
django-opfield

Josh Thomas

Sep 30, 2024

DEVELOPMENT

| | | |
|----------|---|-----------|
| 1 | Requirements | 3 |
| 2 | Getting Started | 5 |
| 3 | Usage | 7 |
| 3.1 | Defining a model | 7 |
| 3.2 | Accessing the secret | 7 |
| 3.3 | Storing references, not secrets | 8 |
| 4 | Documentation | 9 |
| 5 | License | 11 |
| 5.1 | Justfile | 11 |
| 5.1.1 | Commands | 12 |

A custom Django field that integrates with the 1Password op CLI to securely access secrets via the `op:// secret reference URI`.

REQUIREMENTS

- Python 3.8, 3.9, 3.10, 3.11, 3.12
- Django 4.2, 5.0
- [1Password CLI](#) and a [1Password Service Account](#)

GETTING STARTED

1. Install the package from PyPI:

```
python -m pip install django-opfield
```

2. Install the `1Password` CLI tool, making sure it is callable from wherever your application is running.
3. Create a `1Password` service account and make the service account's token available to your application.

Choose one option:

- Set the `OP_SERVICE_ACCOUNT_TOKEN` environment variable
- Configure in your application's `settings.py`:

```
# settings.py
DJANGO_OPFIELD = {
    # Explicitly set here only as an example
    # Use whatever configuration/environment library you prefer
    # (python-dotenv, django-environs, environs, etc.)
    "OP_SERVICE_ACCOUNT_TOKEN": "super-secret-token",
}
```


OPField allows Django models to securely access secrets stored in a 1Password vault, enabling the integration of sensitive data without exposing it directly in your codebase. Secrets are stored using the `op://` URI scheme and can be retrieved dynamically using a corresponding model attribute, `<field_name>_secret`.

3.1 Defining a model

First, let's define a model that includes the OPField. This field will store the reference to the secret in 1Password, not the secret itself.

```
from django.db import models

from django_opfield.fields import OPField

class APIService(models.Model):
    name = models.CharField(max_length=255)
    api_key = OPField()

    def __str__(self):
        return self.name
```

3.2 Accessing the secret

Assume you have a secret API key stored in a 1Password vault named “my_vault” under the item “my_api” with the field “api_key”. Here's how you can store and access this secret within your Django project:

```
>>> from example.models import APIService
>>> my_api = APIService.objects.create(
...     name="My API", api_key="op://my_vault/my_api/api_key"
... )
>>> print(my_api)
<APIService: My API>
>>> print(my_api.name)
'My API'
>>> print(my_api.api_key)
'op://my_vault/my_api/api_key'
>>> # Retrieving the actual secret value is done using the automatically generated '_
```

(continues on next page)

(continued from previous page)

```
↳ secret' attribute  
>>> print(my_api.api_key_secret)  
'your_super_secret_api_token_here'
```

3.3 Storing references, not secrets

Only the URI reference to the secret is ever stored and exposed in the Django admin interface and the database. The actual secret itself is never stored and is only retrieved dynamically when accessed. This approach enables secure management and access to secrets throughout your Django application, safeguarding against potential security vulnerabilities associated with direct exposure.

DOCUMENTATION

Please refer to the [documentation](#) for more information.

`django-opfield` is licensed under the MIT license. See the *LICENSE* file for more information.

5.1 Justfile

This project uses `Just` as a command runner.

The following commands are available:

- *bootstrap*
- *copier-copy*
- *copier-update*
- *copier-update-all*
- *coverage*
- *docs-build*
- *docs-install*
- *docs-serve*
- *fmt*
- *install*
- *lint*
- *makemigrations*
- *manage*
- *migrate*
- *pup*
- *test*
- *testall*
- *types*

5.1.1 Commands

```
$ just --list
```

Available recipes:

```
bootstrap
copier-copy TEMPLATE_PATH DESTINATION_PATH="." # apply a copier template to project
copier-update ANSWERS_FILE *ARGS # update the project using a copier
↳ answers file
copier-update-all *ARGS # loop through all answers files and
↳ update the project using copier
coverage
docs-build LOCATION="docs/_build/html"
docs-install
docs-serve
fmt # format justfile
install
lint # run pre-commit on all files
makemigrations *APPS
mm *APPS # alias for `makemigrations`
manage *COMMAND
migrate *ARGS
pup
test *ARGS
testall *ARGS
types
```

bootstrap

```
$ just bootstrap
```

```
bootstrap:
  @just pup
  @just install
```

copier-copy

```
$ just copier-copy
```

```
# apply a copier template to project
copier-copy TEMPLATE_PATH DESTINATION_PATH="." :
  copier copy {{ TEMPLATE_PATH }} {{ DESTINATION_PATH }}
```


copier-update

```
$ just copier-update
```

```
# update the project using a copier answers file
copier-update ANSWERS_FILE *ARGS:
    copier update --trust --answers-file {{ ANSWERS_FILE }} {{ ARGS }}
```

copier-update-all

```
$ just copier-update-all
```

```
# loop through all answers files and update the project using copier
@copier-update-all *ARGS:
    for file in `ls .copier/`; do just copier-update .copier/$file "{{ ARGS }}" done
```

coverage

```
$ just coverage
```

```
coverage:
    python -m nox --session "coverage"
```

docs-build

```
$ just docs-build
```

```
@docs-build LOCATION="docs/_build/html":
    just _cog
    sphinx-build docs {{ LOCATION }}
```

docs-install

```
$ just docs-install
```

```
@docs-install:
    @just pup
    python -m uv pip install 'django-opfield[docs] @ .'
```

docs-serve

```
$ just docs-serve
```

```
@docs-serve:
  #!/usr/bin/env sh
  just _cog
  if [ -f "./.dockerenv" ]; then
    sphinx-autobuild docs docs/_build/html --host "0.0.0.0"
  else
    sphinx-autobuild docs docs/_build/html --host "localhost"
  fi
```

fmt

```
$ just fmt
```

```
# format justfile
fmt:
  just --fmt --unstable
```

install

```
$ just install
```

```
install:
  python -m uv pip install --editable './[dev]'
```

lint

```
$ just lint
```

```
# run pre-commit on all files
lint:
  python -m nox --session "lint"
```

makemigrations

```
$ just makemigrations
```

```
makemigrations *APPS:
  @just manage makemigrations {{ APPS }}
```

manage

```
$ just manage
```

```
manage *COMMAND:
    #!/usr/bin/env python
    import sys

    try:
        from django.conf import settings
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc

    settings.configure(INSTALLED_APPS=["django_opfield"])
    execute_from_command_line(sys.argv + "{{ COMMAND }}".split(" "))
```

migrate

```
$ just migrate
```

```
migrate *ARGS:
    @just manage migrate {{ ARGS }}
```

pup

```
$ just pup
```

```
pup:
    python -m pip install --upgrade pip uv
```

test

```
$ just test
```

```
test *ARGS:
    python -m nox --session "test" -- "{{ ARGS }}"
```

testall

```
$ just testall
```

```
testall *ARGS:  
  python -m nox --session "tests" -- "{{ ARGS }}"
```

types

```
$ just types
```

```
types:  
  python -m nox --session "mypy"
```