Tony Jiang

Sheltr Haystack and Solr Notes 6/3/2015

Django 1.8.2, Solr 4.9.1, Haystack 2.4

After spending hours on top of hours trying to figure out how to utilize Haystack, this is the information I’ve garnered and pieced together from mass googling and many re-reads of the Haystack docs while trying to **implement more advanced/custom search** (not using the getting started example in the haystacks docs that just provides basic search functionality).

The page in the haystack docs you want to study to add custom search is <http://django-haystack.readthedocs.org/en/latest/views_and_forms.html>

We start with urls.py:

In the example on the Haystack docs, the URL for the search page is:

url(r'^search/', include('haystack.urls')),

If you look into the haystack library in its urls.py file, you’ll find that it’s basically including this:

patterns('haystack.views',

url(r'^$', SearchView(), name='haystack\_search'),

Which means that it is using SearchView, the most basic search view built into haystack that provides the search functionality you see in the haystack getting started example. You can take a look at it in haystack’s views.py file.

There are two options now to achieve custom search functionality:

The old way: We can use the typical SearchView but pass in one of our own custom forms by extending one of haystack’s forms.

The new way as of Haystack 2.4 (requires Django 1.8) released on June 9th 2015. We can create our own view by extending SearchView, or one of the other views built into haystack.

I tried the new way since it’s new. I couldn’t get it to work by not being able to get any search results when querying on the Sheltr website. So I tried the old way and it worked in an instant. Not sure what the issue was but I’m taking the old route. And this is how it goes:

In sheltr/urls.py we have:

urlpatterns += patterns('haystack.views',

url(r'^mysearch/$', SearchView(

template='search/search.html',

form\_class=ProviderSearchForm

), name='provider\_search'),

)

For the url regex I just made up some random name,’mysearch’, so the search page will be at localhost/mysearch.

We are using the SearchView provided by haystack. In its parameters we pass in the HTML template we wish to use for our search page and we pass in the form we wish to use. We’re making our own form by extending one of haystack’s forms. You can see what forms are built into haystack in its forms.py file.

Making our own form ProviderSearchForm in app/forms.py by extending SearchForm—We have:

class ProviderSearchForm(SearchForm):

location\_name = forms.CharField(required=False)

category = forms.CharField(required=False)

def search(self):

# First, store the SearchQuerySet received from other processing.

sqs = super(ProviderSearchForm, self).search()

if not self.is\_valid():

return self.no\_query\_found()

if self.cleaned\_data['location\_name']:

sqs = sqs.filter(address1=self.cleaned\_data['location\_name'])

return sqs

The form is where all the good stuff happens—meaning it’s where we define our query filters.

I’m just going to run through it in case you’re not savvy with django forms.

We’ve defined location\_name and category as two fields for the user to enter queries. At the moment, we made it so that these fields are not required to be used by the user. This is because we’re extending SearchForm which already has a required field labeled “Search”. So we have three fields for the user to enter queries in. That’s the front end.

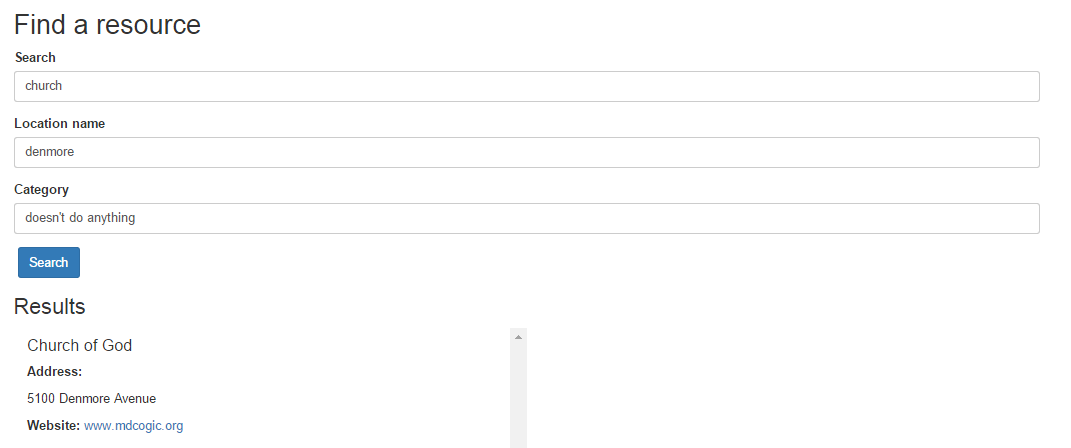
In the search function, which we are overriding from SearchForm, we define how we want to filter the queries entered by the user in the fields we just defined.

The sqs variable stands for search query set, and in it we are storing a SearchQuerySet object defined by haystack. You can basically see it as all the search results, and we can choose to filter certain search results out. So at the beginning, we call the parent’s (SearchForm) search function and it returns a SearchQuerySet that we store into sqs. The SearchForm search just matches whatever text you type in to ANY field you defined in the document, provider\_text.txt (see second bullet in pointers if you don’t know what I’m talking about). I’m not sure about the second if condition but it seems self-explanatory, if the form wasn’t filled out in a valid way, then it won’t return search results.

Finally, the last if statement is our first (and currently only) custom filter that we made. It says that if the location\_name field was filled out, then further filter the search results (sqs) that we got from super.search() by seeing if the query entered by the user into ‘location\_name’ matches address1, an attribute of our Provider model. (Okay right now I know this might be a bit confusing since location\_name is also an attribute of the Provider model—this has NOTHING to do with the location\_name attribute in Provider).

Since we didn’t filter anything based on the category field on the form, it literally does nothing right now so it doesn’t matter if the user enters anything or not—it won’t affect the search results.

Here is an example query in our ProviderSearchForm that returns one search result. The first field is from the parent, SearchForm. It returns all search results with “Church” anywhere in their provider\_name OR address1 since SearchForm.search() looks through provider\_text.txt. Then, those results are further filtered by what is entered into the “Location name” field.



**Notes/pointers on adding more search customization**

* If you want to search through more model attributes, then you must create an index for that attribute in search\_indexes.py like it has been done for provider\_name and address1. Every time you change search\_indexes.py, you have to build the schema file again, and move it to the appropriate location. For Solr 4, this is in solr-4.9.1/example/solr/<corename> /conf. The schema.xml defines which attributes of a model are represented to Solr. So it makes sense that we must rebuild the schema when modifying which attributes we wish for Solr to search through. Remember to also then rebuild the index and restart the Solr server.
* In search\_indexes.py, there is a ‘text’ field. This does not correspond to a single model attribute. What is the text field made up of? The text field is precisely the content in templates/search/indexes/app/provider\_text.txt. If you want a mode attribute to be searchable then you put it in here as {{ object.<model\_attribute> }}. This is what the search() function of SearchForm searches through. So if you had every model attribute of you model in here, search() would look through all of them. Every time you modify provider\_text.txt you will want to rebuild\_index and restart the Solr server.
* When in doubt, restart the Solr server for changes to take place.
* There’s a neat tool in the Solr admin page for testing out queries if you’re not getting much info by querying on the Sheltr site. In the dropdown in the left border, pick the core (collection1), and then click on ‘Query’. The ‘q’ field represents query, where you can enter whatever you want to query for like you would on the website. The default query \*:\* just responds with everything. After clicking ‘Execute Query’, you’ll see all the search results returned along with their individual fields. This way, you know if you’re indexing a field or not. Notice how you can see “text” just like the way you defined it in provider\_text.txt. Note that if a model attribute is blank/doesn’t contain content, then you won’t see its value nor its key show up.

