django-menus Documentation

Release 1.0

Krzysztof Dorosz

Sep 08, 2017

Contents

Contents:

CHAPTER 1

Installation

Getting a module

Grab it from source repository:

```
pip install -r git+git://github.com/cypreess/django-menus.git#egg=django-menus
```

`MENUS_CONF` setting

You should create `MENUS_CONF` setting in your settings.py. Point it as a python import path to a module that stores your project menus, eg.:

```
MENUS_CONF = "myproject.menu"
```

Enabling request template context processor

Menuing system needs to know which page is currently viewed. It reads url from `request` attribute of current context, therefore you need to add proper context processor.

Django brings a `TEMPLATE_CONTEXT_PROCESSORS` setting variable to define any set of context processors. To avoid hard-coding the default set of `TEMPLATE_CONTEXT_PROCESSORS` use following code:

```
from django.conf import global_settings

TEMPLATE_CONTEXT_PROCESSORS = global_settings.TEMPLATE_CONTEXT_PROCESSORS + (
    'django.core.context_processors.request',

    )
```

CHAPTER 2

Building menus

First of all you should make a file that is pointed by setting `MENUS_CONF`. You will put your menus code there.

Note: By the convention use just `menu.py` in root project structure (on the `settings.py` level).

Static menu

The most simple and fast type of menu is a static menu. This is just a list that is declared in global scope of file named after `MENUS_CONF`.

```
# In menu.py file

static_menu = [
        {   'name' : 'Test 1',
            'class' : 'test_1',
            'url' : 'test1/',
            'match' : r'^test1/$',
            'sub' : None,
        },
        {   'name' : 'Test 2',
            'class' : 'test_2',
            'url' : 'test2/',
            'match' : r'^test2/$',
            'sub' : None,
        },

]
```

Warning: In static menu you cannot use `reverse()` function for dry links.

Dynamic menu

Menu can be also represented by any callable. The simplest is using a function:

```
# In menu.py file

def menu(context, variables):
    return [
       {   'name' : 'News',
           'class' : 'news_menu_item',
           'url' : reverse('news'),
           'match' : r'^'+ reverse('news') + r'$',
           'sub' : None,
       },

    ]
```

this allows to use some additional information like template context.

CHAPTER 3

Using menus in templates

django-menus provides only one template tag.

Loading template tags

Remember to load `menus` in every template you use menuing.

```
{% load menus %}
```

`get_menu` template tag

The template tag has following syntax:

```
{% get_menu MENU_NAME [as VARIABLE_NAME] [with VAR1=VAL1 VAR2=VAL2] %}
```

It propagates a menu data structure to the `VARIABLE_NAME`.

Note: If `VARIABLE_NAME` is omitted `MENU_NAME` is used.

Accessing and displaying menus

django-menus support multilevel menus

CHAPTER 4

Advanced usages

DRY patterns

To avoid hard-coding link patterns and keeping them DRY it is possible to use django standard `reverse` function.

Consider an url in urls.py:

```
url(r'^news/$', TemplateView.as_view(template_name="news.html"), name='news')
```

it can be easily provided to menu declaration:

```
def menu(context, variables):
    return [
       {   'name' : 'News',
           'class' : 'news_menu_item',
           'url' : reverse('news'),
           'match' : r'^'+ reverse('news') + r'$',
           'sub' : None,
       },

    ]
```

Note: Attribute `match` can be omitted in most cases. It defaults to ‘‘ r’^’+ url + r’$’,‘‘. This is extremely helpful in simple cases where exact match is what you want for menu element selection.

Internationalization

django-menus support internationalization using django `ugettext`.

```
from django.utils.translation import ugettext_lazy as _

def menu(context, variables):
    return [
       {   'name' : _('News'),
           'class' : 'news_menu_item',
           'url' : reverse('news'),
       },

    ]
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search