DBoilerplate Documentation

Release 3.0

Havas Digital Team

September 30, 2016

Contents

Welcome to the dboilerplate3 documentation. dboilerplate is a django project template that contains the most commonly used tools and configurations that we use at Havas Worldwide London.

You can request us to add new features or change stuff around, if it's a good idea we'll do it!. This boilerplate has tons of tiny changes to help us so please pay attention when you read!.

CHAPTER 1

Requirements

This project requires Django 1.7 and Python 3.3 or higher to run. If you're looking for support for django 1.6 or less and python 2.7 please take a look to the original dboilerplate project in here

This documentation should be fairly valid fot both versions of the boilderplate.

CHAPTER 2

Backend

2.1 How to install

Installing the boilerplate is easy, there's two possible ways to do it.

2.1.1 Clone the repository

You can clone the boilerplate repository to your machine, and use it from your machine, that will guarantee you that you always have access to the boilerplate even if you don't have internet connection.

- Clone the repository

```
$ git clone https://github.com/clione/dboilerplate3.git
```

- After cloning the repository to your machine you can start you project (we assume you already installed django either globally or in your virtual environment)

```
$ django-admin startproject --template=/path/to/the/template <your_project_name>
```

2.1.2 Grab it from GitHub

Downloading the boilerplate from GitHub guarantees you that you always have the latest version, and you can do the installation automatically when you create the project.

```
$ django-admin startproject --template=https://github.com/clione/dboilerplate3/archive/master.zip myp
```

2.2 Settings

dboilerplate contains a multienvironment set of settings. Each settings file will load depending on the environment flags.

The settings files are located in *src/<project name>/settings* and there should be five different files:

- __init__.py
- defaults.py
- development.example
- production.py

- staging.py

2.2.1 __init__

The init file contains the main login to select the environment settings files.

It contains two variables, called *DEBUG* and *STAGING*, and the settings work like this:

Variables	Value	Load settings
DEBUG	True	development.py
STAGING	False	
DEBUG	True	staging.py
STAGING	True	
DEBUG	False	production.py
STAGING	False	
DEBUG	False	production.py
STAGING	True	

The variables can be set up in the *__init__.py* file itself, or you can set the environment variables *DJANGO_IS_DEBUG* and *DJANGO_IS_STAGING* if you're using a user-based deployment.

To add the environment variables to you environment and assuming you're using bash, just open your *.bashrc* file and add:

```
export DJANGO_IS_DEBUG="True"
export DJANGO_IS_STAGING="True"
```

Replace the *"True"* values with the values that you want.

2.2.2 defaults.py

The defaults.py file contains the core of any project, it sets up the main settings for the application, applications to be loaded, middlewares, etc.

There is nothing special to say in this file, but you will find here the settings for allauth as well as the logging and other minor global settings.

2.2.3 development, production and staging

These files contain the enviroment specific settings for each one. That includes the database connections, language settings, django toolbar, email, caching, fixtures, secret_key, allowed hosts, etc.

All of them are prepopulated with dummy data so you know what to modify. You can add as well any other settings that you need for your environment or rewrite the ones loaded already in defaults.

2.2.4 development

By default we packa a *development.example* settings file. This is done on purpose, so the project will fail to run until you set up the development settings and rename the file to *development.py*.

2.3 Requirement files

There is a multienvironment requirements file located inside the /requirements folder.

2.3.1 Populating the requirements

To populate the requirements basically open the file related to the environment that need the package and add it there. Usually the commons.txt is used for dependencies that are required across all environments like django itself, or core libraries that the project might use.

2.3.2 Installing requirements

The installation of the requirements is done the same way as with a standard requirements file, but this time, calling the environment file.

An example of how to install the development requirements would be:

```
$ pip install -r requirements/dev.txt
```

2.4 Creating apps

By default django contains a *startapp* command, but in dboilerplate we created an application generator. This comes in handy to save the first 15 minutes of development of the application. It will ask you for some values and create the the application skeleton and the code inside it for you.

2.4.1 How to use it

To create application using the generator you just have to run the management command from /src

```
$ ./manage.py addapp
```

It will ask you the values and create all for you.

2.5 Abstract models

To be written

2.6 Logging

We improved a bit the logging mechanism of django to made our lifes easier.

Apart from the standard django mailer logging we added file logging that will spit out a django.log file inside the project folder (that is: /src/projectname/django.log)

It will create a rotating file, so you will have available all the logs that you need. This logger also logs **absolutely everything** that happens in your code so it can be properly debugged.

2.6.1 Configuring the logger

By default the loger keeps three 2MB files in your installation, you can change that in teh default.py settings file at the end of it. The parameters are:

- maxBytes

- backupCount

The first value is the amount of MB that a file can reach, in bytes. The second is the number of files to keep, by default is three, and it should be enough, but maybe you need more for your project.

2.6.2 How to log to the logfile

Instead of doing prints here and there you can now log to the file doing the following:

Add this to the file where you want to log:

```
import logging

logger = logging.getLogger(__name__)
```

Then to log something you can use the standard python log levels (DEBUG, INFO, ERROR, etc.)

```
logger.error("This is a error message")
logger.debug("This is a debug message")
logger.info("Seems that something happened")
```

2.7 Sitemaps

CHAPTER 3

Frontend

3.1 How to install

3.2 Settings

3.3 How to work

CHAPTER 4

Indices and tables

- genindex
- modindex
- search