django-app-manage Documentation

Release 0.1

Jonas Obrist

February 01, 2017

Contents

django-app-manage is a library that helps you write `manage.py` files for your Django libraries (as opposed to projects), to make things like testing and managing your project easier.

CHAPTER 1

Quickstart

1.1 Introduction

Let's assume you have just built an awesome Django library called `myapp` that is structured like this:

```
setup.py
README.rst
LICENSE
myapp/
    __init__.py
    admin.py
    models.py
    tests.py
    views.py
    urls.py
```

To use django-app-manage, we'll add a new file at the root of the library (next to `setup.py`) called `manage.py`, so it now looks like this:

```
manage.py
setup.py
README.rst
LICENSE
myapp/
    __init__.py
    models.py
    urls.py
    views.py
    tests.py
```

Inside that file we add the following:

```
import app_manage

if __name__ == '__main__':
    app_manage.main(
        ['myapp'],
        ROOT_URLCONF='myapp.urls',
    )
```

Now try running it with `python manage.py`, you should see output that is the same as it would be from a `manage.py` in a Django project.

1.2 Configuration

The default configuration used for your app is as follows:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    }
}

CACHE_MIDDLEWARE_ANONYMOUS_ONLY = True

DEBUG = True

DATABASE_SUPPORTS_TRANSACTIONS = True

SITE_ID = 1

USE_I18N = True

MEDIA_URL = '/media/'

STATIC_URL = '/static/'

ADMIN_MEDIA_PREFIX = '/static/admin/'

EMAIL_BACKEND = 'django.core.mail.backends.locmem.EmailBackend'

SECRET_KEY = 'secret-key'

INTERNAL_IPS = ['127.0.0.1']

LANGUAGE_CODE = "en"

LANGUAGES = [
    ('en', 'English')
]

PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.MD5PasswordHasher'
]

ALLOWED_HOSTS = ['localhost']

MIDDLEWARE_CLASSES = [
    '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'
]

INSTALLED_APPS = []
```

The list of apps you pass to *app_manage.core.main()* as its first argument will be appended to `INSTALLED_APPS`.

Besides providing an easy way for you to get a working `manage.py` to use to develop your Django library, you can

also have dynamic configuration.

So let's make the DATABASES setting easily configurable:

```
import app_manage

if __name__ == '__main__':
    app_manage.main(
        ['myapp'],
        ROOT_URLCONF='myapp.urls',
        DATABASES=app_manage.DatabaseConfig(
            env='DATABASE_URL',
            arg='--database-url',
            default='sqlite://localhost/local.sqlite'
        )
    )
```

Now you can use a dj_database_url compatible database URL to switch the database used by django-app-manage. You can either use the `--database-url` argument like this `python manage.py --database-url 'postgres://username:password@host:port/database_name'` or set the environment variable DATABASE_URL like this: `DATABASE_URL='postgres://username:password@host:port/database_name' python manage.py`. By default it will use a sqlite3 database named `local.sqlite`.

For most other dynamic settings you can use the *app_manage.config.Config* class. For example to make the LANGUAGE_CODE setting configurable we modify our `manage.py` like this:

```
import app_manage

if __name__ == '__main__':
    app_manage.main(
        ['myapp'],
        ROOT_URLCONF='myapp.urls',
        DATABASES=app_manage.DatabaseConfig(
            env='DATABASE_URL',
            arg='--database-url'
            default='sqlite://localhost/local.sqlite'
        ),
        LANGUAGE_CODE=app_manage.Config(
            env='LANGUAGE_CODE',
            arg='--language-code',
            default='en',
        )
    )
```

Now you can use `--language-code` or LANGUAGE_CODE to change the language code used from its default value of `'en'`.

For your convenience, one more helper class is provided which will create temporary directories that live for the duration of the Django command execution. This is very handy for the MEDIA_ROOT and STATIC_ROOT settings, especially for running tests. So let's add those two to our `manage.py`:

```
import app_manage

if __name__ == '__main__':
    app_manage.main(
        ['myapp'],
        ROOT_URLCONF='myapp.urls',
        DATABASES=app_manage.DatabaseConfig(
            env='DATABASE_URL',
            arg='--database-url'
```

```
            default='sqlite://localhost/local.sqlite'
        ),
        LANGUAGE_CODE=app_manage.Config(
            env='LANGUAGE_CODE',
            arg='--language-code',
            default='en',
        ),
        STATIC_ROOT=app_manage.TempDir(),
        MEDIA_ROOT=app_manage.TempDir(),
    )
```

Lastly, if the value you want to configure is a boolean you can use *app_manage.config.Flag* to instruct django-app-manage to do so. Let's make `USE_TZ` configurable using a flag:

```
import app_manage

if __name__ == '__main__':
    app_manage.main(
        ['myapp'],
        ROOT_URLCONF='myapp.urls',
        DATABASES=app_manage.DatabaseConfig(
            env='DATABASE_URL',
            arg='--database-url'
            default='sqlite://localhost/local.sqlite'
        ),
        LANGUAGE_CODE=app_manage.Config(
            env='LANGUAGE_CODE',
            arg='--language-code',
            default='en',
        ),
        STATIC_ROOT=app_manage.TempDir(),
        MEDIA_ROOT=app_manage.TempDir(),
        USE_TZ=app_manage.Config(
            env='USE_TZ',
            arg=app_manage.Flag('--use-tz'),
            default=False,
        ),
    )
```

1.3 Complex configuration

If the built-in classes do not cover your needs for dynamic settings, you have two options:

- Implement your own subclass of *app_manage.config.DynamicSetting* to configure a single setting at a time. For details, read its API documentation and read the existing subclasses such as *app_manage.config.Config*.
- Pass an instance of *app_manage.config.Argument* to *app_manage.core.main()* to configure multiple settings at once.

1.3.1 Configuring multiple settings with a single argument

Let's say you want a single argument configure more than one setting at once, for example to use a custom `AUTH_USER_MODEL` that requires an extra app to be added to `INSTALLED_APPS`. To do so, change your `manage.py` to look something like this:

```
import app_manage

def install_auth_user_model(settings, value):
    if value:
        settings['AUTH_USER_MODEL'] = 'myapp2.CustomUser'
        settings['INSTALLED_APPS'].append('myapp2')

if __name__ == '__main__':
    app_manage.main(
        ['myapp'],
        app_manage.Argument(
            config=app_manage.Config(
                env='AUTH_USER_MODEL',
                arg=app_manage.Flag('--auth-user-model'),
                default=False,
            ),
            callback=install_auth_user_model
        ),
        ROOT_URLCONF='myapp.urls',
        DATABASES=app_manage.DatabaseConfig(
            env='DATABASE_URL',
            arg='--database-url'
            default='sqlite://localhost/local.sqlite'
        ),
        LANGUAGE_CODE=app_manage.Config(
            env='LANGUAGE_CODE',
            arg='--language-code',
            default='en',
        ),
        STATIC_ROOT=app_manage.TempDir(),
        MEDIA_ROOT=app_manage.TempDir(),
        USE_TZ=app_manage.Config(
            env='USE_TZ',
            arg=app_manage.Flag('--use-tz'),
            default=False,
        ),
    )
```

CHAPTER 2

API

2.1 app_manage.core

app_manage.core.**main**(*apps*, *, *argv=sys.argv*, *environ=os.environ*, **args*, ***config*)

Main entry point into django-app-manage. Configures Django and then runs the command passed in `argv` (or `sys.argv`).

Parameters

- **apps** (*list*) – List of app names (as strings). These will be auto-added to `INSTALLED_APPS`.
- **argv** (*list*) – List of arguments, you shouldn't need to have to set this value yourself.
- **environ** (*dict*) – Environment dictionary, you shouldn't need to have to set this value yourself.
- **args** – Arguments to configure your app. Must be instances of *app_manage.config.Argument*.
- **config** – Django configuration to use in your app. Values can be instances of *app_manage.config.DynamicSetting* subclasses if you want them to be configurable.

2.2 app_manage.config

exception app_manage.config.**DynamicConfigError**

Raised by *Config* if the argument is used as a flag.

class app_manage.config.**DynamicSetting**

Base class for all your dynamic settings. Must be subclassed.

get_value(*argv*, *environ*)

This method must be implemented by subclasses.

Given the arguments and environment, returns the value to be used for the given setting.

`argv` can and should be modified by this method where applicable.

Parameters

- **argv** (*list*) – List of arguments.
- **environ** (*dict*) – Environment dictionary.

cleanup()
Optional method that can be used to do any cleanup where necessary after the Django command finished executing.

class app_manage.config.**TempDir**
DynamicSetting subclass that returns a path to a temporary directory and removes that directory after the Django command executed.

Useful for MEDIA_ROOT and similar settings.

class app_manage.config.**Config**(*env=None*, *arg=None*, *default=NULL*)
The bread-and-butter class to configure your app. env is the key in the environment dictionary and arg the name of the command line argument (must include leading dashes, for example '--arg').

Command line arguments override environment variables.

class app_manage.config.**DatabaseConfig**
Helper class that runs the value passed through dj_database_url.

class app_manage.config.**Flag**(*name*)
Can be used as the arg argument to a *Config* instance to indicate a boolean flag instead of a command line argument.

class app_manage.config.**Argument**(*config*, *callback*)
Class used to do more complex configurations that affect multiple settings. Instances of this class are passed as args to *app_manage.core.main()*.

config is an instance of a *DynamicSetting* subclass and callback is a callable with the following signature:

callback(*settings*, *value*)

Parameters

- **settings** (*dict*) – Dictionary holding Django settings
- **value** – The value returned by config.

CHAPTER 3

History

3.1 0.1 (In Development)

- First release

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

a

A

C

D

F

G

M

T