Django-Select2 Documentation

Release 5.11.0

Nirupam Biswas

Jul 21, 2017

Contents

Contents:

CHAPTER 1

Get Started

Overview

This is a Django integration of Select2.

The application includes Select2 driven Django Widgets and Form Fields.

Installation

1. Install `django_select2`:

```
pip install django_select2
```

2. Add `django_select2` to your `INSTALLED_APPS` in your project settings.

3. Add `django_select` to your `urlconf` **if** you use any *`ModelWidgets`*:

```
url(r'^select2/', include('django_select2.urls')),
```

External Dependencies

- **jQuery version 2** This is not included in the package since it is expected that in most scenarios this would already be available.

Example Application

Please see `tests/testapp` application. This application is used to manually test the functionalities of this package. This also serves as a good example.

CHAPTER 2

API Documentation

Configuration

Settings for Django-Select2.

class `django_select2.conf.`**`Select2Conf`**(***kwargs*)

Bases: `appconf.base.AppConf`

Settings for Django-Select2.

`CACHE_BACKEND` **= u'default'**

Django-Select2 uses Django's cache to sure a consistent state across multiple machines.

Example of settings.py:

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    'select2': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}

# Set the cache backend to select2
SELECT2_CACHE_BACKEND = 'select2'
```

Tip: To ensure a consistent state across all you machines you need to user a consistent external cache backend like Memcached, Redis or a database.

Note: The timeout of select2's caching backend determines how long a browser session can last. Once widget is dropped from the cache the json response view will return a 404.

CACHE_PREFIX = u'select2_'
If you caching backend does not support multiple databases you can isolate select2 using the cache prefix setting. It has set *select2_* as a default value, which you can change if needed.

JS = u'//cdnjs.cloudflare.com/ajax/libs/select2/4.0.3/js/select2.min.js'
The URI for the Select2 JS file. By default this points to the Cloudflare CDN.

If you want to select the version of the JS library used, or want to serve it from the local 'static' resources, add a line to your settings.py like so:

```
SELECT2_JS = 'assets/js/select2.min.js'
```

Tip: Change this setting to a local asset in your development environment to develop without an Internet connection.

CSS = u'//cdnjs.cloudflare.com/ajax/libs/select2/4.0.3/css/select2.min.css'
The URI for the Select2 CSS file. By default this points to the Cloudflare CDN.

If you want to select the version of the library used, or want to serve it from the local 'static' resources, add a line to your settings.py like so:

```
SELECT2_CSS = 'assets/css/select2.css'
```

Tip: Change this setting to a local asset in your development environment to develop without an Internet connection.

I18N_PATH = u'//cdnjs.cloudflare.com/ajax/libs/select2/4.0.3/js/i18n'
The base URI for the Select2 i18n files. By default this points to the Cloudflare CDN.

If you want to select the version of the JS library used, or want to serve it from the local 'static' resources, add a line to your settings.py like so:

```
SELECT2_JS = 'assets/js/i18n'
```

Tip: Change this setting to a local asset in your development environment to develop without an Internet connection.

I18N_AVAILABLE_LANGUAGES = [u'ar', u'az', u'bg', u'ca', u'cs', u'da', u'de', u'el', u'en', u'es', u'et', u'eu', u'fa', u'fi'
List of available translations.

List of ISO 639-1 language codes that are supported by Select2. If currently set language code (e.g. using the HTTP `Accept-Language` header) is in this list, Django-Select2 will use the language code to create load the proper translation.

The full path for the language file consists of:

```
from django.utils import translations

full_path = "{i18n_path}/{language_code}.js".format(
```

```
    i18n_path=settings.DJANGO_SELECT2_I18N,
    language_code=translations.get_language(),
)
```

`settings.DJANGO_SELECT2_I18N` refers to *I18N_PATH*.

class Meta
Prefix for all Django-Select2 settings.

prefix = u'SELECT2'

Widgets

Django-Select2 Widgets.

These components are responsible for rendering the necessary HTML data markups. Since this whole package is to render choices using Select2 JavaScript library, hence these components are meant to be used with choice fields.

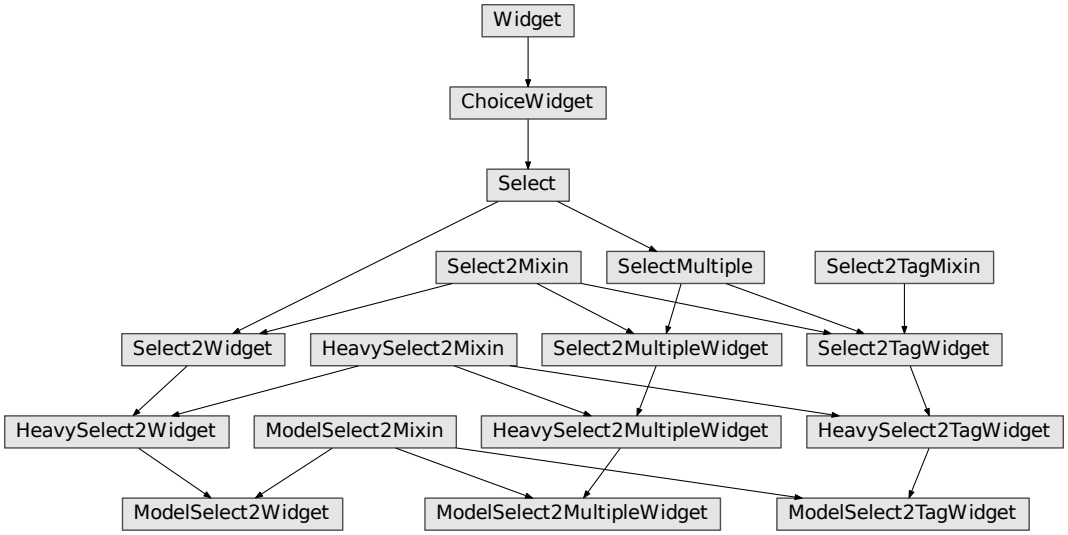
Widgets are generally of two types:

1. **Light** – They are not meant to be used when there are too many options, say, in thousands. This is because all those options would have to be pre-rendered onto the page and JavaScript would be used to search through them. Said that, they are also one the most easiest to use. They are a drop-in-replacement for Django's default select widgets.

2. **Heavy** – They are suited for scenarios when the number of options are large and need complex queries (from maybe different sources) to get the options. This dynamic fetching of options undoubtedly requires Ajax communication with the server. Django-Select2 includes a helper JS file which is included automatically, so you need not worry about writing any Ajax related JS code. Although on the server side you do need to create a view specifically to respond to the queries.

3. **Model** – Model-widgets are a further specialized versions of Heavies. These do not require views to serve Ajax requests. When they are instantiated, they register themselves with one central view which handles Ajax requests for them.

Heavy widgets have the word 'Heavy' in their name. Light widgets are normally named, i.e. there is no 'Light' word in their names.



class django_select2.forms.**Select2Mixin**
Bases: object

The base mixin of all Select2 widgets.

This mixin is responsible for rendering the necessary data attributes for select2 as well as adding the static form media.

build_attrs(**args*, ***kwargs*)
Add select2 data attributes.

optgroups(*name*, *value*, *attrs=None*)
Add empty option for clearable selects.

render_options(**args*, ***kwargs*)
Render options including an empty one, if the field is not required.

media
Construct Media as a dynamic property.

> **Note:** For more information visit https://docs.djangoproject.com/en/1.8/topics/forms/media/#media-as-a-dynamic-property

class django_select2.forms.**Select2TagMixin**
Bases: object

Mixin to add select2 tag functionality.

build_attrs(**args*, ***kwargs*)
Add select2's tag attributes.

class django_select2.forms.**Select2Widget**(*attrs=None*, *choices=()*)
Bases: *django_select2.forms.Select2Mixin*, django.forms.widgets.Select

Select2 drop in widget.

Example usage:

```
class MyModelForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = ('my_field', )
        widgets = {
            'my_field': Select2Widget
        }
```

or:

```
class MyForm(forms.Form):
    my_choice = forms.ChoiceField(widget=Select2Widget)
```

media

class django_select2.forms.**Select2MultipleWidget**(*attrs=None*, *choices=()*)
Bases: *django_select2.forms.Select2Mixin*, django.forms.widgets.SelectMultiple

Select2 drop in widget for multiple select.

Works just like *Select2Widget* but for multi select.

media

class django_select2.forms.**Select2TagWidget**(*attrs=None*, *choices=()*)
Bases: *django_select2.forms.Select2TagMixin*, *django_select2.forms.Select2Mixin*, django.forms.widgets.SelectMultiple

Select2 drop in widget for for tagging.

Example for django.contrib.postgres.fields.ArrayField:

```
class MyWidget(Select2TagWidget):

    def value_from_datadict(self, data, files, name):
        values = super(MyWidget, self).value_from_datadict(data, files, name):
        return ",".join(values)

    def optgroups(self, name, value, attrs=None):
        values = value[0].split(',') if value[0] else []
        selected = set(values)
        subgroup = [self.create_option(name, v, v, selected, i) for i, v in␣
↪enumerate(values)]
        return [(None, subgroup, 0)]
```

media

class django_select2.forms.**HeavySelect2Mixin**(*attrs=None*, *choices=()*, ***kwargs*)
Bases: object

Mixin that adds select2's AJAX options and registers itself on Django's cache.

dependent_fields = {}

get_url()
Return URL from instance or by reversing data_view.

build_attrs(**args*, ***kwargs*)
Set select2's AJAX attributes.

render(**args*, ***kwargs*)
Render widget and register it in Django's cache.

set_to_cache()
Add widget object to Django's cache.

You may need to overwrite this method, to pickle all information that is required to serve your JSON response view.

render_options(**args*)
Render only selected options.

class django_select2.forms.**HeavySelect2Widget**(*attrs=None*, *choices=()*, ***kwargs*)
Bases: *django_select2.forms.HeavySelect2Mixin*, *django_select2.forms.Select2Widget*

Select2 widget with AJAX support that registers itself to Django's Cache.

Usage example:

```
class MyWidget(HeavySelect2Widget):
    data_view = 'my_view_name'
```

or:

```
class MyForm(forms.Form):
    my_field = forms.ChoicesField(
        widget=HeavySelect2Widget(
            data_url='/url/to/json/response'
        )
    )
```

media

class django_select2.forms.**HeavySelect2MultipleWidget**(*attrs=None*, *choices=()*, ***kwargs*)
Bases: *django_select2.forms.HeavySelect2Mixin*, *django_select2.forms.Select2MultipleWidget*

Select2 multi select widget similar to *HeavySelect2Widget*.

media

class django_select2.forms.**HeavySelect2TagWidget**(*attrs=None*, *choices=()*, ***kwargs*)
Bases: *django_select2.forms.HeavySelect2Mixin*, *django_select2.forms.Select2TagWidget*

Select2 tag widget.

media

class django_select2.forms.**ModelSelect2Mixin**(**args*, ***kwargs*)
Bases: object

Widget mixin that provides attributes and methods for *AutoResponseView*.

model = None

queryset = None

search_fields = []
Model lookups that are used to filter the QuerySet.

Example:

```
search_fields = [
        'title__icontains',
    ]
```

max_results = 25
Maximal results returned by *AutoResponseView*.

set_to_cache()
Add widget's attributes to Django's cache.

Split the QuerySet, to not pickle the result set.

filter_queryset(*term*, *queryset=None*, ***dependent_fields*)
Return QuerySet filtered by search_fields matching the passed term.

Parameters

- **term** (*str*) – Search term
- **queryset** (*django.db.models.query.QuerySet*) – QuerySet to select choices from.
- ****dependent_fields** – Dependent fields and their values. If you want to inherit
- **ModelSelect2Mixin and later call to this method, be sure to pop** (*from*) –
- **kwargs everything if it is not a dependent field.** (*from*) –

Returns Filtered QuerySet

Return type QuerySet

get_queryset()
Return QuerySet based on *queryset* or *model*.

Returns QuerySet of available choices.

Return type QuerySet

get_search_fields()
Return list of lookup names.

optgroups(*name*, *value*, *attrs=None*)
Return only selected options and set QuerySet from *ModelChoicesIterator*.

render_options(**args*)
Render only selected options and set QuerySet from `ModelChoiceIterator`.

label_from_instance(*obj*)
Return option label representation from instance.

Can be overridden to change the representation of each choice.

Example usage:

```
class MyWidget(ModelSelect2Widget):
    def label_from_instance(obj):
        return force_text(obj.title).upper()
```

Parameters **obj** (*django.db.models.Model*) – Instance of Django Model.

Returns Option label.

Return type str

class django_select2.forms.**ModelSelect2Widget**(**args*, ***kwargs*)

Bases: *django_select2.forms.ModelSelect2Mixin*, *django_select2.forms.HeavySelect2Widget*

Select2 drop in model select widget.

Example usage:

```
class MyWidget(ModelSelect2Widget):
    search_fields = [
        'title__icontains',
    ]

class MyModelForm(forms.ModelForm):
    class Meta:
        model = MyModel
        fields = ('my_field', )
        widgets = {
            'my_field': MyWidget,
        }
```

or:

```
class MyForm(forms.Form):
    my_choice = forms.ChoiceField(
        widget=ModelSelect2Widget(
            model=MyOtherModel,
            search_fields=['title__icontains']
        )
    )
```

Tip: The ModelSelect2(Multiple)Widget will try to get the QuerySet from the fields choices. Therefore you don't need to define a QuerySet, if you just drop in the widget for a ForeignKey field.

---

media

class django_select2.forms.**ModelSelect2MultipleWidget**(**args*, ***kwargs*)

Bases: *django_select2.forms.ModelSelect2Mixin*, *django_select2.forms.HeavySelect2MultipleWidget*

Select2 drop in model multiple select widget.

Works just like *ModelSelect2Widget* but for multi select.

media

class django_select2.forms.**ModelSelect2TagWidget**(**args*, ***kwargs*)

Bases: *django_select2.forms.ModelSelect2Mixin*, *django_select2.forms.HeavySelect2TagWidget*

Select2 model widget with tag support.

This it not a simple drop in widget. It requires to implement you own `value_from_datadict()` that adds missing tags to you QuerySet.

Example:

```
class MyModelSelect2TagWidget(ModelSelect2TagWidget):
    queryset = MyModel.objects.all()
```

```
    def value_from_datadict(self, data, files, name):
        values = super().value_from_datadict(self, data, files, name)
        qs = self.queryset.filter(**{'pk__in': list(values)})
        pks = set(force_text(getattr(o, pk)) for o in qs)
        cleaned_values = []
        for val in value:
            if force_text(val) not in pks:
                val = queryset.create(title=val).pk
            cleaned_values.append(val)
        return cleaned_values
```

media

URLs

Django-Select2 URL configuration.

Add *django_select* to your `urlconf` **if** you use any 'Model' fields:

```
url(r'^select2/', include('django_select2.urls')),
```

Views

JSONResponse views for model widgets.

class `django_select2.views.`**AutoResponseView**(***kwargs*)

Bases: `django.views.generic.list.BaseListView`

View that handles requests from heavy model widgets.

The view only supports HTTP's GET method.

get(*request*, **args*, ***kwargs*)

Return a `django.http.JsonResponse`.

Example:

```
{
    'results': [
        {
            'text': "foo",
            'id': 123
        }
    ],
    'more': true
}
```

get_queryset()

Get QuerySet from cached widget.

get_paginate_by(*queryset*)

Paginate response by size of widget's *max_results* parameter.

get_widget_or_404()

Get and return widget from cache.

Raises `Http404` – If if the widget can not be found or no id is provided.

Returns Widget from cache.

Return type *ModelSelect2Mixin*

Cache

Shared memory across multiple machines to the heavy AJAX lookups.

Select2 uses django.core.cache to share fields across multiple threads and even machines.

Select2 uses the cache backend defined in the setting `SELECT2_CACHE_BACKEND` [default=‘‘default‘‘].

It is advised to always setup a separate cache server for Select2.

JavaScript

DjangoSelect2 handles the initialization of select2 fields automatically. Just include `{{ form.media.js }}` in your template before the closing `body` tag. That’s it!

If you insert forms after page load or if you want to handle the initialization yourself, DjangoSelect2 provides a jQuery plugin. It will handle both normal and heavy fields. Simply call `djangoSelect2(options)` on your select fields.:

```
$('.django-select2').djangoSelect2();
```

You can pass see Select2 options if needed:

```
$('.django-select2').djangoSelect2({placeholder: 'Select an option'});
```

Security & Authentication

Security is important. Therefore make sure to read and understand what the security measures in place and their limitations.

Set up a separate cache. If you have a public form that uses a model widget make sure to setup a separate cache database for Select2. An attacker could constantly reload your site and fill up the select2 cache. Having a separate cache allows you to limit the effect to select2 only.

You might want to add a secure select2 JSON endpoint for data you don’t want to be accessible to the general public. Doing so is easy:

```
class UserSelect2View(LoginRequiredMixin, AutoResponseView):
    pass

class UserSelect2WidgetMixin(object):
    def __init__(self, *args, **kwargs):
        kwargs['data_view'] = 'user-select2-view'
        super(UserSelect2WidgetMixin, self).__init__(*args, **kwargs)

class MySecretWidget(UserSelect2WidgetMixin, Select2ModelWidget):
    model = MySecretModel
    search_fields = ['title__icontains']
```

CHAPTER 3

Extra

Chained select2

Suppose you have an address form where a user should choose a Country and a City. When the user selects a country we want to show only cities belonging to that country. So the one selector depends on another one.

Models

Here are our two models:

```
class Country(models.Model):
    name = models.CharField(max_length=255)


class City(models.Model):
    name = models.CharField(max_length=255)
    country = models.ForeignKey('Country', related_name="cities")
```

Customizing a Form

Lets link two widgets via *dependent_fields*.

```
class AddressForm(forms.Form):
    country = forms.ModelChoiceField(
        queryset=Country.objects.all(),
        label=u"Country",
        widget=ModelSelect2Widget(
            search_fields=['name__icontains'],
        )
    )

    city = forms.ModelChoiceField(
```

```
        queryset=City.objects.all(),
        label=u"City",
        widget=ModelSelect2Widget(
            search_fields=['name__icontains'],
            dependent_fields={'country': 'country'},
            max_results=500,
        )
    )
```

Interdependent select2

Also you may want not to restrict the user to which field should be selected first. Instead you want to suggest to the user options for any select2 depending of his selection in another one.

Customize the form in a manner:

```
class AddressForm(forms.Form):
    country = forms.ModelChoiceField(
        queryset=Country.objects.all(),
        label=u"Country",
        widget=ModelSelect2Widget(
            search_fields=['name__icontains'],
            dependent_fields={'city': 'cities'},
        )
    )

    city = forms.ModelChoiceField(
        queryset=City.objects.all(),
        label=u"City",
        widget=ModelSelect2Widget(
            search_fields=['name__icontains'],
            dependent_fields={'country': 'country'},
            max_results=500,
        )
    )
```

Take attention to country's dependent_fields. The value of 'city' is 'cities' because of related name used in a filter condition *cities* which differs from widget field name *city*.

Caution: Be aware of using interdependent select2 in parent-child relation. When a child is selected, you are restricted to change parent (only one value is available). Probably you should let the user reset the child first to release parent select2.

Multi-dependent select2

Furthermore you may want to filter options on two or more select2 selections (some code is dropped for clarity):

```
class SomeForm(forms.Form):
    field1 = forms.ModelChoiceField(
        widget=ModelSelect2Widget(
        )
```

```
    )

    field2 = forms.ModelChoiceField(
        widget=ModelSelect2Widget(
        )
    )

    field3 = forms.ModelChoiceField(
        widget=ModelSelect2Widget(
            dependent_fields={'field1': 'field1', 'field2': 'field2'},
        )
    )
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

O

P

Q

R

S