Django

Documentation

Advanced tutorial: How to write reusable apps

This advanced tutorial begins where Tutorial 7 left off. We'll be turning our Web-poll into a standalone Python package you can reuse in new projects and share with other people.

If you haven't recently completed Tutorials 1–7, we encourage you to review these so that your example project matches the one described below.

Reusability matters

It's a lot of work to design, build, test and maintain a web application. Many Python and Django projects share common problems. Wouldn't it be great if we could save some of this repeated work?

Reusability is the way of life in Python. The Python Package Index (PyPI) has a vast range of packages you can use in your own Python programs. Check out Django Packages for existing reusable apps you could incorporate in your project. Django itself is also just a Python package. This means that you can take existing Python packages or Django apps and compose them into your own web project. You only need to write the parts that make your project unique.

Let's say you were starting a new project that needed a polls app like the one we've been working on. How do you make this app reusable? Luckily, you're well on the way already. In Tutorial 3, we saw how we could decouple polls from the project-level URLconf using an `include`. In this tutorial, we'll take further steps to make the app easy to use in new projects and ready to publish for others to install and use.

Package? App?

A Python package provides a way of grouping related Python code for easy reuse. A package contains one or more files of Python code (also known as "modules").

A package can be imported with `import foo.bar` or `from foo import bar`. For a directory (like `polls`) to form a package, it must contain a special file `__init__.py`, even if this file is empty.

A Django *application* is just a Python package that is specifically intended for use in a Django project. An application may use common Django conventions, such as having `models`, `tests`, `urls`, and `views` submodules.

Later on we use the term *packaging* to describe the process of making a Python package easy for others to install. It can be a little confusing, we know.

Your project and your reusable app

After the previous tutorials, our project should look like this:

Language: **en**

Documentation version: **1.9**

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    polls/
        __init__.py
        admin.py
        migrations/
            __init__.py
            0001_initial.py
        models.py
        static/
            polls/
                images/
                    background.gif
                style.css
        templates/
            polls/
                detail.html
                index.html
                results.html
        tests.py
        urls.py
        views.py
    templates/
        admin/
            base_site.html
```

You created **mysite/templates** in Tutorial 7, and **polls/templates** in Tutorial 3. Now perhaps it is clearer why we chose to have separate template directories for the project and application: everything that is part of the polls application is in **polls**. It makes the application self-contained and easier to drop into a new project.

The **polls** directory could now be copied into a new Django project and immediately reused. It's not quite ready to be published though. For that, we need to package the app to make it easy for others to install.

Installing some prerequisites

The current state of Python packaging is a bit muddled with various tools. For this tutorial, we're going to use setuptools to build our package. It's the recommended packaging tool (merged with the **distribute** fork). We'll also be using pip to install and uninstall it. You should install these two packages now. If you need help, you can refer to how to install Django with pip. You can install **setuptools** the same way.

Packaging your app

Python *packaging* refers to preparing your app in a specific format that can be easily installed and used. Django itself is packaged very much like this. For a small app like polls, this process isn't too difficult.

1. First, create a parent directory for **polls**, outside of your Django project. Call this directory **django-polls**.

> **Choosing a name for your app**
>
> When choosing a name for your package, check resources like PyPI to avoid naming conflicts with existing packages. It's often useful to prepend **django-** to your module name when creating a package to distribute. This helps others looking for Django apps identify your app as Django specific.
>
> Application labels (that is, the final part of the dotted path to application packages) *must* be unique in **INSTALLED_APPS**. Avoid using the same label as any of the Django contrib packages, for example **auth**, **admin**, or **messages**.

2. Move the **polls** directory into the **django-polls** directory.

3. Create a file **django-polls/README.rst** with the following contents:

```
django-polls/README.rst
```

```
=====
Polls
=====

Polls is a simple Django app to conduct Web-based polls. For each
question, visitors can choose between a fixed number of answers.

Detailed documentation is in the "docs" directory.

Quick start
-----------

1. Add "polls" to your INSTALLED_APPS setting like this::

    INSTALLED_APPS = [
        ...
        'polls',
    ]

2. Include the polls URLconf in your project urls.py like this::

    url(r'^polls/', include('polls.urls')),

3. Run `python manage.py migrate` to create the polls models.

4. Start the development server and visit http://127.0.0.1:8000/admin/
   to create a poll (you'll need the Admin app enabled).

5. Visit http://127.0.0.1:8000/polls/ to participate in the poll.
```

4. Create a **django-polls/LICENSE** file. Choosing a license is beyond the scope of this tutorial, but suffice it to say that code released publicly without a license is *useless*. Django and many Django-compatible apps are distributed under the BSD license; however, you're free to pick your own license. Just be aware that your licensing choice will affect who is able to use your code.

5. Next we'll create a **setup.py** file which provides details about how to build and install the app. A full explanation of this file is beyond the scope of this tutorial, but the setuptools docs have a good explanation. Create a file **django-polls/setup.py** with the following contents:

django-polls/setup.py

```
import os
from setuptools import find_packages, setup

with open(os.path.join(os.path.dirname(__file__), 'README.rst')) as readme:
    README = readme.read()

# allow setup.py to be run from any path
os.chdir(os.path.normpath(os.path.join(os.path.abspath(__file__), os.pardir)))

setup(
    name='django-polls',
    version='0.1',
    packages=find_packages(),
    include_package_data=True,
    license='BSD License',  # example license
    description='A simple Django app to conduct Web-based polls.',
    long_description=README,
    url='https://www.example.com/',
    author='Your Name',
    author_email='yourname@example.com',
    classifiers=[
        'Environment :: Web Environment',
        'Framework :: Django',
        'Framework :: Django :: X.Y',  # replace "X.Y" as appropriate
        'Intended Audience :: Developers',
        'License :: OSI Approved :: BSD License',  # example license
        'Operating System :: OS Independent',
        'Programming Language :: Python',
        # Replace these appropriately if you are stuck on Python 2.
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.4',
        'Programming Language :: Python :: 3.5',
        'Topic :: Internet :: WWW/HTTP',
        'Topic :: Internet :: WWW/HTTP :: Dynamic Content',
    ],
)
```

Language: **en**

Documentation version: **1.9**

6. Only Python modules and packages are included in the package by default. To include additional files, we'll need to create a **MANIFEST.in** file. The setuptools docs referred to in the previous step discuss this file in more details. To include the templates, the **README.rst** and our **LICENSE** file, create a file **django-polls/MANIFEST.in** with the following contents:

   django-polls/MANIFEST.in

```
include LICENSE
include README.rst
recursive-include polls/static *
recursive-include polls/templates *
```

7. It's optional, but recommended, to include detailed documentation with your app. Create an empty directory **django-polls/docs** for future documentation. Add an additional line to **django-polls/MANIFEST.in**:

```
recursive-include docs *
```

   Note that the **docs** directory won't be included in your package unless you add some files to it. Many Django apps also provide their documentation online through sites like readthedocs.org.

8. Try building your package with **python setup.py sdist** (run from inside **django-polls**). This creates a directory called **dist** and builds your new package, **django-polls-0.1.tar.gz**.

For more information on packaging, see Python's Tutorial on Packaging and Distributing Projects.

Using your own package

Since we moved the **polls** directory out of the project, it's no longer working. We'll now fix this by installing our new **django-polls** package.

Installing as a user library

The following steps install **django-polls** as a user library. Per-user installs have a lot of advantages over installing the package system-wide, such as being usable on systems where you don't have administrator access as well as preventing the package from affecting system services and other users of the machine.

Note that per-user installations can still affect the behavior of system tools that run as that user, so **virtualenv** is a more robust solution (see below).

1. To install the package, use pip (you already installed it, right?):

```
pip install --user django-polls/dist/django-polls-0.1.tar.gz
```

2. With luck, your Django project should now work correctly again. Run the server again to confirm this.

3. To uninstall the package, use pip:

```
pip uninstall django-polls
```

Language: **en**

Documentation version: **1.9**

Publishing your app

Now that we've packaged and tested **`django-polls`**, it's ready to share with the world! If this wasn't just an example, you could now:

- Email the package to a friend.
- Upload the package on your website.
- Post the package on a public repository, such as the Python Package Index (PyPI). packaging.python.org has a good tutorial for doing this.

Installing Python packages with virtualenv

Earlier, we installed the polls app as a user library. This has some disadvantages:

- Modifying the user libraries can affect other Python software on your system.
- You won't be able to run multiple versions of this package (or others with the same name).

Typically, these situations only arise once you're maintaining several Django projects. When they do, the best solution is to use virtualenv. This tool allows you to maintain multiple isolated Python environments, each with its own copy of the libraries and package namespace.

‹ Writing your first Django app, part 7

What to read next ›

News RSS

Django Users Mailing List

Language: **en**

Documentation version: **1.9**