pyscaffoldext-django Documentation

Release unknown

Anderson Bravalheri

Mar 13, 2022

CONTENTS

Integration of **Django**'s built-in generator (`django-admin`) into **PyScaffold**

CHAPTER ONE

CONTENTS

1.1 pyscaffoldext-django

Integration of **Django**'s built-in generator (`django-admin`) into **PyScaffold**

PyScaffold is a development tool focused in distributable Python packages. This extension allows the development of Django websites using PyScaffold sensible project structure, by tapping into the django-admin cli.

LOOKING FOR CONTRIBUTORS - If you use PyScaffold or Django and would like to help us as a contributor (or even as one of the maintainers) for this extension, please send us an email or open an issue, we would love to have you on board.

1.1.1 Quickstart

This extension can be directly installed with `pip`:

```
pip install pyscaffoldext-django
```

Or, if you prefer `pipx`:

```
pipx install pyscaffold  # if you haven't installed pyscaffold yet
pipx inject pyscaffold pyscaffoldext-django
```

Note that, after the installation, `putup -h` will show a new option `--django`. Use this option to indicate when you are trying to create a django app. For example:

```
putup --django myapp
```

Please refer to django-admin documentation for more details.

1.1.2 Alternative Procedure

Using Django extension in PyScaffold is roughly equivalent to first create an app using django-admin and then convert it to PyScaffold. The following manual procedure can be used to replace `pyscaffoldext-django`:

```
django-admin startproject myapp
mkdir myapp/src
mv myapp/myapp myapp/src
mv myapp/manage.py myapp/src/myapp/__main__.py

# edit the location of the database in myapp/src/myapp/setttings.py
# to point to one directory up, similar to:
#
#   PROJECT_DIR = os.path.dirname(BASE_DIR)
#   DATABASES = {'default': { ..., 'NAME': os.path.join(PROJECT_DIR, 'db.sqlite3')}}

putup myapp --force
```

We move/rename the `manage.py` file to `myapp/src/myapp/__main__.py`. This makes it possible to manage the application using `python -m myapp` when it is installed as a package (instead of `python manage.py`). All the arguments remain the same. Please check the next section for more information.

Running the script with `python -m` requires your package to be installed (a simple `pip install -e .` will suffice), however we also generate a new `manage.py` file that is a simple stub pointing to the `__main__.py` and works without explicit installation.

For complex use cases, maybe a better option is to do the conversion manually. If you find problems running PyScaffold with `--django` please try to execute this procedure.

1.1.3 Distributable Django Packages

Django is a framework for creating web applications, and PyScaffold is a tool that helps to build re-usable, distributable packages - which most of the time correspond to libraries or command line tools.

While those two definitions are not mutually exclusive, it is a bit tricky to create a package with PyScaffold that serves a Django app. The first reason is that applications usually require concrete dependencies (pinned version numbers), while libraries are more relaxed and tend to use abstract dependencies (ranges of version numbers). You can read all about the differences between those two approaches in PyScaffold's documentation, however the main point is: when creating packages for webapps you have two options

Use concrete dependencies pin the exact version number for your dependencies to avoid bugs (*it works on my machine*^{TM}), but instruct your users that the package should be installed within a **dedicated** virtual environment to avoid dependency hell; or

Use abstract dependencies prefer relaxed dependency ranges (e.g. relying on stable APIs of dependencies that use semver), but test extensively your module against different installed versions to make sure nothing breaks (tox and nox are good tools for that).

The second reason is that Django expects the user of your application to have control on where the source code is placed, and this simply doesn't go well with pip installing locations deeply hidden somewhere in the file system (e.g. `/home/username/my-venvs/web-app/lib/python3.6/site-packages/my-web-app`)...

For example, before starting a Django application server you are supposed to run migrations to prepare the correct structure in the database to receive your data. This is usually achieved by running `python manage.py migrate` at the root of your directory, however, if someone is installing your app using pip, how does this person knows where to find the `manage.py` file?

To solve this problem, `pyscaffoldext-django` renames `manage.py` to `__main__.py` and moves it inside your web application package. Since it becomes part of your package, the script will be accessible via `python -m YOUR_PACKAGE_NAME <commands>` from everywhere in the system, and therefore no one installing it with pip needs to know where it is.

Another example of the same behaviour is the default SQLite database Django creates. If you simply turn an Django app that was not created with PyScaffold into a package, install it and run the migrations, Django will generate an SQLite file in an arbitrary location in your disk. PyScaffold cannot automatically solve this problem for you. Instead you can follow a few approaches:

1. (*NOT RECOMMENDED*) place your SQLite database inside your package and distribute it as a package data, accessing it via importlib.resources. (Please note resources are supposed to be immutable and not re-written to disk)
2. Allow the person installing your package to specify a different configuration via environment variables. According to the Mozilla's tutorials, the library dj-database-url is good for that.
3. Place your SQLite database somewhere in the user home.

For the sake of pragmatism, PyScaffold will reconfigure `settings.py` to place the database inside the project root in the development environment, but it is your responsibility to change this when going into production.

Finally, it is important to notice that, while it is popular in the Django community to create separated top-level folders for independent applications, this is more or less incompatible with the concept of a Python package… One entry in PyPI should install a single package in your machine. Ideally, if you use multiple apps, you should deploy a different package for each of them and declare them as dependencies of your main project. Alternatively you can also deploy new applications nested inside of your main project package (the one generated by PyScaffold/`django-admin startproject`). Therefore, caution is required when using `python manage.py startapp` (you should either provide the optional `directory` parameter as somewhere inside of your main package, or skip it completely). One example on how to use nested apps is:

```
putup --django website
cd website
# ... do some coding
mkdir src/website/subapp
python manage.py startapp subapp src/website/subapp
# OR python -m website startapp subapp src/website/subapp
#    if you have the package installed in the dev environment
# ... then you can add "website.subapp" to INSTALLED_APPS in src/website/settings.py
# ... remeber to use relative imports or the full package name "website.subapp" when␣
↪needed
```

1.1.4 Tips

1. Have a look on Django's guides, but remember that PyScaffold already do the heavy lifting for you (no need to write packaging configuration from scratch) and that we use a src-based layout
2. Do not assume anything about the file system where the package will be installed.
3. If you really need to write things to disk, you can follow the XDG standards and write to `$XDG_DATA_HOME` (the package appdirs might help).
4. Accept configurations via environment variables, and throw meaningful errors when they are not provided. Even if you prefer reading configurations from a file, you can always let the person installing your package to specify a location for this file as an environment variable.
5. Use environment variables as flags/switches to enable/disable features or select alternative implementations.

6. Be extra careful to not store secrets and confidential info in your source repository.
7. Be extra careful with secrets and confidential info **IN GENERAL**. If really required to store them, use well known cryptography techniques and tweak file/folder permissions in your operating system (e.g. the command `chmod og-rwx` is your friend, but you can also consider `400` permissions). Instructing the person installing your package to create a separated system account to run your web app with limited privileges might also be good.
8. Provide extensive documentation on how your users are supposed to install and run your app (e.g. virtualenv installation instructions, ngnix/apache/systemd configuration examples, etc…)

1.1.5 Making Changes & Contributing

This project uses pre-commit, please make sure to install it before making any changes:

```
pip install pre-commit
cd pyscaffoldext-django
pre-commit install
```

It is a good idea to update the hooks to the latest version:

```
pre-commit autoupdate
```

Please also check PyScaffold's contribution guidelines,

1.1.6 Note

This project has been set up using PyScaffold 4.0a2. For details and usage information on PyScaffold see https://pyscaffold.org/.

1.2 pyscaffoldext

1.2.1 pyscaffoldext namespace

Subpackages

pyscaffoldext.django package

Subpackages

pyscaffoldext.django.templates package

Module contents

Submodules

pyscaffoldext.django.extension module

Module contents

1.3 License

1.4 Contributors

- Anderson Bravalheri <andersonbravalheri@gmail.com>

1.5 Changelog

1.5.1 Version 0.2

- Updated to Django v4

1.5.2 Version 0.1.1

- Improved lookup for django-admin via `pyscaffold.shell.get_command`, see issue #3
- Fix invalid cache in Cirrus CI by using a fingerprint script

1.5.3 Version 0.1

- Initial release extracted from PyScaffold
- Updated to PyScaffold v4
- Added `__main__.py` and custom `manage.py` to bypass Django's packaging limitations, PR #2
- Added support for project path different from package name

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX