wagtail-autocomplete Documentation

Release 0.10.0

Harris Lapiroff

Jul 20, 2023

Contents:

Wagtail Autocomplete provides an edit handler that allows an editor to select related objects via a quick autocompleted searching interface.

CHAPTER 1

Getting Started

1.1 Installation

Install with `pip`:

```
pip install wagtail-autocomplete
```

1.2 Setup

Add `'wagtailautocomplete'` to your project's `INSTALLED_APPS`.

Add Wagtail Autocomplete's URL patterns to your project's URL config, usually in `urls.py`. This should come before your `wagtail_urls` and if you are using the suggested pattern `r'^admin/autocomplete/'` it must also come before your admin urls:

```
from django.conf.urls import include, url

from wagtail.wagtailcore import urls as wagtail_urls

from wagtailautocomplete.urls.admin import urlpatterns as autocomplete_admin_urls

urlpatterns = [
    # ...
    url(r'^admin/autocomplete/', include(autocomplete_admin_urls)),
    url(r'^admin/', include(wagtailadmin_urls)),
    # ...
    url(r'', include(wagtail_urls)),
]
```

This makes available custom API endpoints that provide the search and creation behavior for the widget.

Continue to *Basic Usage* to learn how to use the `AutocompletePanel` on a field in the admin.

CHAPTER 2

Basic Usage

2.1 A Quick Example

We have a `BlogPage` that lets the editor select an `AuthorPage` page.

```
class AuthorPage(Page):
    pass

class BlogPage(Page):
    author = models.ForeignKey(
        'app_label.AuthorPage',
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
    )
```

The `AuthorPage` would traditionally be selected with a `wagtail.wagtailadmin.edit_handlers.PageChooserPanel`, like the following.

```
content_panels = Page.content_panels + [
    PageChooserPanel('author', page_type='app_label.AuthorPage'),
]
```

Instead we can use `AutocompletePanel`.

```
content_panels = Page.content_panels + [
    AutocompletePanel('author'),
]
```

2.2 AutocompletePanel

class wagtailautocomplete.edit_handlers.**AutocompletePanel**(*field_name*, *target_model='wagtailcore.Page'*)

`AutocompletePanel` takes one required argument, the field name. Optionally, you can pass a single `target_model` which will limit the objects an editor can select to that model — this argument can be a reference to a model class or a model string in `app_label.ModelName` syntax.

Note: Unlike `wagtail.wagtailadmin.edit_handlers.PageChooserPanel`, `AutocompletePanel` does not support receiving `target_model` as a list.

Note: `AutocompletePanel` does not support receiving the `can_choose_root` argument that `wagtail.wagtailadmin.edit_handlers.PageChooserPanel` does.

2.3 Multiple Selection With Clusterable Models

`AutocompletePanel` can also be used with a `ParentalManyToManyField` to provide a multiple selection widget. For example:

Note: Use `content_panels` when the model is inherited from `Page`. If it is inherited from `models.Model` or `ClusterableModel`, then we need to use `panels` instead of `content_panels`.

```
from django.db import models
from wagtail.core.models import Page
from modelcluster.models import ClusterableModel
from modelcluster.fields import ParentalManyToManyField

from wagtailautocomplete.edit_handlers import AutocompletePanel

class Book(ClusterableModel):
    title = models.CharField(max_length=255)


class AuthorPage(Page):
    books = ParentalManyToManyField(
        Book,
        null=True,
        related_name='authors'
    )

    content_panels = Page.content_panels + [
        AutocompletePanel('books', target_model=Book)
    ]
```

Note: This above screen capture also shows the availability of Wagtail Autocomplete's "Create New" behavior. To learn more, see *Customization*.

CHAPTER 3

Using Other Models

`AutocompletePanel` works with models other than `wagtail.wagtailcore.Page` and subclasses of it.

3.1 Selecting Snippets

For example, we have a Django model `Link` that we have registered as a snippet. We also have a `BlogPage` model that would traditionally use a `wagtail.wagtailsnippets.edit_handlers.SnippetChooserPanel`

```
from django.db import models

from wagtail.wagtailadmin.edit_handlers import FieldPanel
from wagtail.wagtailcore.models import Page
from wagtail.wagtailsnippets.edit_handlers import SnippetChooserPanel
from wagtail.wagtailsnippets.models import register_snippet

@register_snippet
class Link(models.Model):
    title = models.CharField(max_length=255)
    url = models.URLField()

    panels = [
        FieldPanel('title'),
        FieldPanel('url'),
    ]


class BlogPage(Page):
    external_link = models.ForeignKey(
        'app_label.Link',
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
    )
```

(continues on next page)

(continued from previous page)

```
    content_panels = [
        SnippetChooserPanel('external_link'),
    ]
```

We can replace the `wagtail.wagtailsnippets.edit_handlers.SnippetChooserPanel` usage with `AutocompletePanel`.

```
panels = [
    AutocompletePanel('external_link'),
]
```

Note: Wagtail Autocomplete assumes by default that models have a `title` field. To you autocomplete with target models that don't have a `title` field, see *Customization* for instructions on setting a custom label and search field.

CHAPTER 4

Customization

Wagtail Autocomplete provides the ability to customize the behavior of `AutocompletePanel`.

4.1 "Create New" Behavior

Sometimes you want users to not only be able to select pages or objects, but create new ones on the fly without leaving the object that they're currently editing. This can be particularly useful for tag-like objects, where you want to be able to add a tag with a particular title, even if that tag doesn't already exist in the database.

You can enable this type of behavior by defining an `autocomplete_create` class method on your model. This method should accept a string value and return a new saved model instance:

```
from django.db import models
from wagtailautocomplete.edit_handlers import AutocompletePanel


class MyModel(models.Model):
    title = models.CharField(max_length=255)

    @classmethod
    def autocomplete_create(kls: type, value: str):
        return kls.objects.create(title=value)
```

4.2 Custom Search Field

By default, the autocomplete widget will match input against the `title` field on your model. If you're using a model that doesn't have a `title` attribute, or you just want to search using a different field, you can customize which field it matches against by defining an `autocomplete_search_field` property on your model:

```
from django.db import models
from wagtailautocomplete.edit_handlers import AutocompletePanel


class MyModel(models.Model):
    my_special_field = models.CharField(max_length=255)

    autocomplete_search_field = 'my_special_field'
```

Warning: You will also need to define an `autocomplete_label` function, unless your model has a `title` attribute. See the section on Custom Label Display for more information.

Note: Internally Wagtail Autocomplete uses an `icontains` lookup to search for partial text matches. So, in the example above, if a user enters `'part'` into an autocomplete field, Wagtail Autocomplete will perform the following query to find matches:

```
MyModel.objects.filter(my_special_field__icontains='part')
```

Additionally, this means that `autocomplete_search_field` *must* be a model field and cannot be an arbitrary property or method. There is also the possibility to define a custom filter function, described in *Custom QuerySet Filter Function*.

4.3 Custom Label Display

By default, the autocomplete widget will display the `title` field from a model. You can change this behavior by defining an `autocomplete_label` method on your model:

```
from django.db import models
from wagtailautocomplete.edit_handlers import AutocompletePanel


class MyModel(models.Model):
    my_special_field = models.CharField(max_length=255)

    def autocomplete_label(self):
        return self.my_special_field
```

4.4 Custom QuerySet Filter Function

By default, the autocomplete widget uses an `icontains` lookup to search for matching items of the given model. To change that behavior a custom filter function can be defined, that will be called instead of the default filtering. The function needs to return a QuerySet of the expected model.

```
from django.db import models
from django.db.models import QuerySet
from wagtailautocomplete.edit_handlers import AutocompletePanel

```

(continues on next page)

(continued from previous page)

```
class MyModel(models.Model):
    my_special_field = models.CharField(max_length=255)

    def autocomplete_label(self):
        return self.my_special_field

    @staticmethod
    def autocomplete_custom_queryset_filter(search_term: str) -> QuerySet:
        field_name='my_special_field'
        filter_kwargs = dict()
        filter_kwargs[field_name + '__contains'] = search_term
        return MyModel.objects.filter(**filter_kwargs)
```

CHAPTER 5

Contributing

Wagtail Autocomplete is an open-source project and we welcome contributions! The eventual goal is to merge Wagtail Autocomplete into Wagtail core, so contributions should be made with that in mind.

We accept both issue reports and code contributions through our GitHub repository.

5.1 Code Style

This repo follows Wagtail's guidelines. Clone `wagtail/wagtail` in a separate folder and run linters with their configuration.

```
gem install scss_lint
npm run lint:css -- --config /path/to/wagtail/.scss-lint.yml
npm run lint:js -- --config /path/to/wagtail/.eslintrc

flake8 --config /path/to/wagtail/tox.ini wagtailautocomplete
isort --check-only --diff --recursive wagtailautocomplete
```

5.2 Frontend Development

Wagtail Autocomplete uses *Webpack <https://webpack.js.org/>* to compile our javascript. To have Webpack watch for changes as you develop, first ensure that you have the node requirements installed:

```
npm install
```

then run:

```
npm run start
```

You can end the watch process with `ctrl-C`. *Do* commit compiled Javascript and CSS assets to the repo. Before committing, run:

```
npm run build
```

to create a production-ready build of assets.

5.3 Compiling the documentation

The Wagtail Autocomplete documentation is built with Sphinx. To install Sphinx and compile the documentation, run:

```
cd /path/to/wagtail-autocomplete
pip install -e .[docs]
cd docs
make html
```

The compiled documentation will now be in `docs/_build/html`. Open this directory in a web browser to see it. Python comes with a module that makes it very easy to preview static files in a web browser. To start this simple server, run the following commands:

```
# from insde of /path/to/wagtail-autocomplete/docs
cd _build/html/
python -m http.server 8080
```

Now you can open <http://localhost:8080/> in your web browser to see the compiled documentation.

5.4 Running the test suite

This project uses `pytest` and `tox` to run its test suite. To install `pytest` and run the test suite, run:

```
cd /path/to/wagtail-autocomplete
pip install -e .[test]
pytest
```

To run the test suite against all dependency permutations, ensure that you have all the necessary Python interpreters installed and run:

```
tox
```

If you make changes to test models, you must regenerate the migrations in `wagtailautocomplete/tests/testapp/migrations/`. This can be a sort of tricky process and is left as an excercise to the reader until I'm able to standardize a mechanism for doing so. Since test models are ephemeral it is OK, and even preferable, to regenerate migrations from scratch for each change.

CHAPTER 6

Changelog

6.1 0.10 Release

- Change the search view to use the HTTP POST method, which can prevent the request URI from becoming too long.
- New feature: add the possibility of a custom filter function.

6.2 0.9 Release

- Add Wagtail 3.x compatibility

6.3 0.8.1 Release

- Change in behavior: the autocomplete endpoint will return a 404 response if no objects are found.
- Update Javascript dependencies to remove security vulnerabilities.

6.4 0.7 Release

- Breaking change: Drop deprecated `page_type` and `is_single` arguments from `AutocompletePanel`.
- Update the panel and widget codes based on panels of `wagtail.admin.edit_handlers` – mainly `PageChooserPanel`.
- Update Javascript dependencies to remove security vulnerabilities.
- Update use of deprecated `django.conf.urls.url` function.

6.5 0.6.3 Release

- Remove native browser autocomplete form field.

6.6 0.6 Release

- Add Wagtail 2.8 support

6.7 0.5 Release

- Add Django 3.0 support
- Remove Wagtail 1.x support (Wagtail 2.3 or later now required)
- Documentation fixes

6.8 0.4 Release

- Deprecate `is_single` option, make `target_model` optional. `AutocompletePanel` will now automatically derive these attributes from the field. (#48)
- Remove compatibility for all Python 2.x and Wagtail 1.x versions (#53)

6.9 0.3.1 Release

- Correct documentation for installing tests (#44)
- Correct errors raised by endpoints (#45)

6.10 0.3 Release

- Various improvements to Tox testing and CI setup.
- Various improvements to Webpack compilation.
- Replace `page_type` keyword argument with more accurate `target_model` keyword argument. The old argument still works, but is deprecated.
- Enable autocomplete panel to run its javascript function when it is added to the page dynamically. This allows autocomplete panels to function inside of inline panels.
- Change references from model IDs to model PKs to allow panel compatibility with custom and non-integer primary keys.

Python Module Index

w

Index

A

W