Django Menu Generator Documentation

Release 1.0.3

Milton Lenis

Oct 17, 2018

Contents

Django Menu Generator is a productivity tool that enables the generation of full featured menus through python dictionaries list, you only need to setup the HTML structure once for each menu you like to build and then use the dictionaries to generate menu items

CHAPTER 1

Features:

- Tested support to Python 3.4, 3.5, 3.6
- Tested support to Django 1.8, 1.9, 1.10, 1.11, 2.0
- No database
- Support unlimited menus
- Icons support
- Semi-Automatically identifies the selected item and his breadcrums
- Conditional menu items display through validators (Permissions, Authentications or whatever you want)

CHAPTER 2

Contents:

2.1 Installation

You can install Django Menu Generator with one of these options:

- `easy_install django-menu-generator`
- `pip install django-menu-generator`
- `git clone https://github.com/LaLogiaDePython/django-menu-generator`
  1. `cd django-menu-generator`
  2. run python setup.py
- `wget https://github.com/LaLogiaDePython/django-menu-generator/zipball/master`
  1. unzip the downloaded file
  2. cd into django-menu-generator-* directory
  3. run python setup.py

2.2 Usage

1. Once installed, add `'menu_generator'` to your INSTALLED_APPS.
2. Add `{% load menu_generator %}` to templates that will handle the menus.
3. Add the list dictionaries containing the menu items to the settings

```
################################################################################
Example: settings.py
################################################################################
```

(continues on next page)

(continued from previous page)

```
NAV_MENU_LEFT = [
    {
        "name": "Home",
        "url": "/",
    },
    {
        "name": "About",
        "url": "/about",
    },
]

NAV_MENU_RIGHT = [
    {
        "name": "Login",
        "url": "login_url_view",  # reversible
        "validators": ["menu_generator.validators.is_anonymous"],
    },
    {
        "name": "Register",
        "url": "register_view_url",  # reversible
        "validators": ["menu_generator.validators.is_anonymous"],
    },
    {
        "name": "Account",
        "url": "/acount",
        "validators": ["menu_generator.validators.is_authenticated"],
        "submenu": [
            {
                "name": "Profile",
                "url": "/account/profile",
            },
            {
                "name": "Account Balance",
                "url": "/account/balance",
                "validators": ["myapp.profiles.is_paid_user"],
            },
            {
                "name": "Account Secrets",
                "url": "/account/secrets",
                "validators": ["menu_generator.validators.is_superuser"],
            }
        ],
    },
]

FOOTER_MENU_LEFT = [
    {
        "name": "Facebook",
        "url": "facebook.com/foobar",
    },
    {
        "name": "Contact US",
        "url": "/contact",
    },
]

FOOTER_MENU_RIGHT = [
```

(continues on next page)

(continued from previous page)

```
    {
        "name": "Address",
        "url": "/address",
    },
]
```

Or you can build the menu dictionaries list inside the project apps with `menus.py` files, see *Menu Generation* for more.

4. In your template, load the template tag to generate your menu.

```
{% load menu_generator %}
<!DOCTYPE html>
<html>
    <head><title>Django Menu Generator</title></head>
    <body>
        <!-- NAV BAR Start -->
        {% get_menu "NAV_MENU_LEFT" as left_menu %}
        <div style="float:left;">
            {% for item in left_menu %}
                <li class="{% if item.selected %} active {% endif %}">
                <a href="{{ item.url }}"> <i class="{{ item.icon_class }}"></i> {{␣
↪item.name }}</a>
                </li>
                {% if item.submenu %}
                    <ul>
                    {% for menu in item.submenu %}
                        <li class="{% if menu.selected %} active {% endif %}">
                            <a href="{{ menu.url }}">{{ menu.name }}</a>
                        </li>
                    {% endfor %}
                    </ul>
                {% endif %}
            {% endfor %}
        </div>

        {% get_menu "NAV_MENU_RIGHT" as right_menu %}
        <div style="float:right;">
            {% for item in right_menu %}
                <li class="{% if item.selected %} active {% endif %}">
                    <a href="{{ item.url }}">{{ item.name }}</a>
                </li>
                {% if item.submenu %}
                    <ul>
                    {% for menu in item.submenu %}
                        <li class="{% if menu.selected %} active {% endif %}">
                            <a href="{{ menu.url }}">{{ menu.name }}</a>
                        </li>
                    {% endfor %}
                    </ul>
                {% endif %}
            {% endfor %}
        </div>
        <!-- NAV BAR End -->

        <!-- Footer Start -->
        {% get_menu "FOOTER_MENU_LEFT" as left_footer_menu %}
```

(continues on next page)

(continued from previous page)

```
        <div style="float:left;">
            <!-- loop through your left footer menus -->
        </div>

        {% get_menu "FOOTER_MENU_RIGHT" as right_footer_menu %}
        <div style="float:right;">
            <!-- loop through your right footer menus -->
        </div>
        <!-- Footer End -->
    </body>
</html>
```

5. Now you must to see your menus generated when you run your project

2.3 Menu Generation

Django Menu Generator uses python dictionaries to represent the menu items, usually a menu item is as follows:

```
{
    "name": 'some name',
    "icon_class": 'some icon class',
    "url": URL spec,
    "root": True | False,
    "validators": [ list of validators ],
    "submenu": Dictionary like this
}
```

Where each key is as follows:

- `name`: A string representing the label of the menu item. If you are using i18n here you can pass the name with the `ugettext_lazy` function
- `icon_class`: A string representing the class of the icon you wish to show on the menu item, e.g you can use font-awesome
- `url`: See *URLs*
- `root`: A flag to indicate this item is the root of a path, with this you can correctly mark nested menus as selected.
- `validators`: See *Validators*
- `submenu`: You can create infinite nested submenus passing here menu items like this

Django Menu Generator offers two ways to generate the menus, through the Django settings and through each of the Django apps

2.3.1 Generating menus through settings

You can add various list dictionaries representing each menu you have as explained in *Usage* We recommend to have a `menus.py` file with the menu list dictionaries and then import it to the settings file if you go this way

2.3.2 Generating menus through apps

Some people prefer to isolate all the aspects of a project between apps, so, we add this feature to allow the menus live inside each app.

You need to add inside the app a `menus.py` file that contains a dictionary called `MENUS`, each element of the dictionary will be a menu list dictionary with all the configuration needed to display that menu, e.g:

```
MENUS = {
    'NAV_MENU_LEFT': [
        {
            "name": "App1 Feature",
            "url": "/app1-feature"
        }
    ],
    'NAV_MENU_TOP': [
        {
            "name": "Second Menu Feature",
            "url": "named_url"
        }
    ]
}
```

So, as an example, for the `'NAV_MENU_LEFT'`, Django Menu Generator will loop each app searching for the `'NAV_MENU_LEFT'` list dictionaries inside of the `MENUS` and build all the menu configuration to build the whole menu.

With this feature you can have a project structure like this:

```
your_project/
├── config_folder/
│   └── ...
├── app1
│   └── models.py
│       forms.py
│       views.py
│       menus.py
│
├── app2
│   └── models.py
│       forms.py
│       views.py
│       menus.py
│
 ...
```

You can have a mix of the two approaches if you wish

2.4 URLs

You can pass the URL parameters to menu items in three ways.

2.4.1 Raw URLs

A hard-coded url:

```
"url": '/some-path/to-feature'
```

2.4.2 Reversible URLs

An url that can be reversed with the *reverse* method:

```
"url": 'named_url'
```

2.4.3 URL with args or kwargs

e.g. If you have an url with kwargs like this:

```
url(r'^update/(?P<pk>\d+)/$', SomeView.as_view(), name='update'),
```

you can pass the url as follows:

```
"url": {"viewname": 'update', "kwargs": {"pk": 1}}
```

In fact, you can pass any of the parameters of the reverse method through the dictionary

For Django 1.10 the reverse method sign is: `reverse(viewname, urlconf=None, args=None, kwargs=None, current_app=None)`

2.5 Validators

Django Menu Generator uses validators to allow the displaying of menu items.

A validator is a function that receives the request as arg and returns a boolean indicating if the check has passed

for Django Menu Generator the validators must always be a list containing at least one callable or python path to a callable. If there is more than one validator, all of them will be evaluated using the AND connector.

2.5.1 Built-in validators

Django Menu Generator has the following built-in validators:

- is_superuser:

  A validator to check if the authenticated user is a superuser

  Usage:

  ```
  "validators": ['menu_generator.validators.is_superuser']
  ```

- is_staff:

  A validator to check if the authenticated user is member of the staff

  Usage:

  ```
  "validators": ['menu_generator.validators.is_staff']
  ```

- is_authenticated:

A validator to check if user is authenticated

Usage:

```
"validators": ['menu_generator.validators.is_authenticated']
```

- is_anonymous:

  A validator to check if the user is not authenticated

  Usage:

```
"validators": ['menu_generator.validators.is_anonymous']
```

- user_has_permission:

  A validator to check if the user has the given permission

  Usage:

```
"validators": [
    ('menu_generator.validators.user_has_permission', 'app_label.
↪permission_codename')
]
```

- More than one validator:

  You can pass more than one validator to evaluate using the AND connector

```
"validators": [
    'menu_generator.validators.is_staff',
    ('menu_generator.validators.user_has_permission', 'some_app.some_
↪permission')
    ...
]
```

2.5.2 Custom validators

You can build your own validators and use them with Django Menu Generator

Let's build a validator that checks if the user have more than one pet (dummy example) assuming the user has a many to many relation called pets

Assuming we build the function inside `your_project/app1` on a `menu_validators.py` we have:

```
# Remember you always must to past the request as first parameter
def has_more_than_one_pet(request):

    return request.user.pets.count() > 0
```

So we can use it as a validator

```
"validators": ['your_project.app1.menu_validators.has_more_than_one_pet']
```

Now let's build a validator that checks if the user's pet belongs to a specific type to illustrate the validators with parameters.

Assuming we build the function inside the same path and the user have a foreign key called pet

```
def has_a_pet_of_type(request, type):

    return request.user.pet.type == type
```

So we use the validator like this:

```
"validators": [
    ('your_project.app1.menu_validators.has_a_pet_of_type', 'DOG')
]
```

As you can see, we use tuples to pass parameters to the validators, where the first position is the validator and the rest are the function parameters

2.6 CHANGELOG

2.6.1 1.0.4(2018-02-19)

- **Features:**
  - Adding support for AppConfig in INSTALLED_APPS

2.6.2 1.0.3(2018-01-31)

- **Project Enhancements:**
  - Dropping support for Python 2
  - Updating docs
  - Readme updating
  - Support for Django 2.0 (Thanks to @lachmanfrantisek)
- **Features:**
  - Adding support to root paths (Thanks to @lucaskuzma)
- **BugFix:**
  - Fixing backward compatibility with Django < 1.10

2.6.3 1.0.2(2017-04-29)

- Updating .gitignore to ignore sphinx builds
- Updating docs

2.6.4 1.0.1(2017-04-29)

- Added docs
- Readme enhanced
- BUGFIX: Added a correction to the validators evaluation with the AND connector
- Added flake8 to CI

- Added tox

2.6.5 1.0.0 (2017-04-09)

- Initial release

2.7 Running the tests

To run the tests against configured environments:

```
tox
```

2.8 Authors

Milton Lenis - miltonln04@gmail.com

Juan Diego García - juandgoc@gmail.com

2.9 Contribute

- Issue tracker: https://github.com/LaLogiaDePython/django-menu-generator/issues
- Source code: https://github.com/LaLogiaDePython/django-menu-generator

2.10 Support

If you are having issues, please let us know. Contact us at: miltonln04@gmail.com

2.11 License

Released under a (MIT) license.

2.12 Credits

We would like to thank Val Kneeman, the original author of this project under the name 'menuware' https://github.com/un33k/django-menuware