django-knocker Documentation

Release 0.5.1

Iacopo Spalletti

Apr 16, 2024

CONTENTS

Contents:

CHAPTER

ONE

DJANGO-KNOCKER

Channels-based desktop notification system

1.1 Documentation

The full documentation is at https://django-knocker.readthedocs.io.

1.2 Usage

See https://django-knocker.readthedocs.io/en/latest/usage.html

1.3 Features

- Sends desktop notifications to connected browsers
- Multilianguage support (with django-parler and django-hvad)
- Uses django-meta API for a consistent metadata handling

1.4 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements-test.txt
(myenv) $ python cms_helper.py
```

1.5 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage-helper

CHAPTER

TWO

INSTALLATION

- Install it:

```
pip install django-knocker
```

- Add it to INSTALLED_APPS with channels:

```
INSTALLED_APPS = [
    ...
    'channels,
    'knocker',
    ...
]
```

- Load the knocker routing into channels configuration:

```
CHANNEL_LAYERS={
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            'hosts': [os.environ.get('REDIS_URL', 'redis://localhost:6379')],
        }
    },
}

ASGI_APPLICATION='myproject.routing.channel_routing',
```

Check channels documentation for more detailed information on CHANNEL_LAYERS setup.

- Add to myproject.routing.channel_routing.py the knocker routes:

```
# -*- coding: utf-8 -*-

from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
from django.urls import path
from knocker.routing import channel_routing as knocker_routing

application = ProtocolTypeRouter({
    'websocket': AuthMiddlewareStack(
        URLRouter([
            path('knocker/', knocker_routing),
        ])
```

(continues on next page)

(continued from previous page)

```
    ),
})
```

2.1 Upgrade

Upgrade from channels 1 version of django-knocker require updating the configuration and minor changes

2.1.1 Configuration

- Discard existing configuration
- Rewrite the main router according to channels 2 specifications and include knocker router. Example:

```
application = ProtocolTypeRouter({
    'websocket': AuthMiddlewareStack(
        URLRouter([
            path('knocker/', knocker_routing),
        ])
    ),
})
```

2.1.2 API Changes

If you added a custom `should_knock` or `as_knock` methods, you must add the `signal_type` argument to match the current signature:

```
def should_knock(self, signal_type, created=False):
    ...

def def as_knock(self, signal_type, created=False):
    ...
```

CHAPTER

THREE

USAGE

After installing and configuring it, you need to adapt your models to use `knocker` interface.

- Extend your model to use `KnockerModel` and `ModelMeta`
- Override the *api* if needed
- Load `{% static "js/knocker.js" %}` and `{% static "js/reconnecting-websocket.min.js" %}` into the templates
- Add the following code:

  ```
  <script type="text/javascript">
    var knocker_language = '{{ LANGUAGE_CODE }}';
    var knocker_url = '/notifications';  // Set this to the actual URL
  </script>
  ```

  The value of `knocker_url` must match the path configured in `myproject.routing.channel_routing.py`.

- Deploy you project according to the channels documentation

Now, for every user which has of the knocker-enabled pages opened, whenever an instance of your knocker-enabled models is saved, a desktop notification is emitted.

Knocker provides a default signal which is fired whenever a model instance is saved and is registered automatically.

If you have any issue with signal firing, please open an issue.

For a complete implementation of a knocker-enabled application refer to the sample app included in knocker tests.

CHAPTER

FOUR

KNOCKER API

The Knocker API is a very thin layer of syntactic sugar on top of django-meta and channels.

4.1 Attributes

KnockerModel mixin defines the attribute to build the notification information:

```
_knocker_data = {
    'title': 'get_knocker_title',
    'message': 'get_knocker_message',
    'icon': 'get_knocker_icon',
    'url': 'get_absolute_url',
    'language': 'get_knocker_language',
}
```

Each key in the _knocker_data attribute is an attribute of the notification package delivered to the client. Each key can be overridden in the __init__ method or the attribute entirely redefined in the model class:

```
class Post(KnockerModel, ModelMeta, models.Model):
    title = models.CharField(_('Title'), max_length=255)
    ...

    _knocker_data = {
        'title': 'get_my_title',
        'message': 'get_message',
        'icon': 'get_knocker_icon',
        'url': 'get_absolute_url',
        'language': 'get_knocker_language',
    }

    def get_message(self):
        return self.title

    def get_my_title(self):
        return 'hello'
```

4.1.1 Attributes

- title: the title that appears in the desktop notification; defaults to `New Model {{ verbose name }}`;
- message: the content of the desktop notification; default to the result of `self.get_title` on the model instance;
- icon: an icon displayed on the notification; defaults to the value of `KNOCKER_ICON_URL`;
- url: the url the notification is linked to; default to the model `get_absolute_url`;
- language: the language group the notification is sent; if the model uses django-parler or django-hvad the language of the instance is determined by calling `self.get_current_language()`, otherwise the current django language is used.

4.2 Methods

`django-knocker` defines a few methods that are intended to be overridden in the models

`class knocker.mixins.`**`KnockerModel`**`(`**args*, ***kwargs*`)`

`get_knocker_icon``()`

Generic function to return the knock icon

Defaults to the value of settings.KNOCKER_ICON_URL

`get_knocker_language``()`

Returns the current language.

This will call `selg.get_current_language` if available or the Django `django.utils.translation.get_language()` otherwise

`get_knocker_message``()`

Generic function to return the knock message.

Defaults to calling `self.get_title`

`get_knocker_title``()`

Generic function to return the knock title.

Defaults to 'new *model_verbose_name*'

`should_knock``(`*signal_type*, *created=False*`)`

Generic function to tell whether a knock should be emitted.

Override this to avoid emitting knocks under specific circumstances (e.g.: if the object has just been created or update)

Parameters

- **`signal_type`** – type of signal between pre_save, post_save, pre_delete, post_delete
- **`created`** – True if the object has been created

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/nephila/django-knocker/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

django-knocker could always use more documentation, whether as part of the official django-knocker docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/nephila/django-knocker/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up `django-knocker` for local development.

1. Fork the `django-knocker` repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/django-knocker.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv django-knocker
   $ cd django-knocker/
   $ pip install -r requirements-test.txt
   $ pip install -e .
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, pip install it into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

Development tips

This project allows you to use pre-commit to ensure an easy compliance to the project code styles.

If you want to use it, install it globally (for example with `pip3 install --user precommit`, but check *installation instruction <https://pre-commit.com/#install>*. When first cloning the project ensure you install the git hooks by running `pre-commit install`.

From now on every commit will be checked against our code style.

Check also the available tox environments with `tox -l`: the ones not marked with a python version number are tools to help you work on the project buy checking / formatting code style, running docs etc.

Testing tips

You can test your project using any specific combination of python, django and django cms.

For example `tox -epy37-django30-cms37` runs the tests on python 3.7, Django 3.0 and django CMS 3.7.

As the project uses pytest as test runner, you can pass any pytest option by setting the `PYTEST_ARGS` environment variable, usually by prepending to the `tox` command. Example:

```
PYTEST_ARGS=" -s  tests/test_plugins.py::PluginTest -p no:warnings" tox -epy37-django30-
↪cms37
```

5.1.6 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. Pull request must be named with the following naming scheme:

   `<type>/(<optional-task-type>-)<number>-description`

   See below for available types.

2. The pull request should include tests.

3. If the pull request adds functionality, the docs should be updated. Documentation must be added in `docs` directory, and must include usage information for the end user. In case of public API method, add extended docstrings with full parameters description and usage example.

4. Add a changes file in `changes` directory describing the contribution in one line. It will be added automatically to the history file upon release. File must be named as `<issue-number>.<type>` with type being:

   - `.feature`: For new features.
   - `.bugfix`: For bug fixes.
   - `.doc`: For documentation improvement.
   - `.removal`: For deprecation or removal of public API.
   - `.misc`: For general issues.

   Check towncrier documentation for more details.

5. The pull request should work for all python / django / django CMS versions declared in tox.ini. Check the CI and make sure that the tests pass for all supported versions.

5.1.7 Release a version

1. Update authors file
2. Merge `develop` on `master` branch
3. Bump release via task: `inv tag-release (major|minor|patch)`
4. Update changelog via towncrier: `towncrier --yes`
5. Commit changelog with `git commit --amend` to merge with bumpversion commit
6. Create tag `git tag <version>`
7. Push tag to github
8. Publish the release from the tags page
9. If pipeline succeeds, push `master`
10. Merge `master` back on `develop`
11. Bump developement version via task: `inv tag-dev -l (major|minor|patch)`
12. Push `develop`

CHAPTER SIX

CREDITS

6.1 Development Lead

- Iacopo Spalletti <i.spalletti@nephila.it>

6.2 Contributors

- Adam Chainz
- Daniel Rios
- Daniel Santos

CHAPTER

SEVEN

HISTORY

7.1 0.5.1 (2023-04-18)

7.1.1 Features

- Add support for django 4.2 (#22)

7.2 0.5.0 (2023-02-19)

7.2.1 Features

- Upgrade to Channels 4.0 (#19)
- Add support for Django 3.2 - 4.1

7.3 0.4.0 (2020-05-20)

- Migrate to Channels 2
- Add support for Django 2.2 / 3.0
- Drop support for Python 2
- Drop support for Django < 2.2

7.4 0.3.3 (2018-01-01)

- Fix support for newer channel versions
- Fix error in signal handling
- Add support for Django 1.11
- Improv test coverage

7.5 0.3.2 (2016-12-02)

- Add support for Django 1.10

7.6 0.3.1 (2016-09-10)

- Fix error in js message'

7.7 0.3.0 (2016-08-03)

- Make easier to customize the knocker url

7.8 0.2.0 (2016-06-11)

- Fix documentation
- Improv routing setting in tests

7.9 0.1.1 (2016-04-08)

- Add Add pause_knocks / active_knocks functions.

7.10 0.1.0 (2016-04-07)

- First release on PyPI.

INDEX