django-simple-menu Documentation

Release 1.2.1

Evan Borgstrom

Sep 26, 2017

Contents

django-simple-menu allows you to define multiple menus using straight forward python code in each of your installed apps.

The menus you define can be simple & straight forward with a static title and url or they can change based on the current request to support dynamic titles or complex permissions.

Contents:

CHAPTER 1

Installing django-simple-menu

1. Install `django-simple-menu` using pip:

```
pip install django-simple-menu
```

2. Add `menu` to your `INSTALLED_APPS` list in your settings
3. `django-simple-menu` requires that the `request` object be available in the context when you call the `{% generate_menu %}` template tag. This means that you need to ensure that your `TEMPLATE_CONTEXT_PROCESSORS` setting includes `django.core.context_processors.request`, which it doesn't by default.

CHAPTER 2

Configuring django-simple-menu

`django-simple-menu` has some configuration options that can be configured in your main Django settings file.

MENU_SELECT_PARENTS

Default: ‘‘False‘‘

`MENU_SELECT_PARENTS` controls if parent menu items should automatically have their `selected` property set to `True` if one of their children has its `selected` property set to `True`.

MENU_HIDE_EMPTY

Default: ‘‘False‘‘

`MENU_HIDE_EMPTY` controls if menu items without an explicit `check` callback should be visible even if they have no children

CHAPTER 3

Using django-simple-menu

Usage Overview

`django-simple-menu` lets you define multiple different menus that you can access from within your templates. This way you can have a menu for your main navigation, another menu for logged in users, another menu for anonymous users, etc.

Note: Since we use the `menu` namespace you will get `ImportError` if you have any files named `menu.py`, ensure you use a plural version: `menus.py`

To define your menus you need to create a file named `menus.py` inside of the app that you wish to hook menus up to. In the `menus.py` file you should import the `Menu` and `MenuItem` classes from the `menu` package:

```
from menu import Menu, MenuItem
```

The `Menu` class exposes a class method named `add_item` that accepts two arguments; the menu name you want to add to, and the `MenuItem` you're going to add.

The `MenuItem` class should be instantiated and passed to the `add_item` class method with the appropriate parameters. `MenuItem` accepts a wide number of options to its constructor method, the majority of which are simply attributes that become available in your templates when you're rendering out the menus. The required arguments to MenuItem are the first two; the title of the menu and the URL, and the keywords that affect menu generation are:

- The `weight` keyword argument affects sorting of the menu.
- The `children` keyword argument is either a list of `MenuItem` objects, or a callable which accepts the request object and returns a list.
- The `check` keyword argument is a callable that accepts the request object and returns `True` or `False` if the `MenuItem` should be visible for this request

Additional kwargs can be passed to `MenuItem` and these will become available in your templates. For example adding `separator=True` could be used to add separators to menus `{% if item.separator %}<li class="divider"></li>{% endif %}`

For the full list of `MenuItem` options see the menu __init__.py source file.

Usage Example

Example:

```
# Add two items to our main menu
Menu.add_item("main", MenuItem("Tools",
                               reverse("myapp.views.tools"),
                               weight=10,
                               icon="tools"))

Menu.add_item("main", MenuItem("Reports",
                               reverse("myapp.views.reports"),
                               weight=20,
                               icon="report"))


# Define children for the my account menu
myaccount_children = (
    MenuItem("Edit Profile",
             reverse("accounts.views.editprofile"),
             weight=10,
             icon="user"),
    MenuItem("Admin",
             reverse("admin:index"),
             weight=80,
             separator=True,
             check=lambda request: request.user.is_superuser),
    MenuItem("Logout",
             reverse("accounts.views.logout"),
             weight=90,
             separator=True,
             icon="user"),
)

# Add a My Account item to our user menu
Menu.add_item("user", MenuItem("My Account",
                               reverse("accounts.views.myaccount"),
                               weight=10,
                               children=myaccount_children))
```

Once you have your menus defined you need to incorporate them into your templates. This is done through the `generate_menu` template tag:

```
{% extends "base.html" %}
{% load menu %}

{% block content %}
{% generate_menu %}
...
{% endblock %}
```

Note that `generate_menu` must be called inside of a block.

Once you call `generate_menu` all of your MenuItems will be evaluated and the following items will be set in the context for you.

1. `menus` - This is an object that contains all of the lists of menus as attribute names:

```
{% for item in menus.user %} ... {% endfor %}
```

2. `selected_menu` - This is the `MenuItem` object of the most specific URL match.
3. `submenu` - This is the submenu object of the most specific URL match.
4. `has_submenu` - This is `True` or `False` if the selected menu has children.

See the bootstrap-navbar.html file in the templates dir of the source code for an example that renders menus for the Twitter Bootstrap Navbar Component. You can use it like:

```
{% with menu=menus.main %}{% include "bootstrap-navbar.html" %}{% endwith %}
```

Check generalizations

If your application is dynamic enough, or complex enough, you may find that you want to generalize your check logic based on a permissions model, or something similar. To accomplish this you can create your own custom `MenuItem` implementation with a `check` method.

This assumes you have a `utils` package.

`utils/menus.py`:

```
from django.core.urlresolvers import resolve

from menu import MenuItem


class ViewMenuItem(MenuItem):
    """Custom MenuItem that checks permissions based on the view associated
    with a URL"""

    def check(self, request):
        """Check permissions based on our view"""
        is_visible = True
        match = resolve(self.url)

        # do something with match, and possibly change is_visible...

        self.visible = is_visible
```

`reports/menus.py`:

```
from utils.menus import ViewMenuItem

from menu import Menu, MenuItem

from django.core.urlresolvers import reverse

# Since we use ViewMenuItem here we do not need to define checks, instead
# the check logic will change their visibility based on the permissions
# attached to the views we reverse here.
reports_children = (
    ViewMenuItem("Staff Only", reverse("reports.views.staff")),
    ViewMenuItem("Superuser Only", reverse("reports.views.superuser"))
```

```
)

Menu.add_item("main", MenuItem("Reports Index",
                               reverse("reports.views.index"),
                               children=reports_children))
```

CHAPTER 4

Example Project

If you're looking for a quick way to evaluate django-simple-menu there's an example project located in the github source repository that can be quickly setup to learn and tinker with the menu system. Simply clone the github repo to your local development workspace and then follow the README file located in the "example" folder of the sources.