django-sitecats Documentation

Release 1.2.2

Igor 'idle sign' Starikov

Dec 18, 2021

Contents

https://github.com/idlesign/django-sitecats

CHAPTER 1

Description

Django reusable application for content categorization.

Nay, - you say, - all that tags business lacks structuring.

This application is just about structuring your data: build categories hierarchy and link your site entities to those categories.

CHAPTER 2

Requirements

1. Python 3.6+
2. Django 2.0+
3. Django Auth contrib enabled
4. Django Admin contrib enabled (optional)
5. `django-admirarchy` for Django Admin contrib (optional)
6. Django Messages contrib enabled (optional)
7. jQuery (for client side additional functionality; optional)

CHAPTER 3

Table of Contents

3.1 Quickstart

Note: Add the **sitecats** application to INSTALLED_APPS in your settings file (usually 'settings.py').

Let's allow categorization for `Article` model.

1. First, inherit that model from **sitecats.models.ModelWithCategory**.
2. Then we need to add at least one root category, to show up in our categories editor.

   Let's create two root categories with *language* and *os* aliases. This could be done either programmatically (instantiate *Category* model) or with Django Admin Contrib (install django-admirarchy to simplify navigation).
3. Now to our views:

```
# Somewhere in views.py
from django.shortcuts import render, get_object_or_404

# `Article` model has sitecats.models.ModelWithCategory class mixed in.
from .models import Article


def article_details(self, request, article_id):
    """Nothing special in this view, yet it'll render a page with categories for our␣
↪articles."""

    article = get_object_or_404(Article, pk=article_id)

    # Let's configure our category lists a little bit:
    # * show titles of parent categories,
    # * apply Twitter Bootstrap 3 css classes to categories.
    article.set_category_lists_init_kwargs({
        'show_title': True,
```

(continues on next page)

(continued from previous page)

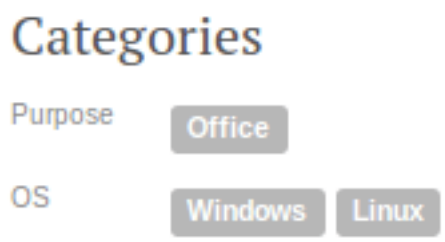
```
        'cat_html_class': 'label label-default'
    })

    return self.render(request, 'article.html', {'article': article})

def article_edit(self, request, article_id):
    """This time we allow the view to render and handle categories editor."""
    article = get_object_or_404(Article, pk=article_id)

    # Now we enable category editor for an article, and allow users
    # to add subcategories of `language`, and `os` categories and link articles to␣
↪them.
    article.enable_category_lists_editor(
        request,
        editor_init_kwargs={'allow_new': True},
        lists_init_kwargs={
            'show_title': True,
            'cat_html_class': 'label label-default'
        },
        additional_parents_aliases=['language', 'os']
    )

    form = ... # Your usual Article edit handling code will be here.

    return render(request, 'article.html', {'article': article, 'form': form})
```

4. Template coding basically boils down to `sitecats_categories` template tags usage:

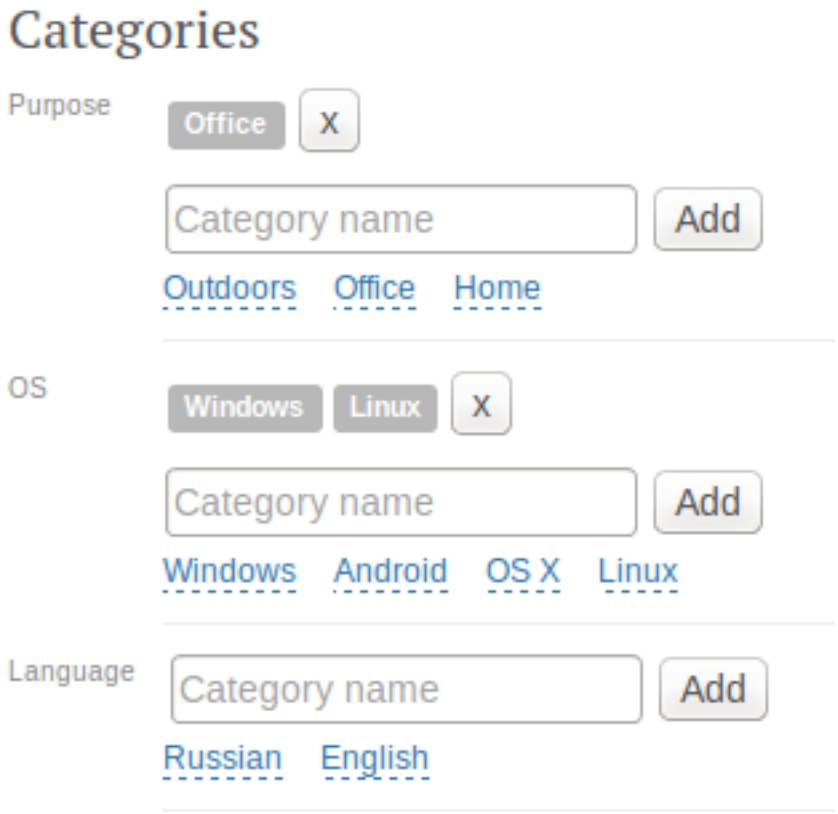
```
<!-- article.html
     The same html is just fine for demonstration purposes for both our views.
     Do not forget to load `sitecats` template tags library. -->
{% extends "base.html" %}
{% load sitecats %}

{% block contents %}
    <!-- Some additional functionality (e.g. categories cloud rendering,
         editor enhancements) will require JS. -->
    <script src="{{ STATIC_URL }}js/sitecats/sitecats.min.js"></script>

    <h1>{{ article.title }}</h1>
    <div id="article_categories">
        {% sitecats_categories from article %} <!-- And that's it. -->
    </div>
    <!-- Form code goes somewhere here. -->
{% endblock %}
```

Add some categories.

That's roughly what we could have on details page:



And on edit page:



3.2 Toolbox

Here are most important tools exposed by `sitecats`.

3.2.1 Settings

You can override the following settings for your project:

- **SITECATS_MODEL_CATEGORY** - Path to a model to be used as a Category (e.g. *myapp.MyCategory*).
- **SITECATS_MODEL_TIE** - Path to a model to be used as a category-to-object Tie (e.g. *myapp.MyTie*).

3.2.2 toolbox.get_category_model

Returns the Category model, set for the project.

Defaults to *models.Category*, can be customized by subclassing *models.CategoryBase*.

3.2.3 toolbox.get_tie_model

Returns the Tie model, set for the project.

Defaults to *models.Tie*, can be customized by subclassing *models.TieBase*.

3.2.4 models.TieBase

Base class for ties models.

Ties are relations between site entities and categories (see above).

Inherit from this model and override **SITECATS_MODEL_TIE** in *settings.py* to customize model fields and behaviour.

You can get tie model with *get_tie_model*.

Whether you need to know categories your site items are currently linked to alongside with ties themselves you can use *get_linked_objects* method.

get_linked_objects(cls, filter_kwargs=None, id_only=False, by_category=False):
Returns objects linked to categories in a dictionary indexed by model classes.

Parameters

- **filter_kwargs** (*dict*) – Filter for ties.
- **id_only** (*bool*) – If True only IDs of linked objects are returned, otherwise - QuerySets.
- **by_category** (*bool*) – If True only linked objects and their models a grouped by categories.

3.2.5 models.ModelWithCategory

Helper class for models with tags.

Mix in this helper to your model class to be able to categorize model instances.

set_category_lists_init_kwargs(self, kwa_dict):
Sets keyword arguments for category lists which can be spawned by get_categories().

Parameters **kwa_dict** (*dict|None*) –

enable_category_lists_editor(self, request, editor_init_kwargs=None, additional_parents_al
lists_init_kwargs=None, handler_init_kwargs=None):
Enables editor functionality for categories of this object.

Parameters

- **request** (*Request*) – Django request object
- **editor_init_kwargs** (*dict*) – Keyword args to initialize category lists editor with. See CategoryList.enable_editor()

- **additional_parents_aliases** (*list*) – Aliases of categories for editor to render even if this object has no tie to them.
- **lists_init_kwargs** (*dict*) – Keyword args to initialize CategoryList objects with
- **handler_init_kwargs** (*dict*) – Keyword args to initialize CategoryRequestHandler object with

add_to_category(*self*, *category*, *user*)
Add this model instance to a category.

E.g: my_article.add_to_category(category_one, request.user).

Parameters

- **category** (*Category*) – Category to add this object to
- **user** (*User*) – User heir who adds

remove_from_category(self, category):
Removes this object from a given category.

E.g: my_article.remove_from_category(category_one).

Parameters **category** (*Category*) –

get_ties_for_categories_qs(cls, categories, user=None, status=None):
Returns a QuerySet of Ties for the given categories.

E.g: Article.get_ties_for_categories_qs([category_one, category_two]).

Parameters

- **categories** (*list|Category*) –
- **user** (*User|None*) –
- **status** (*int|None*) –

get_from_category_qs(cls, category):
Returns a QuerySet of objects of this type associated with the given category.

E.g: Article.get_from_category_qs(my_category).

Parameters **category** (*Category*) –

3.2.6 toolbox.get_category_lists

get_category_lists(init_kwargs=None, additional_parents_aliases=None, obj=None):
Returns a list of CategoryList objects, optionally associated with a given model instance.

Parameters

- **init_kwargs** (*dict|None*) –
- **additional_parents_aliases** (*list|None*) –
- **obj** (*Model|None*) – Model instance to get categories for

Return type list

3.2.7 toolbox.get_category_aliases_under

get_category_aliases_under(parent_alias=None):

Returns a list of category aliases under the given parent.

Could be useful to pass to *ModelWithCategory.enable_category_lists_editor* in *additional_parents_aliases* parameter.

Parameters **parent_alias** (*str|None*) – Parent alias or None to categories under root

Return type list

3.3 Usage examples

Let's suppose we have a simple two-level categories hierarchy:

```
|-- Language (alias: langs)
|   |-- Russian
|   |-- English
|
|-- OS (alias: os)
|   |-- Linux
|   |-- Android
|   |-- OS X
|
|-- Direction (alias: dir)
    |-- North
    |-- South
    |-- East
    |-- West
```

And we have two models inherited from *ModelWithCategory*: *Article* and *Comment*.

Now to what we can do with it.

3.3.1 Get categories for a certain object

```
def view_article_details(request, article_id):

    article = get_object_or_404(Article, pk=article_id)

    # Setting init keyword arguments for for every CategoryList.
    #
    # E.g.: if this article is in the `Linux` and `English` categories,
    # two CategoryLists objects will be created:
    # one for `OS` and the other for `Language` parent categories.
    #
    # With `show_title`=True `{% sitecats_categories from article %}` template tag
    # will render parent categories titles alongside with `Linux` and `English`␣
↪categories.
    article.set_category_lists_init_kwargs({'show_title': True})

    ...

    return ...
```

3.3.2 Get category editor for a certain object

```
from sitecats.toolbox import get_category_aliases_under

def view_article_edit(request, article_id):

    article = get_object_or_404(Article, pk=article_id)

    ...

    article.enable_category_lists_editor(request,
        # By default editor will render only parent categories
        # for children associated with this article.
        # If we want other parent categories to be available, we should
        # give the editor their aliases with `additional_parents_aliases`.
        #
        # Here we use `get_category_aliases_under()` helper without arguments
        # to get aliases of root categories (Language, OS, Direction).
        additional_parents_aliases=get_category_aliases_under(),

        # Setting up editor parameters.
        # Here `allow_add` allows adding this article into existing categories;
        # `allow_remove` allows removing this article from associated categories;
        # `allow_new` allows superusers to create new subcategories and add this␣
↪articles into them.
        editor_init_kwargs={'allow_add': True, 'allow_remove': True, 'allow_new':␣
↪request.user.is_superuser}

        # Setting up category editor requests handler.
        # Passing extra tags for error messages generated by with `error_messages_
↪extra_tags`
        # for error messages styling purposes.
        handler_init_kwargs={'error_messages_extra_tags': 'alert alert-danger'},

        # Setting init keyword arguments for for every CategoryList.
        lists_init_kwargs={'show_title': True},
    )

    # Now {% sitecats_categories from article %}` template tag will render
    # a category editor with set properties.

    ...

    return ...
```

3.3.3 Get categories having associated objects

```
from sitecats.toolbox import get_category_lists, get_category_aliases_under

def view_categories(request):
    lists = get_category_lists(
        # We need to render categories under root parents (Language, OS, Direction).
        additional_parents_aliases=get_category_aliases_under(),
        init_kwargs={
            # We'll show parent categories titles.
```

(continues on next page)

(continued from previous page)

```
            'show_title': True,
            # We'll create links for category details pages.
            # Let's suppose we have `category-details` URL pattern defined.
            'show_links': lambda cat: reverse('category-details', args=[cat.id])
        })

    # Now {% sitecats_categories from article %}` template tag will render
    # all categories having associated objects.

    return ...
```

3.3.4 Get model instances associated with a category

```
def view_articles_for_category(request, category_id):

    ...

    objects = Article.get_from_category_qs(category)

    ...
```

3.3.5 Get all ties by categories

```
from sitecats.toolbox import get_tie_model

linked_objects = get_tie_model().get_linked_objects(by_category=True)
```

3.4 Additional functionality via JS

sitecats bundles JS providing some additional functionality.

Do not forget to include JS file in your HTML to use it:

```
<script src="{{ STATIC_URL }}js/sitecats/sitecats.min.js"></script>
```

Note: jQuery is required for **sitecats** client side functionality.

3.4.1 Rendering categories cloud

To convert your categories list into a cloud where category title font size depends upon a number of items belonging to this category, use *make_cloud()* method.

```
// `article_categories` is an ID of HTML element containing categories lists.
sitecats.make_cloud('article_categories');
```

CHAPTER 4

Get involved into django-sitecats

Submit issues. If you spotted something weird in application behavior or want to propose a feature you can do that at https://github.com/idlesign/django-sitecats/issues

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/django-sitecats, write your code, whether it should be a bugfix or a feature implementation, and make a pull request right from the forked project page.

Translate. If want to translate the application into your native language use Transifex: https://www.transifex.com/projects/p/django-sitecats/.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish them.

CHAPTER 5

Also

If the application is not what you want for content categorization / tagging, you might be interested in considering other choices — https://www.djangopackages.com/grids/g/tagging/

Index

A