django-project-portfolio Documentation

Release 1.0

James Bennett

November 14, 2016

Contents

`django-project-portfolio` is a simple Django for displaying information about software projects you maintain.

CHAPTER 1

Documentation contents

1.1 Installation guide

Before installing `django-project-portfolio`, you'll need to have a copy of Django already installed. For information on obtaining and installing Django, consult the Django download page, which offers convenient packaged downloads and installation instructions.

The 1.0 release of `django-project-portfolio` supports Django 1.7 and 1.8, on any of Python 2.7, 3.3 or 3.4. Older versions of Django and/or Python are not supported.

1.1.1 Normal installation

The preferred method of installing `django-project-portfolio` is via `pip`, the standard Python package-installation tool. If you don't have `pip`, instructions are available for how to obtain and install it.

Once you have `pip`, simply type:

```
pip install django-project-portfolio
```

1.1.2 Manual installation

It's also possible to install `django-project-portfolio` manually. To do so, obtain the latest packaged version from the listing on the Python Package Index. Unpack the `.tar.gz` file, and run:

```
python setup.py install
```

Once you've installed `django-project-portfolio`, you can verify successful installation by opening a Python interpreter and typing `import projects`.

If the installation was successful, you'll simply get a fresh Python prompt. If you instead see an `ImportError`, check the configuration of your install tools and your Python import path to ensure `django-project-portfolio` installed into a location Python can import from.

1.1.3 Installing from a source checkout

The development repository for `django-project-portfolio` is at <https://github.com/ubernostrum/django-project-portfolio>. Presuming you have git installed, you can obtain a copy of the repository by typing:

```
git clone https://github.com/ubernostrum/django-project-portfolio.git
```

From there, you can use normal git commands to check out the specific revision you want, and install it using `python setup.py install`.

1.2 Models for software projects

`django-project-portfolio` provides three models which work together to describe software projects: *`Project`* represents a software project, *`Version`* represents a particular version of a project, and *`License`* represents the license under which a particular version is released.

class `projects.models.`**`License`**

The license under which a particular *`Version`* is released. This is tied to `Version` rather than *`Project`* in order to allow the possibility of relicensing from one version to another.

A `License` has three fields, all of which are required:

`name`

`CharField(max_length=255)`

The name of the license (for example, "GPLv2" or "MIT").

`slug`

`SlugField` (prepopulated from *`name`*)

A short, descriptive URL-safe string to identify the license. Currently there are no views in `django-project-portfolio` which make use of this, but the field is provided so that custom views can make use of it.

`link`

`URLField`

A link to an online version of the license's terms, or to a description of the license. For open-source licenses, individual license pages in the OSI license list are useful values for this field.

class `projects.models.`**`Project`**

A software project.

Four fields (all required) provide basic metadata about the project:

`name`

`CharField(max_length=255)`

The name of the project.

`slug`

`SlugField` (prepopulated from *`name`*)

A short, descriptive URL-safe string to identify the project.

`description`

`TextField`

A free-form text description of the project.

`status`

`IntegerField` with choices

Indicates whether the project is public or not. May be expanded to include additional options in future versions, hence the implementation as an `IntegerField` with choices instead of a `BooleanField`. Valid choices are:

PUBLIC_STATUS
Indicates a project which is public; this will cause built-in views to list and display the project.

HIDDEN_STATUS
Indicates a project which is hidden; built-in views will not list or display the project.

Four additional fields, all optional, allow additional useful data about the project to be specified:

package_link
URLField

URL of a location where packages for this project can be found.

repository_link
URLField

URL of the project's source-code repository.

documentation_link
URLField

URL of the project's online documentation.

tests_link
URLField

URL of the project's online testing/continuous integration status.

One utility method is also defined on instances of `Project`:

latest_version()
Returns the latest *Version* of this project (as defined by the `is_latest` field on `Version`), or `None` if no such version exists.

Finally, the default manager for `Project` defines one custom query method, `public()`, which returns only instances whose *status* is *PUBLIC_STATUS*. This is implemented via a custom `QuerySet` subclass, so the method will be available on any `QuerySet` obtained from `Project` as well.

class projects.models.**Version**
A particular version of a software project.

There are six fields, all of which are required:

project
ForeignKey to *Project*

The project this version corresponds to.

version
CharField(max_length=255)

A string representing the version's identifier. This is deliberately freeform to support different types of versioning systems, but be aware that it will (with the built-in views) be used in URLs, so URL-safe strings are encouraged here.

is_latest
BooleanField

Indicates whether this is the latest version of the project. When a `Version` is saved with `is_latest=True`, a `post_save` signal handler will toggle all other versions of that *Project* to `is_latest=False`.

status
IntegerField with choices

The status of this version. Valid choices are (taken from the Python Package Index's status choices):

PLANNING_STATUS
This is an early/planning version.

PRE_ALPHA_STATUS
This is a pre-alpha version.

ALPHA_STATUS
This is an alpha version.

BETA_STATUS
This is a beta version.

STABLE_STATUS
This is a stable version.

license
`ForeignKey` to *License*

The license under which this version is released.

release_date
The date on which this version was released.

Additionally, the default manager for `Version` defines one custom query method, `stable()`, which returns only instances whose *status* is *STABLE_STATUS*. This is implemented via a custom `QuerySet` subclass, so the method will be available on any `QuerySet` obtained from `Version` as well, and also on any related `QuerySet` obtained through an instance of *Project*.

1.3 Views for software projects

`django-project-portfolio` provides four built-in views for displaying information about software projects. Though not all possible views of the data are included here, the built-in views strive to cover the common cases.

class `projects.views.`**ProjectDetail**
Subclass of Django's generic DetailView.

Detail view of a *Project*. Has one required argument which must be captured in the URL:

`slug`

The *slug* of the project.

By default, this view will only display projects whose *status* is *PUBLIC_STATUS*.

class `projects.views.`**ProjectList**
Subclass of Django's generic ListView.

List of *Project* instances.

By default, this view will only display projects whose *status* is *PUBLIC_STATUS*.

class `projects.views.`**VersionDetail**
Subclass of Django's generic DetailView.

Detail view of a *Version*. Has two required arguments which must be captured in the URL:

`project_slug`

The *slug* of the *Project* with which this `Version` is associated.

`slug`

The *version* of the Version.

By default, only versions associated with a *Project* whose *status* is *PUBLIC_STATUS* can be displayed.

class projects.views.**LatestVersionList**

Subclass of django.views.generic.ListView.

List of the latest *Version* of each public (i.e., *status* is *PUBLIC_STATUS*) *Project*.

p