage):

```
Creating test database for alias 'default'...
F
======================================================================
FAIL: test_bad_maths (djangocms_maths.tests.SmokeTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "./djangocms_maths/tests.py", line 6, in test_bad_maths
    self.assertEqual(1 + 1, 3)
AssertionError: 2 != 3

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

All commands take a form similar to the one you've just run, sharing the basic command structure:

```
django-app-helper <application> <command> [options ...]
```

where **<application>** is the Django application name and **<command>** is one of the available commands. Options vary for each command.

But I haven't written any tests yet!

It helps if you actually have some tests of course - if you don't, simply create a `tests.py` file in your application (not in this directory, but in the package directory, alongside its models and views and so on):

```
from django.test import TestCase

class SmokeTest(TestCase):

    # a deliberately-failing test
    def test_bad_maths(self):
        self.assertEqual(1 + 1, 3)
```

The –cms option

You'll need the `--cms` option most of the time. It sets up the virtual project appropriately for django CMS, providing the required configuration (see *–cms option* for details).

Other commands

Try a couple of the other commands; they're mostly self-explanatory:

```
django-app-helper <myapp> shell --cms  # start a Django shell for the virtual project

django-app-helper <myapp> check --cms  # runs the Django check command

django-app-helper <myapp> cms_check  # runs the django CMS check command
```

Note that the last of these doesn't require the `--cms` option, because of course that is implied anyway by `cms_check`.

Integrated runner

In some contexts running commands by using the complete syntax can be clunky or unfeasible.

Django App Helper contains function that allows to run the commands with a much shorter syntax:

```
python helper.py
```

to run tests

Or:

```
python helper.py server
```

to invoke a server.

See *Integrated runner* for details.

Sphinx integration

When documenting a project using Sphinx autodoc, you mostly need a proper project setup, because the imports in your application's modules will trigger Django setup code anyway.

Using the *Naked setup* it's easy to let helper setup an environment for you:

- setup the *Naked setup*
- add the following code to sphinx `conf.py`:

```
sys.path.insert(0, os.path.abspath('..'))
import app_helper
app_helper.setup()
```

8.1.2 Django App Helper reference

Commands

Commands take the general form:

```
django-app-helper <application> <command> [options ...]
```

where **<application>** is the Django application name and **<command>** is a Django supported command, *or* one of the django-app-helper commands detailed below. Options vary for each command.

Note: while all examples here use the `django-app-helper` CLI, a more idiomatic way to run commands is by using *Integrated runner*.

Common options

- `--extra-settings=path`: loads the extra settings from the provided file instead of the default `helper.py`
- `cms`: loads django CMS specific options (see *–cms option* for details)

Django commands

Django App Helper supports any Django command available according to the project setup; the general syntax is:

```
django-app-helper <application> <command> [options] [--extra-settings=</path/to/settings.
↪py>] [--cms]
```

Example: `django-app-helper some_application shell --cms`

Arguments

- <command> is any available Django command
- [options] is any option/argument accepted by the above command

test

```
django-app-helper <application> test [--failfast] [--migrate] [<test-label>...] [--xvfb]␣
↪[--runner=<test.runner.class>] [--extra-settings=</path/to/settings.py>] [--cms] [--
↪simple-runner] [--runner-options=<option1>,<option2>]
```

Example: `django-app-helper some_application test --cms`

Runs the application's test suite in Django App Helper's virtual environment.

Arguments

- <test-label>: a space-separated list of tests to run; test labels depends on the runner test suite building protocol, please, refer to the runner documentation to know the test label format;

Options

- `--runner`: custom test runner to use in dotted path notation;
- `--runner-options=<option1>,<option2>`: comma separated list of command line options for the test runner: e.g. `--runner-options="--with-coverage,--cover-package=my_package"`
- `--failfast`: whether to stop at first test failure;
- `--migrate`: use migrations (default);
- `--persistent`: use persistent storage for media and static; by default storage is created in `data` directory in the root of the application; if a different directory is needed, use `--persistent-path` to provide the path;
- `--persistent-path`: persistent storage path, instead of `data`
- `--no-migrate`: skip migrations;
- `--xvfb`: whether to configure `xvfb` (for frontend tests);
- `--native` use the native Django command: the use of this option is **incompatible** with the options above.

Test structure

Currently two different tests layouts are supported:

- tests outside the application module:

```
setup.py
tests
    __init__.py
    test_module1.py
    ....
```

- tests inside the application:

```
setup.py
application
    tests
        __init__.py
        test_module1.py
        ...
```

Depending on the used test runner you may need to setup your tests accordingly.

The default runner is the Django one, but it's possible to specify your own custom runner with the `--runner` option.

cms_check

```
django-app-helper <application> cms_check [--extra-settings=</path/to/settings.py>] [--
↪migrate]
```

Runs the django CMS `cms check` command.

Example: `django-app-helper some_application cms_check`

update and compile locales

```
django-app-helper <application> makemessages [--extra-settings=</path/to/settings.py>] [-
↪-cms] [--locale=locale]
django-app-helper <application> compilemessages [--extra-settings=</path/to/settings.py>
↪] [--cms]
```

Examples:

```
django-app-helper some_application makemessages --cms
django-app-helper some_application compilemessages --cms
```

These two commands compiles and update the locale messages.

Options

- `--locale=locale`: `makemessages` allows a single option to choose the locale to update. If not provided **en** is used.

makemigrations

```
django-app-helper <application> makemigrations [--extra-settings=</path/to/settings.py>]␣
↪[--cms] [--merge] [--dry-run] [--empty] [<extra-applications>...]
```

Updates the application migrations (south migrations or Django migrations according to the current installed Django version). For South, it automatically handles `initial` and `auto` options.

Options

- `--merge`: Enable fixing of migration conflicts
- `--empty`: It generates an empty migration for customisations
- `--dry-run`: Does not create migrations file

Arguments

- `<extra-applications>`: Spaces separated list of applications to migrate

squashmigrations

```
django-app-helper <application> squashmigrations <migration-name>
```

Runs the `squashmigrations` command. It operates on the current application.

Arguments

- `<migration-name>`: Squash migrations until this migration

authors

```
django-app-helper <application> authors [--extra-settings=</path/to/settings.py>] [--cms]
```

Generates an authors list from the git log, in a form suitable for the **AUTHORS** file.

server

```
django-app-helper <application> server [--port=<port>] [--bind=<bind>] [--extra-settings=
↪</path/to/settings.py>] [--cms] [--migrate] [--no-migrate] [--persistent | --
↪persistent-path=<path>] [--verbose=<level>] [--use-daphne] [--use-channels]
```

Starts a runserver instance.

- `--port=<port>`: port to bind the server on;
- `--bind=<bind>`: address to bind the server on;
- `--extra-settings=</path/to/settings.py>`: path to extra settings file;
- `--cms`: enable django CMS settings;
- `--migrate`: run migrations on server start (default);
- `--no-migrate`: do not run migrations on server start;
- `--persistent | --persistent-path=<path>`: persist generated media directory; optionally you can provide a fixed path;
- `--verbose=<level>`: verbosity level;

- `--use-daphne`: use daphne server;
- `--use-channels]`: use channels server;

8.1.3 Project settings with Django App Helper

Extra settings

Django App Helper provide a basic set of settings, you'll probably need to provide your own.

Extra settings can be provided by creating a `helper.py` file in the application root directory and providing the settings as a dictionary named `HELPER_SETTINGS`:

```
HELPER_SETTINGS={
    'INSTALLED_APPS': [
        'any_django_app',
    ],
    'ANY_SETTING: False,
    ...
}
```

An alternative, and possibly clearer form is:

```
HELPER_SETTINGS=dict(
    INSTALLED_APPS=[
        'any_django_app',
    ],
    ANY_SETTING=False,
    ...
)
```

By default any setting option provided in `helper.py` will override the default ones.

> **Warning:** On Django 4.2 and up you **cannot** use `DEFAULT_FILE_STORAGE` and `STATICFILES_STORAGE` in `HELPER_SETTINGS`: use new `STORAGES` setting instead.

Special settings

The following settings will not override the defaults ones, but they are appended to the defaults to make easier to customise the configuration:

- `INSTALLED_APPS`
- `TEMPLATE_CONTEXT_PROCESSORS`
- `TEMPLATE_LOADERS`
- `TEMPLATE_DIRS`
- `MIDDLEWARE_CLASSES`

Other extra setting:

- `TOP_INSTALLED_APPS`: items in this setting will be inserted on top of `INSTALLED_APPS` (e.g.: to control the templates and static files override from standard applications configured by django-app-helper).

- TOP_MIDDLEWARE_CLASSES: items in this setting will be inserted on top of MIDDLEWARE_CLASSES.

Django 1.8 support

All TEMPLATES_ settings from Django 1.6/1.7 are automatically translated to Django 1.8 TEMPLATE setting. To support both, just use the **old** names, and `django-app-helper` will take care of converting.

default settings

These are the applications, context processors and middlewares loaded by default

Applications:

```
'django.contrib.contenttypes',
'django.contrib.auth',
'django.contrib.sessions',
'django.contrib.sites',
'django.contrib.staticfiles',
'django.contrib.admin',
'app_helper.test_data',  # this provides basic templates and urlconf
'django.contrib.messages',
```

Template context processors:

```
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
'django.core.context_processors.i18n',
'django.core.context_processors.csrf',
'django.core.context_processors.debug',
'django.core.context_processors.tz',
'django.core.context_processors.request',
'django.core.context_processors.media',
'django.core.context_processors.static',
```

Note: On Django 1.8 these are translated to the new path `django.template.context_processors.*`

Middlewares:

```
'django.middleware.http.ConditionalGetMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
'django.middleware.locale.LocaleMiddleware',
'django.middleware.common.CommonMiddleware',
```

–cms option

When using `--cms` option, `INSTALLED_APPS`, `TEMPLATE_CONTEXT_PROCESSORS` and `MIDDLEWARE_CLASSES` related to django CMS are added to the default settings so you won't need to provide it yourself.

Applications:

```
'djangocms_admin_style',
'mptt',
'cms',
'menus',
'sekizai',
```

When django CMS 3.1+ is used, `treebeard` is configured instead of `mptt`.

Template context processors:

```
'cms.context_processors.cms_settings',
'sekizai.context_processors.sekizai',
```

Middlewares:

```
'cms.middleware.language.LanguageCookieMiddleware',
'cms.middleware.user.CurrentUserMiddleware',
'cms.middleware.page.CurrentPageMiddleware',
'cms.middleware.toolbar.ToolbarMiddleware',
```

`django-app-helper` discovers automtically the South / Django migrations layout and configure the settings accordingly. As of the current version `filer`, `djangocms_text_ckeditor`, `cmplugin_filer` are supported.

8.1.4 Integrated runner

Django App Helper provide a runner to invoke the commands without requiring the `django-app-helper` file; this can be useful to invoke tests with coverage or to have a simpler syntax to remember.

Typically you'd setup the runner function in the *extra settings file*:

```
HELPER_SETTINGS={
    'INSTALLED_APPS': [
        'any_django_app',
    ],
    'ANY_SETTING: False,
    ...
}


def run():
    from app_helper import runner
    runner.cms('my_app')

if __name__ == "__main__":
    run()
```

with the above code in place you can run any Django App Helper command as:

```
python helper.py <command>
```

and adding the `test_suite` argument to `setup.py`:

```
setup(
    ...
    test_suite='app_helper.run',
    ...
)
```

you can invoke the tests with:

```
python setup.py test
```

Django environment

If you don't need django CMS, you can use a runner function with no CMS attached:

```
def run():
    from app_helper import runner
    runner.run('my_app')

if __name__ == "__main__":
    run()
```

> **Warning:** The runner **must** be invoked from the **settings** file. The runner takes care of setting up the file in which is invoked as the `extra_settings` file.

Naked setup

Sometimes you just want to properly setup a Django environment without running any commands (e.g: when building Sphinx docs using autodoc). Naked setup allows to do so:

```
def setup():
    import sys
    from app_helper import runner
    runner.setup('my_app', sys.modules[__name__], use_cms=True)

if __name__ == "cms_helper":
    setup()
```

the last lines allows to auto-load naked setup when runner file is imported. This is useful when running tests in a PyCharm environment. In case you customized the runner filename, replace `"cms_helper"` with the custom name.

> **Warning:** The runner **must** be invoked from the **settings** file. The runner takes care of setting up the file in which is invoked as the `extra_settings` file.

_migrating:

8.1.5 Migrating from djangocms-helper to django-app-helper

This project used to be called djangocms-helper. It's been renamed in version 2.0 to clarify that it's not limited to django CMS apps.

Migration is straightforward as it does not require any change to the codebase:

- all imports from `djangocms_helper` namespace are still valid and they won't be deprecated soon
- runner filename `cms_helper.py` is still valid and it won't be deprecated soon

Migration path

- Replace `djangocms-helper` package name from any dependendency declaration (`setup.py`, `tox.ini`, `requirements.txt` ...)

That's it!

Bugfixes and further development

Bugfixes to djangocms-helper 1.2.x will be released until reasonable under the old package name, while new features (including new Django / django CMS versions support will only be available in the django-app-helper package).

8.1.6 Base test mixins

The following mixins are available to provide helpers and methods that implements helpers and functions commonly used in tests. `BaseTestCase`, `BaseTransactionTestCase` are concrete classes implementing all the mixins and extending respectively `django.tests.TestCase` and `django.tests.TransactionTestCase`

class app_helper.base_test.**RequestTestCaseMixin**

Provide methods to get complex request objects.

Resulting `request` has more realistic attributes (i.e.: all the attributes found in a non-test request) than the plain `django.test.RequestFactory`.

login_user_context(*user*, *password=None*)

Context manager to make logged in requests.

Usage:

```
with self.login_user_context("<username>", password="<password>"):
    request = self.request("/", lang="en")
    ... # this request has <username> as user
```

Parameters

- **user** – user username
- **password** – user password (if omitted, username is used)

request(*path*, *method='get'*, *data=None*, *page=None*, *lang=''*, *user=None*, *use_middlewares=False*, *secure=False*, *use_toolbar=False*)

Create a request for the given parameters.

Request will be enriched with:

- session

- cookies
- user (Anonymous if :param:user is *None*)
- django CMS toolbar (is set)
- current_page (if provided)

Parameters

- **path** (*str*) – request path
- **method** (*str*) – HTTP verb to use
- **data** (*dict*) – payload to pass to the underlying `django.test.RequestFactory` method
- **page** (*cms.models.Page*) – current page object
- **lang** (*str*) – request language
- **user** (`django.contrib.auth.models.AbstractUser`) – current user
- **use_middlewares** (*bool*) – pass the request through configured middlewares
- **secure** (*bool*) – create HTTPS request
- **use_toolbar** – add django CMS toolbar

Returns
request

class app_helper.base_test.**CreateTestDataMixin**

Provide methods to automatically create users on test setup and shortcut to generate test data.

classmethod _setup_users()

Create standard users.

- `user`: superuser
- `user_staff`: staff user
- `user_normal`: plain django user

classmethod _teardown_users()

Delete existing users.

_admin_user_username = 'admin'

Username for auto-generated superuser

_admin_user_password = 'admin'

Password for auto-generated superuser

_admin_user_email = 'admin@admin.com'

Email for auto-generated superuser

_staff_user_username = 'staff'

Username for auto-generated staff user

_staff_user_password = 'staff'

Password for auto-generated staff user

_staff_user_email = 'staff@admin.com'

Email for auto-generated staff user

`_user_user_username = 'normal'`

Username for auto-generated non-staff user

`_user_user_password = 'normal'`

Password for auto-generated non-staff user

`_user_user_email = 'user@admin.com'`

Email for auto-generated non-staff user

`classmethod create_django_image()`

Create a django image file object suitable for FileField It also sets the following attributes:

- `self.image_name`: the image base name
- `self.filename`: the complete image path

Returns
(django file object, path to file image)

It requires Pillow installed in the environment to work

`create_django_image_object()`

Create a django image file object suitable for FileField It also sets the following attributes:

- `self.image_name`: the image base name
- `self.filename`: the complete image path

Returns
django file object

It requires Pillow installed in the environment to work

`classmethod create_filer_image(user, image_name)`

Create a filer image object suitable for FilerImageField It also sets the following attributes:

- `self.image_name`: the image base name
- `self.filename`: the complete image path
- `self.filer_image`: the filer image object

Parameters

- **user** – image owner
- **image_name** – image name

Returns
filer image object

It requires Pillow and django-filer installed in the environment to work

`create_filer_image_object()`

Create a filer image object suitable for FilerImageField It also sets the following attributes:

- `self.image_name`: the image base name
- `self.filename`: the complete image path
- `self.filer_image`: the filer image object

Returns
filer image object

It requires Pillow and django-filer installed in the environment to work

`static create_image(mode='RGB', size=(800, 600))`

Create a random image suitable for saving as DjangoFile :param mode: color mode :param size: tuple of width, height :return: image object

It requires Pillow installed in the environment to work

`create_user(username, email, password, is_staff=False, is_superuser=False, base_cms_permissions=False, permissions=None)`

Creates a user with the given properties

Parameters

- **username** – Username
- **email** – Email
- **password** – password
- **is_staff** – Staff status
- **is_superuser** – Superuser status
- **base_cms_permissions** – Base django CMS permissions
- **permissions** – Other permissions

Returns
User instance

`class app_helper.base_test.CMSPageRenderingMixin`

Provide hooks to create sample pages in tests and helper methods to render pages and plugins.

`classmethod _setup_cms()`

Setup data required by django CMS as class attributes.

- `site_1`: instance of the first `Site`
- `languages`: list of configured languages

`_pages_data = ()`

List of pages data for the different languages.

Each item of the list is a dictionary containing the attributes (as accepted by `cms.api.create_page()`) of the page to be created.

The first language will be created with `cms.api.create_page()` the following languages using `cms.api.create_title()`.

Example:
Single page created in en, fr, it languages:

```
_pages_data = (
    {
        'en': {'title': 'Page title', 'template': 'page.html', 'publish':␣
↪True},
        'fr': {'title': 'Titre', 'publish': True},
        'it': {'title': 'Titolo pagina', 'publish': False}
```

(continues on next page)

(continued from previous page)

```
    },
)
```

static create_pages(*source*, *languages*)

Build pages according to the pages data provided by *get_pages_data()* and returns the list of the draft version of each

get_content_renderer(*request*)

Returns a the plugin renderer. Only for django CMS 3.4+

Parameters
request – request instance

Returns
ContentRenderer instance

get_page_request(*page*, *user*, *path=None*, *edit=False*, *lang='en'*, *use_middlewares=False*, *secure=False*)

Deprecated, use *get_toolbar_request()*.

get_pages()

Create pages using self._pages_data and self.languages

Returns
list of created pages

get_pages_data()

Construct a list of pages in the different languages available for the project. Default implementation is to return the *_pages_data* attribute

Returns
list of pages data

get_plugin_context(*page*, *lang*, *plugin*, *edit=False*)

Returns a context suitable for CMSPlugin.render_plugin / render_placeholder

Parameters

- **page** – Page object
- **lang** – Current language
- **plugin** – Plugin instance
- **edit** – Enable edit mode for rendering

Returns
PluginContext instance

get_request(*page*, *lang*, *user=None*, *path=None*, *use_middlewares=False*, *secure=False*, *use_toolbar=False*)

Create a GET request for the given page and language.

Parameters

- **page** – current page object
- **lang** – request language
- **user** – current user
- **path** – path (if different from the current page path)

- **use_middlewares** – pass the request through configured middlewares.
- **secure** – create HTTPS request
- **use_toolbar** – add django CMS toolbar

Returns
request

get_toolbar_request(*page*, *user*, *path=None*, *edit=False*, *lang='en'*, *use_middlewares=False*, *secure=False*)

Create a GET request for the given page suitable for use the django CMS toolbar.

This method requires django CMS installed to work. It will raise an ImportError otherwise; not a big deal as this method makes sense only in a django CMS environment.

Parameters

- **page** – current page object
- **user** – current user
- **path** – path (if different from the current page path)
- **edit** – whether enabling editing mode
- **lang** – request language
- **use_middlewares** – pass the request through configured middlewares.
- **secure** – create HTTPS request

Returns
request

post_request(*page*, *lang*, *data*, *user=None*, *path=None*, *use_middlewares=False*, *secure=False*, *use_toolbar=False*)

Create a POST request for the given page and language with CSRF disabled.

Parameters

- **page** – current page object
- **lang** – request language
- **data** – POST payload
- **user** – current user
- **path** – path (if different from the current page path)
- **use_middlewares** – pass the request through configured middlewares.
- **secure** – create HTTPS request
- **use_toolbar** – add django CMS toolbar

Returns
request

render_plugin(*page*, *lang*, *plugin*, *edit=False*)

Renders a single plugin using CMSPlugin.render_plugin

Parameters

- **page** – Page object
- **lang** – Current language

- **plugin** – Plugin instance
- **edit** – Enable edit mode for rendering

Returns
Rendered plugin

class app_helper.base_test.**GenericHelpersMixin**

captured_output()

Context manager that patches stdout / stderr with StringIO and return the instances.

Useful to test output.

Returns
stdout, stderr wrappers

reload_model(*obj*)

Reload models instance from database.

Contrary to `refresh_from_db` returns a completely new instance, instead of updating the current one.

Parameters
obj – model instance to reload

Returns
the reloaded model instance

static reload_urlconf(*urlconf=None*)

Reload django urlconf and any attached apphook.

temp_dir()

Return the context manager of a temporary directory that is removed upon exit.

Usage:

```
with self.temp_dir() as temp_path:
    test_file = os.path.join(temp_path, "afile")
    ... # do something with test_file
... # test_file and containing directory is removed
```

class app_helper.base_test.**BaseNoDataTestCaseMixin**

Provide helper methods to setup and interact with Django testing framework.

Does not create in `setUpClass()`, but provides all the methods to tap into automatic data generation.

Implements:

- *CreateTestDataMixin*
- *CMSPageRenderingMixin*
- *GenericHelpersMixin*

class app_helper.base_test.**BaseTestCaseMixin**

Provide helper methods to setup and interact with Django testing framework.

Like *BaseNoDataTestCaseMixin* but create sample data in `setUpClass()` according to *CreateTestDataMixin* and *CMSPageRenderingMixin* configuration

Implements:

- *CreateTestDataMixin*

- *CMSPageRenderingMixin*
- *GenericHelpersMixin*

class app_helper.base_test.**BaseTestCase**(*methodName='runTest'*)

Base class that implements *BaseTestCaseMixin* and `django.tests.TestCase`

class app_helper.base_test.**BaseTransactionTestCase**(*methodName='runTest'*)

Base class that implements *BaseTestCaseMixin* and `django.tests.TransactionTestCase`

8.1.7 ASGI / Channels support

django-app-helper comes with minimal channels / ASGI support.

Default configuration provides a sample `asgi` application already enabled in `ASGI_APPLICATION` setting.

This means that if you install `channels` or `daphne` in your rest environment `./helper.py server` can run a channels / ASGI enabled instance.

Note: Pure ASGI support is available only for Django 3.0+.

Run with channels

To run with channels you must provide an `ASGI_APPLICATION` in the project `helper.py` pointing to your base channels application.

Optionally you can set `CHANNEL_LAYERS`.

Example:

```
HELPER_SETTINGS = dict(
    ...
    # required
    ASGI_APPLICATION='tests.example_app.routing.application',
    # Optional
    CHANNEL_LAYERS={
        'default': {
            'BACKEND': 'channels_redis.core.RedisChannelLayer',
            'CONFIG': {
                'hosts': [('localhost', 6379)],
            },
        },
    },
    ...
)
```

The run the `server` command with the `--use-channels` option set:

```
$ python helper.py server --use-channels
```

Run with daphne

To run with daphne you can provide a custom `ASGI_APPLICATION` in the project `helper.py` if you actually have one or more ASGI application configure beyond django. The default `ASGI_APPLICATION` will run the django runserver command.

Example:

```
HELPER_SETTINGS = dict(
    ...
    ASGI_APPLICATION='my_project.asgi:application',
    ...
)
```

The run the `server` command with the `--use-daphne` option set:

```
$ python helper.py server --use-daphne
```

8.1.8 pytest support

While django-app-helper was born with Django `TestCase` in mind, it can be used with `pytest` with some configuration together with `pytest-django`.

django-app-helper runner

You can run pytest tests by using a custom runner (based on pytest-django documentation); to enable it, add the following to project `helper.py` file:

```
HELPER_SETTINGS = {
    ...
    "TEST_RUNNER": "app_helper.pytest_runner.PytestTestRunner",
    ...
}
```

Using this approach you can mix pytest tests and Django `TestCase` ones, the runner will take care of discovering and running both.

Running tests

Invoke `app_helper` as usual:

```
$ python helper.py <app-name> test
```

pytest options

The runner support translates the following Django test runner options to pytest ones:

- `verbosity == 0`: `--quiet`
- `verbosity == 2`: `--verbose`
- `verbosity == 3`: `-vv`
- `failfast`: `--exitfirst`
- `keepdb`: `--reuse-db`

All the other pytest and pytest plugins are supported either via `PYTEST_ARGS` enviroment variable or `--runner-options` cmdline argument.

Environment variable example:

```
PYTEST_ARGS='-s -k my_test' python helper.py test
```

argument variable example:

```
python helper.py test --runner-options="-k my_test"
```

In case arguments are passed via both channels they are merged together, with runner-options arguments having priority over environment variables in case of overlapping options.

standard pytest

Running tests

Invoke `pytest` as usual:

```
$ python -mpytest <args>
```

or:

```
$ pytest <args>
```

In this case you don't need any special syntax to pass commands as the django-app-helper pytest runner is not executed and pytest is full in control.

Using BaseTestCaseMixin

While its *BaseTestCaseMixin* is built on Django `TestCase`, it can be used in pytest classes:

Fixtures, markers and decorators can be used as usual on test methods as in classic pytest classes.

```
class TestTags(BaseTestCaseMixin):
    ...
    def test_foo(self):
        ...
```

8.1.9 Development & community

Django App Helper is an open-source project.

You don't need to be an expert developer to make a valuable contribution - all you need is a little knowledge, and a willingness to follow the contribution guidelines.

Nephila

Django App Helper was created by Iacopo Spalletti at Nephila and is released under a GNU GENERAL PUBLIC LICENSE.

Nephila is an active supporter of django CMS and its community. Django App Helper is intended to help make it easier for developers in the django CMS ecosystem to work effectively and create high quality applications.

Nephila maintains overall control of the Django App Helper repository.

Standards & policies

Django App Helper is a django CMS application, and shares much of django CMS's standards and policies (when relevant).

These include:

- guidelines and policies for contributing to the project
- a code of conduct for community activity

Please familiarise yourself with this documentation if you'd like to contribute to Django App Helper.

8.1.10 Contributing

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

Please read the instructions here to start contributing to *django-app-helper*.

8.1.11 History

3.3.4 (2024-01-28)

Features

- Add DjangoCMS 3.11 to Django 5.0 test matrix (#252)

Bugfixes

- Fix support for DEFAULT_FILE_STORAGE/STATICFILES_STORAGE in django 4.2 (#255)

3.3.3 (2023-11-28)

Features

- Prepare for Django 5.0 / Python 3.12 compatibility (#244)
- Switch to Coveralls Github action (#248)

3.3.2 (2023-09-26)

Features

- Migrate to bump-my-version (#237)

Bugfixes

- Fix ruff linting (#232)

3.3.1 (2023-07-09)

Bugfixes

- Fix runner-options argument on Django test runner (#220)
- Do not add mptt with django-filer 3+ (#225)

3.3.0 (2023-05-07)

Features

- Add support for Django 4.x (#208)

3.2.0 (2023-01-19)

Features

- Add djangocms 3.11 to tox, fix tests accordingly (#311)

3.1.0 (2022-07-29)

Features

- Upgrade Python / Django versions (#204)
- Add minimal django 4.0 support (#208)

3.0.1 (2020-12-09)

Bugfixes

- Fix loading setting with pytest-django and django 3.1 (#202)

3.0.0 (2020-11-14)

Features

- Add support for Django 3.1 / django CMS 3.8 (#196)
- Add Django 3.0 / django CMS 3.7.2 support (#142)
- Drop Python 2 / Django 1.11 (#148)
- Add support for Daphne / channels runserver (#198)
- Refactor BaseTestCaseMixin to more composable mixins (#107)
- Replace makefile with invoke (#143)
- Use pre-commit for code formatting (#149)
- Allow to pass arguments to pytest via runner-options argument (#159)
- Add support to pytest command (#167)
- Update dotfiles to latest version (#189)
- Reorganize tests outside main package (#191)
- Remove support for aldryn-boilerplates (#199)

Bugfixes

- Fix runner_options support (#92)
- Improve GA - Update contribution guide (#161)
- Allow extra arguments in PytestTestRunner.run_tests (#165)
- Update isort and linting configuration (#188)

Misc

- #152, #185

2.2.2 (2020-05-15)

Bugfixes

- Fix pytest args splitting (#155)
- Fix runserver autoreload with channels 2.4 (#157)

2.2.1 (2020-04-23)

- Fix packaging error

2.2.0 (2020-04-23)

Features

- Add Django 3.0 / django CMS 3.7.2 support (#142)
- Replace makefile with invoke (#143)

2.1.1 (2020-02-04)

- Improved pytest compatibility

2.1.0 (2019-12-27)

- Reformat code with black and improve flake8 configuration
- Add pytest-compatible runner

2.0.1 (2019-12-22)

- Add Django 3.0 preliminary support

2.0.0 (2019-10-13)

- Rename application to django-app-helper

1.2.5 (2019-08-16)

- Add django CMS 3.7
- Add Django 2.2

1.2.4 (2019-08-08)

- Fix regression introduced by #116

1.2.3 (2019-08-05)

- Move pyflakes to extras_require
- Fix error in get_request / post_request not preserving current_page

1.2.2 (2019-07-05)

- Improve request generation by adding a more generic request method

1.2.1 (2019-07-04)

- Fix error when creating users with non-writable email attribute

1.2.0 (2019-03-22)

- Drop compatiblity with Django <1.11, Python 3.4
- Add django CMS 3.6
- Add django 2.0, 2.1

1.1.1 (2019-07-03)

- Fix error when creating users with non-writable email attribute

1.1.0 (2018-02-20)

- Remove Django <1.8, Python 2.6, 3.3 from setup.py
- Add Django 1.11, Python 3.6
- Switch to new-style middlewares for Django 1.10+
- Create static methods to generate images
- Fix persistent option behavior with arbitrary commands
- Add minimal changes to allow third party application to run test on django 2.0
- Fix options for channels runserver
- Remove support for django-nose test runner

1.0.0 (2017-07-25)

- Add ApphookReloadMiddleware in server mode
- Add a default for FILE_UPLOAD_TEMP_DIR
- Add fix for django CMS 3.4.4 render_plugin

0.9.8 (2017-03-04)

- Fix compatibility with newer channels releases

0.9.7 (2016-12-03)

- Add support for django-sekizai 0.10
- Fix mock dependency in setup.py
- Fix issue with server command in Django 1.10
- Fix issue with urls.py in Django 1.10
- Fix issue in tests with django CMS 3.4

0.9.6 (2016-08-25)

- Add support for channels runserver.
- Add verbosity level to server command.
- Add support for Django 1.10.
- Add support for django CMS 3.4.

0.9.5 (2016-06-06)

- Fix issue with mocked session storage
- Add verbosity level to tests
- Fix user creation
- Add option to allow parametrizing auto-created user
- Fix extra_applications

0.9.4 (2016-01-20)

- Add Naked setup mode
- Add TEMPLATE_DIRS to special settings
- Add TEMPLATE_LOADERS to special settings
- Allow to specify a locale in makemessages

0.9.3 (2015-10-07)

- Add –no-migrate option to skip migrations
- Add secure argument to generate HTTPS requests
- Better request mocking
- Fix test on django CMS 3.2 (develop)
- Add support for Python 3.5
- Add –persistent option for persistent storage

0.9.2 (2015-09-14)

- Add support for apphooks and parent pages in BaseTestCase.create_pages
- If pages contains apphook, urlconf is reloaded automatically
- Improve documentation
- Add support for top-positioned MIDDLEWARE_CLASSES
- Code cleanup

0.9.1 (2015-08-30)

- Better support for aldryn-boilerplates

0.9.0 (2015-08-20)

- Complete support for Django 1.8 / django CMS develop
- Support for aldryn-boilerplates settings
- Migrations are now enabled by default during tests
- Minor BaseTestCase refactoring
- Remove support for Django 1.5
- Fix treebeard support
- Minor fixes
- Adds login_user_context method to BaseTestCase

0.8.1 (2015-05-31)

- Add basic support for Django 1.8 / django CMS develop
- Code cleanups
- Smarter migration layout detection

0.8.0 (2015-03-22)

- Add –native option to use native test command instead of django-app-helper one
- Use django-discover-runner on Django 1.5 if present
- Better handling of runner options
- Add support for empty/dry-run arguments to makemigrations
- Add USE_CMS flag to settings when using django CMS configuration

0.7.0 (2015-01-22)

- Fix an error which prevents the runner to discover the settings
- django CMS is no more a dependency, install it manually to enable django CMS support

0.6.0 (2015-01-10)

- Add a runner to make cms_helper file itself a runner for django-app-helper
- Fix issues with mptt / treebeard and Django 1.7
- Fix some makemigrations / –migrate issues
- Make django-app-helper less django CMS dependent

0.5.0 (2015-01-01)

- Fixing bugs when using extra settings
- Add messages framework to default environment
- Add CSRF middleware / context_processor to default settings
- Add base helper class for test cases
- Complete Django 1.7 support
- Smarter detection of migration operations in Django 1.6-
- Add option to create migrations for external applications

0.4.0 (2014-09-18)

- Add support for command line test runner options;
- Add check command on Django 1.7+;
- Add cms check command (which triggers cms inclusion);
- Add squashmigration command Django 1.7+;
- Add support for makemigrations merge on Django 1.7+;
- Add helpers for custom user models;

0.3.1 (2014-08-25)

- Add staticfiles application;
- Add djangocms_admin_style if cms is enabled;

0.3.0 (2014-08-14)

- Add support for django nose test runner;
- Add default CMS template;

0.2.0 (2014-08-12)

- Add option to customize sample project settings;
- Add option to exclude django CMS from test project configurations;
- Add support for Django 1.7;

0.1.0 (2014-08-09)

- First public release.

INDEX

Symbols

B

C

G

L

P

R

T