Django SmartSettings Documentation

Release 0.1.0

Tomek Kopczuk

Sep 27, 2017

Contents

Release v0.1.0.

Developers should never waste time worrying about using the live database by accident.
On the other hand production often need a different set of middlewares or configs.

These need to be tested as well.
By developers.

Never run on Production by accident and never run your Local settings on Production ever again.
But do anything you want if you want to. Easily.

CHAPTER 1

Intro

This library enables Django projects to be developed by multiple developers and deployed to many environments.

No more usual hassle of having the wrong settings file, new developer not having the right settings file.
No more worrying about the possibility of working on a live database.

CHAPTER 2

Usage

Installation

Distribute & Pip

Install using pip:

```
$ pip install django-smartsettings
```

Install using easy_install easy_install:

```
$ easy_install django-smartsettings
```

Finished? *See the next step*.

Install from source

Grab the latest source.

Clone the repository:

```
git clone git://github.com/tkopczuk/django-smartsettings.git
```

Install:

```
$ python setup.py install
```

Quickstart

Get it running

Put the following code at the end of your **settings.py** file:

```
import smartsettings
smartsettings.config(globals())
```

Create incremental settings files for your environments:

```
cd <directory where your settings.py file resides>
touch devsettings.py
touch productionsettings.py
```

Example files

devsettings.py:

```
DEBUG = True
```

productionsettings.py:

```
DATABASE_URL = "postgres://user:pass@localhost/project"
DEBUG = False
```

You're ready to go!

If you define *DATABASE_URL* instead of *DATABASES* SmartSettings will automatically do:: DATABASES = {'default': dj_database_url.config(default=DATABASE_URL)}

What does it all do

SmartSettings will always load your **settings.py** file and then load the environment's own incremental settings file.

If you don't declare DATABASES setting anywhere, SmartSettings will check the DATABASE_URL variable and, if found, use it through djdatabaseurl.

By default you are always using the DEV settings.

Local developer settings

If you create a local (ignored by DVCS) file called **localsettings.py**, it will automatically be used in the DEV environment.

Developers should keep their local database credentials, cache settings, etc. in this file.

Example *localsettings.py*:

```
DATABASE_URL = "postgres://devuser:devpass@localhost/projectdev"
```

Switching environments

Switching environments is as simple as setting the ENV_FLAVOUR env. variable to your desired environment name.

Example:

```
export ENV_FLAVOUR=PRODUCTION
```

Or:

```
export ENV_FLAVOUR=STAGING
```

If you want to temporarily switch to production settings from your local dev. machine to e.g. run a command, do:

```
export ENV_FLAVOUR=PRODUCTION

<run a command, e.g. ./manage.py migrate>

unset ENV_FLAVOUR
```

Telling production machine it's a production machine

On your production machine add a following line to your **/etc/environment** file:

```
ENV_FLAVOUR=PRODUCTION
```

Advanced settings

Modifying default environments

To modify the default environments you have to provide a dictionary to the SmartSettings constructor:

```
import smartsettings
smartsettings.config({
        'FLAVOURS': (
        'DEV',
        'STAGING',
        'PRODUCTION',
    ),
    'DEFAULT': 'DEV'  # default flavour always loads localsettings.py!
}, globals())
```

This dictionary must contain 2 keys:

- FLAVOURS – tuple or a list defining the environments
- DEFAULT – name of the default DEV environment

Imported incremental settings file is determined by formula:: <lowercase flavour name>settings.py

Hence STAGING flavour will use file:: stagingsettings.py