django-cms-search Documentation

Release 0.6.2

Divio GmbH

February 04, 2016

Contents

This package provides multilingual search indexes for easy Haystack integration with django CMS.

CHAPTER 1

Requirements

- Django >= 1.3
- django CMS >= 2.1.3
- django-haystack >= 1.2.4, < 2.0 (support for 2.0 will be a priority once it is released)
- django-classy-tags >= 0.3.2

CHAPTER 2

Usage

After installing django-cms-search through your package manager of choice, add `cms_search` to your `INSTALLED_APPS`. That's it.

Warning: Since version 0.5, the `HaystackSearchApphook` is not registered automatically anymore. If you want do use the default app hook provided by django-cms-search, add this (e.g. in `models.py`):

```
from cms_search.cms_app import HaystackSearchApphook
apphook_pool.register(HaystackSearchApphook)
```

For setting up Haystack, please refer to their documentation.

2.1 Customizing the Index

You can customize what parts of a `CMSPlugin` end up in the index with two class attributes on `CMSPluginBase` subclasses:

`search_fields`
a list of field names to index.

`search_fulltext`
if `True`, the index renders the plugin and adds the result (sans HTML tags) to the index.

Note: Before version 0.6 of this library, this attributes had to be defined on `CMSPlugin`. Starting with 0.6, it can also be defined on `CMSPluginBase`, for cases where one `CMSPlugin` subclass is used by multiple `CMSPluginBase` subclasses with different search requirements.

CHAPTER 3

Helpers

django-cms-search provides a couple of useful helpers to deal with multilingual content.

class `cms_search.search_helpers.indexes.`**`MultiLanguageIndex`**

A `SearchIndex` that dynamically adds translated fields to the search index. An example for when this is useful is the app hook infrastructure from django CMS. When a model's `get_absolute_url()` uses a url pattern that is attached to an app hook, the URL varies depending on the language. A usage example:

```
from haystack import indexes
from cms_search.search_helpers.indexes import MultiLanguageIndex

class NewsIndex(MultiLanguageIndex):
    text = indexes.CharField(document=True, use_template=True)
    title = indexes.CharField(model_attr='title')
    url = indexes.CharField(stored=True)

    def prepare_url(self, obj):
        return obj.get_absolute_url()

    class HaystackTrans:
        fields = ('url', 'title')
```

Note:

- *`MultiLanguageIndex`* dynamically creates translated fields. The name of the dynamic fields is a concatenation of the original field name, an underscore and the language code.
- If you define a `prepare()` method for a translated field, that method will be called multiple times, with changing active language.
- In the above example, you might want to catch `NoReverseMatch` exceptions if you don't have activated the app hook for all languages defined in `LANGUAGES`.
- The `model_attr` attribute is handled somewhat specially. The index tries to find a field on the model called `model_attr + '_' + language_code`. If it exists, it is used as the translated value. But it isn't possible to supply the name of a model method and let the index call it with varying activated languages. Use `prepare_myfieldname()` for that case.

Note: django CMS monkeypatches `django.core.urlresolvers.reverse()` to enable language namespaces. To ensure that this monkeypatching happens before haystack autodiscovers your indexes, your `search_sites.py` should look somewhat like this:

```
    from cms.models import monkeypatch_reverse
    import haystack

    monkeypatch_reverse()
    haystack.autodiscover()
```

class cms_search.search_helpers.fields.**MultiLangTemplateField**
A haystack.indexes.CharField subclass that renders the search template in all languages defined in LANGUAGES and concatenates the result.

Note: If you plan to render django CMS placeholders in the template, make sure to pass the needs_request argument to *cms_search.search_helpers.fields.MultiLangTemplateField()*.

3.1 {% get_translated_value %} template tag

This template tag is most useful in combination with the *MultiLanguageIndex*. You can use it while looping through search results, and it will automatically pick up the translated field for the current language or fall back to another available language (in the order defined in LANGUAGES). Example:

```
{% load cms_search_tags %}

<ul class="search-results">
    {% for result in page.object_list %}
        <li><a href="{% get_translated_value result "url" %}">{% get_translated_value result "title"
    {% endfor %}
</ul>
```

Note: If you plan to use this template tag, you have to add *cms_search.search_helpers* to your INSTALLED_APPS.

CHAPTER 4

Settings

4.1 CMS_SEARCH_INDEX_BASE_CLASS

Default: `haystack.indexes.SearchIndex`

This setting can be used to add custom fields to the search index if the included fields do not suffice. Make sure to provide the full path to your `SearchIndex` subclass.

C

C

G

S

T