Django Active Link Documentation

Release 0.1.5

Valery Melou

Mar 23, 2019

Contents

Contents:

CHAPTER 1

Django Active Link

The simplest way to highlight active links in your Django app.

1.1 Documentation

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

1.2 Quick start

Install Django Active Link:

```
pip install django-active-link
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'active_link',
    ...
)
```

To use the `active_link` template tag you need to load `active_link_tags` templatetags library:

```
{% load active_link_tags %}
```

To add an `active` CSS class to a link when the request path matches a given view just do something like this.

```
<a href="{% url 'view-name' %}" class="{% active_link 'view-name' %}">Menu item</a>
```

If you has a sub-menu or tabs and needs they be active and parent too, you can use `||` to check this:

```
<a href="{% url 'view-name' %}" class="{% active_link 'view-name || view-sub-name' %}
↪">Menu Item</a>
<a href="{% url 'view-sub-name' %}" class="{% active_link 'view-sub-name' %}">Tab Item
↪</a>
```

You can also use a custom CSS class:

```
<a href="{% url 'view-name' %}" class="{% active_link 'view-name' 'custom-class' %}">
↪Menu item</a>
```

or:

```
<a href="{% url 'view-name' %}" class="{% active_link 'view-name' css_class='custom-
↪class' %}">Menu item</a>
```

By default `active_link` will not perform a strict match. If you want to add the `active` class only in case of a strict match pass the `strict` argument to the tag:

```
<a href="{% url 'view-name' %}" class="{% active_link strict=True %}">Menu item</a>
```

Replace `view-name` with the name of your view (including namespaces).

1.3 Settings

You can override the default active class and strict mode with the settings `ACTIVE_LINK_CSS_CLASS` and `ACTIVE_LINK_STRICT`.

Key	Description	Default Value
ACTIVE_LINK_CSS_CLASS	Active class to use.	*active*
ACTIVE_LINK_STRICT	Designates whether to perform a strict match or not.	*False*

For more usage examples, please check the full documentation at https://django-active-link.readthedocs.io.

IMPORTANT: Django Active Link requires that the current request object is available in your template's context. This means you must be using a *RequestContext* when rendering your template, and *django.core.context_processors.request* must be in your *TEMPLATE_CONTEXT_PROCESSORS* setting. See https://docs.djangoproject.com/en/dev/ref/templates/api/#subclassing-context-requestcontext for more information.

1.4 TODO

- Write the documentation

1.5 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.6 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-active-link
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-active-link
$ pip install django-active-link
```

Add *active_link* to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'active_link',
    ...
)
```

That's it. You can start using Django Active Link in your templates.

CHAPTER 3

Usage

To use Django Active Link in a project, add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'active_link',
    ...
)
```

IMPORTANT: Django Active Link requires that the current request object is available in your template's context. This means you must be using a *RequestContext* when rendering your template, and *django.core.context_processors.request* must be in your *TEMPLATE_CONTEXT_PROCESSORS* setting. See [the documentation](https://docs.djangoproject.com/en/dev/ref/templates/api/#subclassing-context-requestcontext) for more information.

To use the *active_link* template tag you need to load *active_link_tags* templatetags library:

    {% load active_link_tags %}

To add an *active* CSS class to a link when the request path matches a given view just do something like this.

    Menu item

If you has a sub-menu or tabs and needs they be active and your parent too, you can use `||` to check this:

    Menu Item Tab Item

Replace *view-name* with the name of your view (including namespaces).

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/valerymelou/django-active-link/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/valerymelou/django-active-link/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 :)

4.2 Get Started!

Ready to contribute? Here's how to set up *django-active-link* for local development.

1. Fork the *django-active-link* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-active-link.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-active-link
$ cd django-active-link/
$ python setup.py develop
```

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:

```
$ flake8 active_link tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them 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.

4.3 Pull Request Guidelines

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

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/valerymelou/django-active-link/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_active_link
```

CHAPTER 5

Credits

5.1 Development Lead

- Valery Melou <valerymelou@gmail.com>

5.2 Contributors

- Paulo Reis <paulovitin@gmail.com>

CHAPTER 6

History

6.1 0.1.0 (2017-07-10)

- First release on PyPI.