# Django-Select2 Documentation

**Nirupam Biswas**

**Mar 11, 2018**

# Contents

Contents:

CHAPTER 1

# Get Started

## 1.1 Overview

This is a Django integration of Select2.

The application includes Select2 driven Django Widgets and Form Fields.

## 1.2 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')),
```

## 1.3 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.

## 1.4 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

## 2.1 Configuration

## 2.2 Widgets

## 2.3 URLs

## 2.4 Views

## 2.5 Cache

## 2.6 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'});
```

## 2.7 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

## 3.1 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.

### 3.1.1 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")
```

### 3.1.2 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(
            model=Country,
            search_fields=['name__icontains'],
        )
    )
```

(continues on next page)

(continued from previous page)

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

## 3.2 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.

## 3.3 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

## D