django-admin-autocomplete-all Documentation

Release 0.6.6

Mirek Zvolský

Feb 16, 2023

Contents

Contents:

CHAPTER 1

django_admin_autocomplete_all

3 different things in Django Admin:

1) Use select2 (autocomplete_fields) everywhere (because implicit is better than explicit).
2) Get more context for filtering in get_search_results; allows relative easy filtering of popup options
3) Hide danger delete/edit buttons near to the ForeignKey popups

1.1 Important note

There are many ways how to include autocomplete support for ForeignKeys in Django.

Here we have a manual way, which probably is not good way to go, but we can understand how the things could work internally: https://simpleisbetterthancomplex.com/tutorial/2018/01/29/how-to-implement-dependent-or-chained-dropdown-list-with-django.html

The robust industrial and standard way (I think so) is the usage of django-autocomplete-light. The package can be used outside of and inside in the django Admin. This requires lot of clones of very similar code: 1) in urls.py to attach all ajax entrypoints, 2) in views.py lot of ajax views based on autocomplete.Select2QuerySetView with get_queryset() method which handles the seeked string and optionally the from client forwarded filtering values, 3) the forms which attach the autocomplete.ModelSelect2 widgets, 4) inside the admin.py the attaching of such forms to ModelAdmin or Inlines.

It is (django-autocomplete-light) clean and easy solution. However with 20 models in db schema previous can be a difficult work for many hours where you can make lot of mistakes. This is probably one area where django-admin-autocomplete-all can be easier and faster to implement: Just turn it on (see Usage) and *./manage.py check* will write where you need add a ModelAdmin (or alternativelly HiddenAdmin) and define its get_search_results (or get_search_results_ajax) method.

So it is one way how to go: inside Admin django-admin-autocomplete-all and outside of Admin django-autocomplete-light. However after some tweaking I see the idea of Django 2+ autocomplete_fields (used by django-admin-autocomplete-all) as pure implemented. So I will personally prefer use of the great django-autocomplete-light package everywhere. Thats why you cannot wait big improvements of this package in the future. But I think with Py 3.9, Dj 3.1 it simple works.

1.2 Documentation

The full documentation is at https://django-admin-autocomplete-all.readthedocs.io.

1.3 Quickstart

Install django-admin-autocomplete-all:

```
pip install django-admin-autocomplete-all
```

Add *autocomplete_all* into *INSTALLED_APPS*, then collectstatic (both not required if you don't want enhanced get_search_results filtering.)

1.4 Features

(1) **Add autocomplete_fields for all foreign keys.**

No need to change INSTALLED_APPS to achieve this. In your admin.py do *import autocomplete_all as admin*.

```
import autocomplete_all as admin
class MyModelAdmin(admin.ModelAdmin):
    ....
```

Alternatively import ModelAdmin, StackedInline and/or TabularInline 'from autocomplete_all' instead of 'from admin'.

```
import autocomplete_all
class MyModelAdmin(autocomplete_all.ModelAdmin):
    ....
```

You will then need implement lot of search_fields=.. settings in related ModelAdmins. You can try start (ie. runserver) without adding *search_fields* and Django will show you what is required.

(2) **Get more context in get_search_results for additional dynamic filtering.**

Standard Django *autocomplete_fields* cannot inside *get_search_results* distinguish between the ForeignKey which asks for the queryset, especially if 2 ForeignKey's from single model target into same model (often example: ForeignKey into User model). If you add this package ('autocomplete_all') into INSTALLED_APPS, then ?key=… will be added into url. Inside *get_search_results* you will have access to: application, model, ForeignKey. See example in *static/autocomplete_all/js/autocomplete_all.js*.

You need implement filtering into get_search_results of target ModelAdmin (you can use HiddenAdmin class instead). Instead of get_search_results you can use get_search_results_ajax which run for the autocomplete/ access only.

You can also implement dynamic filters based on current value of other form fields. See Usage for details or read in source code: *autocomplete_all/js/autocomplete_all.js* and *autocomplete_all.py: ModelAdmin,get_search_results_ajax*.

(3) **Hide danger buttons in Admin ChangeForm.**

The edit & delete buttons near the ForeignKey have very difficult and danger logic what they will do. If you add *autocomplete_all* in *INSTALLED_APPS* before *django.contrib.admin* (or some application which replaces admin design, like django-baton), then the danger buttons will disapear. Place the *autocomplete_all* "lower" in list if you don't want this effect.

1.5 Running Tests

Does the code actually work? /N/A while we haven't the 1st test yet./

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements_test.txt
(myenv) $ tox
```

1.6 Credits

Tools used in rendering this package:

- cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

For usage:

```
$ pip install django-admin-autocomplete-all
```

For development, with virtualenvwrapper installed:

```
$ git clone https://github.com/pyutil/django-admin-autocomplete-all.git
$ mkvirtualenv django-admin-autocomplete-all -p python3 -a django-admin-autocomplete-
↪all/ -r requirements.txt
```

CHAPTER 3

Usage

This package has 3 independent functions:

1. Add autocomplete_fields for all foreign keys. (no need to change *INSTALLED_APPS*, but in admin.py do *import autocomplete_all as admin*)
2. Add ?key=... into url to have more information inside get_search_results. (requires adding of *autocomplete_all* into *INSTALLED_APPS*)
3. Hide danger delete/edit buttons near the ForeignKey popup (*INSTALLED_APPS*: place *autocomplete_all* before *django.contrib.admin*)

Contents

3.1 1. Add autocomplete_fields for all foreign keys.

To achieve this, in your *admin.py* do: *import autocomplete_all as admin*.

```
import autocomplete_all as admin
class MyModelAdmin(admin.ModelAdmin):
    ....
```

Alternatively import ModelAdmin, StackedInline and/or TabularInline from *autocomplete_all* instead of from *admin*

```
import autocomplete_all
class MyModelAdmin(autocomplete_all.ModelAdmin):
    ....
```

In such class no other settings are required, but you can:

```
autocomplete_except = [<field1>, ..]    # disable adding autocomplete_fields for␣
↪listed fields
autocomplete_all = False                # disable automatic adding of autocomplete_
↪fields at all
```

At the first start you will probably receive a lot of django errors/warnings. They mean that you don't have registered the related ModelAdmin classes or that such class lacks search_fields=..

To solve these messages you can register the required ModelAdmin classes (make sure they have search_fields=..). See ModelAdmin.autocomplete_fields in Django docs for details.

Alternatively you can disable the functionality in particular case using autocomplete_exclude=[..]. This is useful if you have models not fully used yet (ie. empty) and you want prevent them to be accessible in admin interface.

Example: In the application models you have ForeignKey(s) related into User model (outside of this application). To avoid error messages while starting your Django project add:

```
from django.contrib.auth.models import User
from django.contrib.auth.admin import UserAdmin  # must define search_fields=..␣
↪(which is true in this case)
admin.site.register(User, UserAdmin)
```

Recap - Steps to implement:

1. For the popup (for the ForeignKey) we need the Target Admin - ModelAdmin of the model, where the ForeignKey links
2. If Target Admin doesn't exist and we don't want them, we will replace it using HiddenAdmin (from django-admin-autocomplete-all)
3. In the Target Admin we need define search_fields=..

3.2 2. Get more context in get_search_results for additional dynamic filtering.

To achieve this, add *autocomplete_all* into INSTALLED_APPS. The Referer url will then contain *?key*=... and all form values. You can add .get_search_results_ajax() method in the Admin which defines search_fields and you will have access to the values from the form (in key & urlparams variables).

If you want **2 dependent popups** (example: Country/City):

```
# from django.contrib import admin
import autocomplete_all as admin

from .models import City, Country, Friend


@admin.register(Country)
class CountryAdmin(admin.ModelAdmin):
    search_fields = ('name',)
```

(continues on next page)

(continued from previous page)

```
@admin.register(City)
↪# Target admin (searches for popup options)
class CityAdmin(admin.ModelAdmin):
    search_fields = ('name',)

    def get_search_results_ajax(self, queryset, referer, key, urlparams):
        if referer.startswith('friends/friend/'):   # <app>/<model>/  # model of the␣
↪Source (which has popup) Admin (not of the Inline)

            # example for the plain popup
            if key == 'id_city':                          # <field ~ foreignkey>
                queryset = queryset.filter(country=urlparams['country'][0])

            # example for the popup inside the Inline (which lists more locations)
            if key.startswith(before := 'id_location_set-') and key.endswith(after :=
↪'-city'):
                idx = key[len(before):-len(after)]
                queryset = queryset.filter(country=urlparams[f'location_set-{idx}-
↪country'][0])

        return queryset


@admin.register(Friend)
class FriendAdmin(admin.ModelAdmin):   # if you don't need ModelAdmin you can use␣
↪HiddenAdmin instead
    search_fields = ('nick',)

    # no more needed here; autocomplete_all.js is automatically added and gives all␣
↪forms values in the urlparams variable

    # but alternatively you can limit the form values transferred by the ajax request:
    # class Media:
    #     js = ('autocomplete_all/js/autocomplete_all.js', 'friends/js/friend.js')
↪# Source admin

    # `friends.js` you need to create inside the `friends` application. Here is␣
↪example:
    #
    #    function expand_ajax_params($, key) {
    #        return '&country=' + $('#id_country').val();
    #    }
```

Previous will give required data for your *.get_search_results_ajax()* method (of the relational targeted ModelAdmin). That way you can control queryset filtering based on: 1) application, 2) model (where in change_form the popup is), 3) the ForeignKey of the popup.

Warning: At this time we don't support the constraint between the source condition and dependent ForeignKey full. If user has set the Foreignkey for some condition and he/she changes the condition later, the old (inconsistent) value can remain. It is up on to you to clear the popup together with the change of the filtering condition. This could be hard to do. The alternative approach can be raise at least the ValidationError with help of similar definition in your model:

```
# https://stackoverflow.com/questions/2281179/adding-extra-constraints-into-fields-in-
↪django
```

(continues on next page)

(continued from previous page)

```
def clean(self):
    if self.city is not None and self.city.country != self.country:
        raise ValidationError(_("Friend model: City doesn't correspond with the␣
↪selected Country."))
```

Recap - Steps to implement:

1 - 2 - 3. same as above

4. autocomplete_all in INSTALLED_APPS, collectstatic.
5. In Target Admin we add additional filtering with help of the .get_search_results_ajax(self, queryset, referer, key, urlparams) method.

The functionality (giving more context for .get_search_results()) is especially **workaround for pure behaviour of autocomplete_fields** in Django (2,3). Probably you cannot modify the native Django ajax url (../autocomplete/) and you can only access the Referer url during get_search_results.

Lets say, **you have inside single model 2 <select>s with same target model of ForeignKey** (example: User, in two different roles). In such case you cannot identify on the server-side (in get_search_results) which one <select> is active. This package will extend the Referer url to give more info to the server-side.

Basically ?key=<fieldname> will be added to identify the <select>.

3.3 3. Hide danger buttons in Admin ChangeForm.

The edit & delete buttons near the ForeignKey have very difficult and danger logic what they will do. If you add *autocomplete_all* in *INSTALLED_APPS* before *django.contrib.admin* (or some application which replaces admin design, like *django-baton*), then the danger buttons will disapear. Place the *autocomplete_all* “lower” in *INSTALLED_APPS* if you don’t want this effect.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/pyutil/django-admin-autocomplete-all/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

django-admin-autocomplete-all could always use more documentation, whether as part of the official django-admin-autocomplete-all docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/pyutil/django-admin-autocomplete-all/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *django-admin-autocomplete-all* for local development. This is a standard workflow. You can find similar alternative description in github's README.md.

1. Fork the *django-admin-autocomplete-all* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/django-admin-autocomplete-all.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv django-admin-autocomplete-all
   $ cd django-admin-autocomplete-all/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. /N/A while we haven't 1st test./ When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 django_admin_autocomplete_all tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv: pip install -r requirements_test.txt

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for new Python 3 versions and for PyPy (optional). Check tox.ini and make sure that the tests pass for all supported Python versions.

CHAPTER 5

Credits

5.1 Development Lead

- Mirek Zvolský <zvolsky@seznam.cz>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.6.6 (2022-02-16)

Solved HttpRequest.is_ajax() deprecation (Dj 3.1) - thx Darío Fragas for help and contribution

6.2 0.6.3 (2021-03-10)

INCOMPATIBLE: MediaMixin renamed to AutocompleteAllMixin (please rename if you have this class in your code)

0.6.0 (2021-03-09) INCOMPATIBLE: autocomplete_params.js renamend to autocomplete_all.js (please rename if you have this in your code) autocomplete_all.js is now always attached in MediaMixin [later: AutocompleteAllMixin] (for ModelAdmin and both Inlines) ajax autocomplete call now contains all values from the form (see the urlparams variable): you can easier make filtered popups ++++++++++++++++++

0.5.0 (2021-03-05) HiddenAdmin: allow make related admins for search_fields=.. but hide them for direct access (example: we have them better accessible as Inlines) ++++++++++++++++++

0.4.0 (2021-03-04) INCOMPATIBLE: expand_ajax_location_search func renamed to expand_ajax_params (please rename the function if you have it in your javascript) wrapper for queryset filtering moved from example (ie. from commented code) to real code; new method .get_search_results_ajax() in ModelAdmin new documentation in usage.rst ++++++++++++++++++

6.3 0.3.0 (2021-03-03)

- if used in INSTALLED_APPS before django.contrib.admin (or admin rewriting app), danger ForeignKey buttons (edit,delete) will disapear
- import admin methods (example: .register): in many cases you can just *import autocomplete_all as admin* and no more changes in admin.py are needed

6.4 0.2.6 (2020-05-06)

- Fix: added class Media to fix some scenario(s) where widget is missing

6.5 0.2.4 (2020-01-27)

- gives additional context in get_search_results()
- Fix: missing .js (in 0.2.0-0.2.3)

6.6 0.1.6 (2020-01-24)

- Fix in docs: proper attribute name is: autocomplete_except

6.7 0.1.4 (2020-01-22)

- First acceptable version.