cmsplugin-form-handler Documentation

Release 0.1.3

Martin Koistinen

December 06, 2016

Contents

This package aims to provide a mechanism for handling form-submissions in django-CMS plugins.

Contents:

CHAPTER 1

Background & Approach

1.1 Background

Plugins are a key component of django CMS for creating reusable, configurable content fragments in django CMS projects. Due to their flexibility and utility, project developers would benefit from emitting forms and handling form submissions using plugins.

Since CMS plugins are fragments of a page, they do not provide a unique, URL for receiving and handling form submissions. This presents numerous challenges when attempting to process form submissions.

1.2 Approach

To get around these limitations, the approach taken in this package is to direct form submissions from plugins which sub-class `FormPluginBase` to a URL that is outside of the django CMS URL-space and handled by a `ProcessFormView` provided by this package.

The `ProcessFormView` accepts form-submissions, processes them, and if valid, sends the resulting form back to the plugin class for handling and then responds to the request with a redirect to a `success_url` provided by the plugin.

On validation errors, the view will redirect the request back to the originating page and provide the form data via a session variable back to the plugin's form.

The user experience is precisely as expected and the handling of the form is performed without "thrown HTTPRedirectResponses" or any special middleware.

This package encapsulates all extra logic so that the plugin developer need only to subclass `FormPluginBase` rather than the usual `cms.plugin_base.CMSPluginBase`.

The `Form` or `ModelForm` presented in the CMS plugin should also include the "mixin" `FormPluginFormMixin`.

CHAPTER 2

Quickstart

To get started quickly, first install the package:

```
pip install cmsplugin-form-handler
```

Add the package to `settings.INSTALLED_APPS`:

```
# my_cool_project/settings.py

INSTALLED_APPS = (
    ...
    'cmsplugin_form_handler',
)
```

Add an extra line in your url configuration:

```
urlpatterns = i18n_patterns('',
    url(r'^admin/', include(admin.site.urls)),
    ...
    url(r'^plugin_forms/', include('cmsplugin_form_handler.urls',
                                   namespace='cmsplugin_form_handler')),
    url(r'^', include('cms.urls')),
)
```

Add the `FormPluginFormMixin` mixin to your `Form`:

```
# my_cool_project/forms.py

from django import forms
from cmsplugin_form_handler.forms import FormPluginFormMixin

class MyCoolForm(FormPluginFormMixin, forms.Form):
    # everything else is your normal form.
    my_cool_field = forms.CharField(...)
    ...
```

Or, if you're using a `ModelForm`:

```
# my_cool_project/forms.py

from django import forms
from cmsplugin_form_handler.forms import FormPluginFormMixin

class MyCoolModelForm(FormPluginFormMixin, forms.ModelForm):
    # everything else is your normal form.
```

```
    class Meta:
        model = MyCoolModel
    ...
```

Subclass your cms plugin from `FormPluginBase`:

```
# my_cool_project/cms_plugins.py

from cmsplugin_form_handler.cms_plugins import FormPluginBase

class MyCoolPlugin(FormPluginBase):
    # Use your normal CMSPlugin attributes...
    render_template = 'plugins/my_cool_plugin.html'
    # Note that ``cache = False`` will automatically be set

    # These should be overridden in sub-classes
    form_class = MyCoolForm  # Or, see: get_form_class()
    success_url = '/static/success/url/here'  # Or, see: get_success_url()

    def render(self, context, instance, placeholder):
        context = super(MyCoolPlugin, self).render(context, instance, placeholder)

        # Do your normal thing here
        ...

        return context

    def get_form_class(self, request, instance):
        # Use this method to programmatically determine the form_class.
        # This is what this method does by default:
        return self.form_class

    def get_success_url(self, request, instance):
        # Use this method to programmatically determine the success_url.
        # This is what this method does by default:
        return self.success_url

    def form_valid(self, request, instance, form):
        # Optionally do something with the rendered form here
        # This is what this method does by default:
        form.save()
```

Finally, update your plugin's template:

```
# my_cool_project/templates/plugins/my_cool_plugin.html

{% load cmsplugin_form_tags %}

<h2>Form Plugin</h2>
<form action="{% cmsplugin_form_action %}" method="post">
    {% csrf_token %}
    {{ cmsplugin_form }}
    <input type="submit">
</form>
```

CHAPTER 3

Reference

3.1 cmsplugin_form_handler.cms_plugins

This module contains an alternative super-class for CMS plugins that encapsulates all plugin-related cmsplugin-form-handler logic.

3.1.1 FormPluginBase

class cmsplugin_form_handler.cms_plugins.**FormPluginBase**

This class is a sub-class of the normal cms.plugin_base.CMSPluginBase but offers additional functionality for dealing with plugin-based forms.

Attributes

cmsplugin_form_handler.cms_plugins.**cache**
: This base-class will automatically set the normal cache attribute to False. This can be overridden in the project's plugin class, but it is not recommended because presenting a form should also include a CSRF token, which should never be cached.

cmsplugin_form_handler.cms_plugins.**form_class**
: Set this to the forms.Form or forms.ModelForm you wish this plugin to present. If you need to determine which form to present based on the specific plugin instance, see *get_form_class()*.

cmsplugin_form_handler.cms_plugins.**success_url**
: Set this to the URL of the "success page" of the form. Using this attribute is simple and suitable for static success URLs. However, in most projects, it is likely more appropriate to use *get_success_url()*.

Methods

cmsplugin_form_handler.cms_plugins.**get_form_class**(*request*, *instance*)
: Returns the class of the form that this plugin presents. The default implementation of this method is to simply return the contents of *form_class*. Override this method if different plugins instances of the same plugin class should return different forms.

Parameters

- **request** (*HTTPRequest*) – This is the request object for the form-submission. This may be useful for making a determination about which form class to return.

- **instance** (*CMSPlugin*) – This is the CMS plugin instance of the plugin used to produce the form.

cmsplugin_form_handler.cms_plugins.**get_success_url**(*request*, *instance*)

Returns the desired URL that the user should be redirected to if their form submission validates.

Parameters

- **request** (*HTTPRequest*) – This is the request object for the form-submission. This may be useful for making a determination about which success URL to return.
- **instance** (*CMSPlugin*) – This is the CMS plugin instance of the plugin used to produce the form. (Hint: you could present a list of choices in the `CMSPlugin``model using a ``cms.models.fields.PageField`.)

The default implementation of this method is to simply return the contents of *success_url*, but in most cases, a static URL is inappropriate. For example, it may be better to return the absolute URL of a specific CMS page (which could be moved by the content managers to different paths). In this case, something like this may be useful:

```
# NOTE: only relevant code is shown here...

from cms.models import Page
from cms.utils import get_language_from_request
from cms.utils.i18n import get_default_language

from cmsplugin_form_handler.cms_plugins import FormPluginBase

class SomePlugin(FormPluginBase):
    ...
    success_url = '/'  # a sane default
    ...

    def get_success_url(self, request, instance):
        # Be sure to set this in the Advanced Settings tab of the
        # desired CMS Page.
        reverse_id = 'success_page'

        # We'll need to know which language is relevant...
        lang = get_language_from_request(request) or get_default_language()

        try:
            page = Page.objects.get(
                reverse_id=reverse_id,
                publisher_is_draft=False
            )
        except Page.DoesNotExist:
            # Can't find the success page, return the something sane...
            return self.success_url
        else:
            return page.get_absolute_url(lang)
```

Or, as hinted above, you could use the CMSPlugin model to present a set of choices using a `cms.models.fields.PageField` to the Content Manager when creating the plugin instance, then, use the `get_success_url` method to return the absolute URL of the selected choice.

cmsplugin_form_handler.cms_plugins.**form_valid**(*request*, *instance*, *form*)

This method is called if the form is valid.

Parameters

- **request** (*HTTPRequest*) – This is the request object for the form-submission. This may be useful for determining what to do with the valid form.
- **instance** (*CMSPlugin*) – This is the CMS plugin instance of the plugin used to produce the form.
- **form** (*Form*) – This is the validated form.

The default implementation simply calls the `save` method on the form.

3.2 cmsplugin_form_handler.forms

3.2.1 FormPluginFormMixin

This module contains code that encapsulates the cmsplugin-forms-handler functionality relating to forms.

class `cmsplugin_form_handler.forms.`**FormPluginFormMixin** (*source_url*, **args*, ***kwargs*)

This class is a form "mixin" that may be applied to `forms.Form` or `forms.ModelForm` classes. The mixin embeds a hidden field for passing the source URL which is required for the correct operation of this package.

It also modifies the constructor signature of the form by adding a new, required arg: source_url, but in most cases, this is transparently dealt with by the package.

3.3 cmsplugin_form_handler.templatetags.cmsplugin_form_tags

This module contains template tags that are provided by this package.

3.3.1 form_action

This template tag provides the URL for the form action. It simply returns the correct URL to use for submitting the form. It is roughly equivalent to:

```
{% url 'plugin_form_handler:process_form' instance.pk %}
```

Although simple, the purpose of this tag is to encapsulate the implementation details of cmsplugin-form-handler so that future changes can occur as necessary without breaking existing projects.

param int plugin_pk This can be used to specify the ID of the plugin that the view should use to process the form. If the developer uses CMS development conventions, this parameter should never be necessary. However, there may be some cases where the `render()` method uses a variable other than `instance` in its context. In these cases, it may be necessary to use that variable in this template tag as follows:

```
# In this example, the context includes the variable ``plugin``
# that contains the plugin instance to render

{% load cmsplugin_form_tags %}
...
<form action="{% form_action plugin %}" method="post">
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

C

C

F

G

S

T