django-pip-starter Documentation

Release 1.0.3

Marius Grigaitis

Sep 28, 2017

Contents

Have you ever had problems deploying and configuring `django` project? This project removes headaches that you used to have when quick-starting `django` project, configuring environments, downloading packages and etc.

It creates `django` project by using one simple command, ready for running.

Solution is based on `pip` and `virtualenv`, it has minimal external requirements.

CHAPTER 1

Installation

To install this package:

```
pip install django-pip-starter
```

If you already have django-pip-starter install you can use the following command to upgrade installation:

```
pip install --upgrade django-pip-starter
```

CHAPTER 2

Quick start

The following commands creates empty, configured `django` project in virtual environment. Additionally it will install `south` package. For development environment it additionally installs `django-debug-toolbar`, `ipython`, `ipdb`

```
django-pip-starter.py project-name
cd project-name
make
make run
```

Where `project-name` is destination folder where starter should create files.

`make` command will download and setup development virtualenv, download latest stable `django` version and create sqlite3 database, load initial data.

`make run` will run development server. It's the same as running `project/manage.py runserver` which would also work.

Default logins for django administration are user: `admin` pass: `admin`

CHAPTER 3

Documentation

You can read documentation at http://readthedocs.org/docs/django-pip-starter/

CHAPTER 4

History

Idea for this project came from Mantas Zimickas (sirex, https://bitbucket.org/sirex/django-starter/overview). This was based on `zc.buildout` solution. After some time using `django-starter` we had problems deploying it and Petras Zdanavicius (petraszd) made a fork of `django-starter` that used only `pip`. This was simple and elegant solution that Marius Grigaitis (marltu) expanded and packaged it into this project.

CHAPTER 5

Table of Contents

Layout

```
.
- .hgignore - ignore file for Mercurial VCS
- config - package requirements and other configs
- Makefile - commands for make
- var
|   - development.db - sqlite3 database
|   - htdocs - directory that should be handled by webserver in production
|   |   - media
|   |   - static
|   - mail - directory where mails are stored in development by default
- project
    - development.py.sample
    - development.py - settings used for development environment
    - production.py - settings used for production environment
    - initial_data.json - default logins and site
    - __init__.py
    - manage.py - standart django commands
    - settings.py - settings for development and production
    - static - static files for /static/
    - templates
    - urls.py
    - wsgi.py - script for WSGI
```

Configuration files

There are 3 settings files: `development.py`, `production.py`, `settings.py`.

By checking if settings files exists it's determined which environment should be used. If `production.py` is present - production environment is used.

When running `make` for development environment it creates `development.py` by default if it's not present.

You can also create development.py manually by running:

```
make project/development.py
```

Database

Creating or recreating database

To remove old database, create new, fake migrations (`south`) and load initial data run:

```
make syncdb
```

By default in development environment sqlite3 database is used. It can be located in `var/` directory.

Package management

Structure

There are 3 types of files that can be located in `config/` directory which are used to install packages.

- `requirements.txt` - used for installing packages to all environments
- `devel-requirements.txt` - used for installing packages only to development environment in addition to `requirements.txt`
- `prod-requirements.txt` - used for installing packages only to production environment in addition to `requirements.txt` (by default not present but can be created)

Syntax of requirements files is `pip` requirement file syntax (http://www.pip-installer.org/en/latest/requirements.html)

Commands

To install packages from requirements file run the following command (it skips already installed packages):

```
make requirements
```

You can also update all packages to latest version:

```
make upgrade
```

Overriding

You can manage packages by hand in environments by running pip from environment (you may also use prod-env):

```
devel-env/bin/pip install django
```

Speedup

Sometimes you have many projects and you don't want to download all packages every time. You can use `pip` download cache by setting environment variable `PIP_DOWNLOAD_CACHE`. It should point to directory that you would like to store downloaded files. You may want to set it in your shell rc script.

Deployment

Installation

To make production installation you'll need to create `project/production.py` configuration file before running `make`. This will force using production environment. You can use `development.py.sample` as reference.

Configuration

You should override default configs from `settings.py` in `production.py`. **Don't forget** to specify new `SECRET_KEY`.

Static files

By running `make` in production environment it automatically collects static files into `var/htdocs/static`. You can run it manually:

```
make collectstatic
```

Apache 2 configuration

You can generate virtualhost configuration by running:

```
make config/apache2.conf
```

It creates config file by using config/apache2.conf.sample and:

- replaces `__DOMAIN__` with parent directory name (for example if project is located at `/var/www/vhosts/www.example.com`, `www.example.com` will be used.
- replaces `__STARTER_PATH__` with project location.

You can include it in apache config by using `Include <file>` directive.

Utilities

Mercurial

You can create empty mercurial repository only for starter files by running:

```
make .hg
```

This will initialize repository, add files and commit without message (which should prompt message input).

ctags

To collect ctags run:

```
make ctags
```

It will create `tags` file in main directory.