django-contact-form Documentation

Release 1.8.1

James Bennett

Feb 17, 2020

Installation and configuration

django-contact-form provides customizable contact-form functionality for Django-powered Web sites.

Basic functionality (collecting a name, email address and message) can be achieved out of the box by setting up a few templates and adding one line to your site's root URLconf:

```
path('contact/', include('contact_form.urls')),
```

For notes on getting started quickly, and on how to customize django-contact-form's behavior, read through the full documentation below.

CHAPTER 1

Installation guide

The 1.8.1 release of django-contact-form supports Django 2.2 and 3.0 on the following Python versions:

- Django 2.2 supports Python 3.5, 3.6, 3.7, and 3.8.
- Django 3.0 supports Python 3.6, 3.7, and 3.8.

1.1 Normal installation

The preferred method of installing django-contact-form is via *pip*, the standard Python package-installation tool. If you don't have *pip*, instructions are available for how to obtain and install it. If you're using a supported version of Python, *pip* should have come bundled with your installation of Python.

Once you have *pip*, type:

```
pip install django-contact-form
```

If you plan to use the included spam-filtering contact form class, *AkismetContactForm*, you will also need the Python akismet module. You can manually install it via *pip install akismet*, or tell django-contact-form to install it for you, by running:

```
pip install django-contact-form[akismet]
```

If you don't have a copy of a compatible version of Django, installing django-contact-form will also automatically install one for you.

1.2 Installing from a source checkout

If you want to work on django-contact-form, you can obtain a source checkout.

The development repository for django-contact-form is at <https://github.com/ubernostrum/django-contact-form>. If you have git installed, you can obtain a copy of the repository by typing:

```
git clone https://github.com/ubernostrum/django-contact-form.git
```

From there, you can use git commands to check out the specific revision you want, and perform an “editable” install (allowing you to change code as you work on it) by typing:

```
pip install -e .
```

1.3 Next steps

To get up and running quickly, check out *the quick start guide*. For full documentation, see *the documentation index*.

CHAPTER 2

Quick start guide

First you'll need to have Django and django-contact-form installed; for details on that, see *the installation guide*.

Once that's done, you can start setting up django-contact-form. First, add *'contact_form'* to your `INSTALLED_APPS` setting. Then, you can begin configuring.

2.1 URL configuration

The quickest way to set up the views in django-contact-form is to use the provided URLconf, found at *contact_form.urls*. You can include it wherever you like in your site's URL configuration; for example, to have it live at the URL */contact/*:

```
from django.urls import include, path


urlpatterns = [
    # ... other URL patterns for your site ...
    path('contact/', include('contact_form.urls')),
]
```

If you'll be using a custom form class, you'll need to manually set up your URLs so you can tell django-contact-form about your form class. For example:

```
from django.urls import include, path
from django.views.generic import TemplateView

from contact_form.views import ContactFormView

from yourapp.forms import YourCustomFormClass


urlpatterns = [
    # ... other URL patterns for your site ...
```

(continues on next page)

(continued from previous page)

```
    path('contact/',
        ContactFormView.as_view(
            form_class=YourCustomFormClass
        ),
        name='contact_form'),
    path('contact/sent/',
        TemplateView.as_view(
            template_name='contact_form/contact_form_sent.html'
        ),
        name='contact_form_sent'),
]
```

Important: Where to put custom forms and views

When writing a custom form class (or custom *ContactFormView* subclass), **don't** put your custom code inside django-contact-form. Instead, put your custom code in the appropriate place (a *forms.py* or *views.py* file) in an application you've written.

2.2 Required templates

The two views above will need several templates to be created.

2.2.1 *contact_form/contact_form.html*

This is used to display the contact form. It has a `RequestContext` (so any context processors will be applied), and also provides the form instance as the context variable *form*.

2.2.2 *contact_form/contact_form_sent.html*

This is used after a successful form submission, to let the user know their message has been sent. It has a `RequestContext`, but provides no additional context variables of its own.

2.2.3 *contact_form/contact_form.txt*

Used to render the subject of the email. Will receive a `RequestContext` with the following additional variables:

body The message the user typed.

email The email address the user supplied.

name The name the user supplied.

site The current site. Either a `Site` or `RequestSite` instance, depending on whether Django's sites framework is installed).

2.2.4 *contact_form/contact_form_subject.txt*

Used to render the subject of the email. Will receive a `RequestContext` with the following additional variables:

body The message the user typed.

email The email address the user supplied.

name The name the user supplied.

site The current site. Either a `Site` or `RequestSite` instance, depending on whether Django's sites framework is installed).

Warning: Subject must be a single line

In order to prevent header injection attacks, the subject *must* be only a single line of text, and Django's email framework will reject any attempt to send an email with a multi-line subject. So it's a good idea to ensure your *contact_form_subject.txt* template only produces a single line of output when rendered; as a precaution, however, django-contact-form will, by default, condense the output of this template to a single line.

2.3 Using a spam-filtering contact form

Spam filtering is a common desire for contact forms, due to the large amount of spam they can attract. There is a spam-filtering contact form class included in django-contact-form: *`AkismetContactForm`*, which uses the Wordpress Akismet spam-detection service.

To use this form, you will need to do the following things:

1. Install the Python *akismet* module to allow django-contact-form to communicate with the Akismet service. You can do this via *pip install akismet*, or as you install django-contact-form via *pip install django-contact-form[akismet]*.
2. Obtain an Akismet API key from <https://akismet.com/>, and associate it with the URL of your site.
3. Supply the API key and URL for django-contact-form to use. You can either place them in the Django settings `AKISMET_API_KEY` and `AKISMET_BLOG_URL`, or in the environment variables *PYTHON_AKISMET_API_KEY* and *PYTHON_AKISMET_BLOG_URL*.

Then you can replace the suggested URLconf above with the following:

```
from django.urls import include, path


urlpatterns = [
    # ... other URL patterns for your site ...
    path('contact/', include('contact_form.akismet_urls')),
]
```

CHAPTER 3

Contact form classes

There are two contact-form classes included in django-contact-form; one provides all the infrastructure for a contact form, and will usually be the base class for subclasses which want to extend or modify functionality. The other is a subclass which adds spam filtering to the contact form.

3.1 The ContactForm class

class `contact_form.forms.`**`ContactForm`**

The base contact form class from which all contact form classes should inherit.

If you don't need any customization, you can use this form to provide basic contact-form functionality; it will collect name, email address and message.

The `ContactFormView` included in this application knows how to work with this form and can handle many types of subclasses as well (see below for a discussion of the important points), so in many cases it will be all that you need. If you'd like to use this form or a subclass of it from one of your own views, here's how:

1. When you instantiate the form, pass the current `HttpRequest` object as the keyword argument *request*; this is used internally by the base implementation, and also made available so that subclasses can add functionality which relies on inspecting the request (such as spam filtering).
2. To send the message, call the form's `save()` method, which accepts the keyword argument *fail_silently* and defaults it to *False*. This argument is passed directly to Django's `send_mail()` function, and allows you to suppress or raise exceptions as needed for debugging. The `save()` method has no return value.

Other than that, treat it like any other form; validity checks and validated data are handled normally, through the `is_valid()` method and the `cleaned_data` dictionary.

Under the hood, this form uses a somewhat abstracted interface in order to make it easier to subclass and add functionality.

The following attributes play a role in determining behavior, and any of them can be implemented as an attribute or as a method (for example, if you wish to have `from_email` be dynamic, you can implement a method named `from_email()` instead of setting the attribute `from_email`).

from_email
The email address (`str`) to use in the *From:* header of the message. By default, this is the value of the Django setting `DEFAULT_FROM_EMAIL`.

recipient_list
A `list` of recipients for the message. By default, this is the email addresses specified in the setting `MANAGERS`.

subject_template_name
A `str`, the name of the template to use when rendering the subject line of the message. By default, this is *contact_form/contact_form_subject.txt*.

template_name
A `str`, the name of the template to use when rendering the body of the message. By default, this is *contact_form/contact_form.txt*.

And two methods are involved in producing the contents of the message to send:

message()
Returns the body of the message to send. By default, this is accomplished by rendering the template name specified in `template_name`.

Return type str

subject()
Returns the subject line of the message to send. By default, this is accomplished by rendering the template name specified in `subject_template_name`.

Return type str

Warning: Subject must be a single line

The subject of an email is sent in a header (named *Subject:*). Because email uses newlines as a separator between headers, newlines in the subject can cause it to be interpreted as multiple headers; this is the header injection attack. To prevent this, `subject()` will always force the subject to a single line of text, stripping all newline characters. If you override `subject()`, be sure to either do this manually, or use `super()` to call the parent implementation.

Finally, the message itself is generated by the following two methods:

get_message_dict()
This method loops through `from_email`, `recipient_list`, `message()` and `subject()`, collecting those parts into a dictionary with keys corresponding to the arguments to Django's *send_mail* function, then returns the dictionary. Overriding this allows essentially unlimited customization of how the message is generated. Note that for compatibility, implementations which override this should support callables for the values of `from_email` and `recipient_list`.

Return type dict

get_context()
For methods which render portions of the message using templates (by default, `message()` and `subject()`), generates the context used by those templates. The default context will be a `RequestContext` (using the current HTTP request, so user information is available), plus the contents of the form's `cleaned_data` dictionary, and one additional variable:

site If *django.contrib.sites* is installed, the currently-active `Site` object. Otherwise, a `RequestSite` object generated from the request.

Return type dict

Meanwhile, the following attributes/methods generally should not be overridden; doing so may interfere with functionality, may not accomplish what you want, and generally any desired customization can be accomplished in a more straightforward way through overriding one of the attributes/methods listed above.

request
The `HttpRequest` object representing the current request. This is set automatically in *__init__()*, and is used both to generate a `RequestContext` for the templates and to allow subclasses to engage in request-specific behavior.

save()
If the form has data and is valid, will send the email, by calling *get_message_dict()* and passing the result to Django's `send_mail()` function.

Note that subclasses which override *__init__* or *save()* need to accept **args* and ***kwargs*, and pass them via `super()`, in order to preserve behavior (each of those methods accepts at least one additional argument, and this application expects and requires them to do so).

3.2 The Akismet (spam-filtering) contact form class

class contact_form.forms.**AkismetContactForm**
A subclass of *ContactForm* which adds spam filtering, via the Wordpress Akismet spam-detection service.

Use of this class requires you to provide configuration for the Akismet web service; you'll need to obtain an Akismet API key, and you'll need to associate it with the site you'll use the contact form on. You can do this at <https://akismet.com/>. Once you have, you can configure in either of two ways:

1. Put your Akismet API key in the Django setting `AKISMET_API_KEY`, and the URL it's associated with in the setting `AKISMET_BLOG_URL`, or
2. Put your Akismet API key in the environment variable *PYTHON_AKISMET_API_KEY*, and the URL it's associated with in the environment variable *PYTHON_AKISMET_BLOG_URL*.

You will also need the Python Akismet module to communicate with the Akismet web service. You can install it by running *pip install akismet*, or django-contact-form can install it automatically for you if you run *pip install django-contact-form[akismet]*.

Once you have an Akismet API key and URL configured, and the *akismet* module installed, you can drop in *AkismetContactForm* anywhere you would have used *ContactForm*. A URLconf is provided in django-contact-form, at *contact_form.akismet_urls*, which will correctly configure *AkismetContactForm* for you.

CHAPTER 4

Built-in views

class `contact_form.views.`**`ContactFormView`**

The base view class from which most custom contact-form views should inherit. If you don't need any custom functionality, and are content with the default `ContactForm` class, you can also use it as-is (and the provided URLConf, *contact_form.urls*, does exactly this).

This is a subclass of Django's `FormView`, so refer to the Django documentation for a list of attributes/methods which can be overridden to customize behavior.

One non-standard attribute is defined here:

`recipient_list`

The list of email addresses to send mail to. If not specified, defaults to the `recipient_list` of the form.

Additionally, the following standard (from `FormView`) methods and attributes are commonly useful to override (all attributes below can also be passed to `as_view()` in the URLconf, permitting customization without the need to write a full custom subclass of `ContactFormView`):

`form_class`

The form class to use. By default, will be `ContactForm`. This can also be overridden as a method named `form_class()`; this permits, for example, per-request customization (by inspecting attributes of *self.request*).

`template_name`

A `str`, the template to use when rendering the form. By default, will be *contact_form/contact_form.html*.

`get_success_url``()`

The URL to redirect to after successful form submission. Can be a hard-coded string, the string resulting from calling Django's `reverse()` helper, or the lazy object produced by Django's `reverse_lazy()` helper. Default value is the result of calling `reverse_lazy()` with the URL name *'contact_form_sent'*.

Return type str

`get_form_kwargs``()`

Returns additional keyword arguments (as a dictionary) to pass to the form class on initialization.

By default, this will return a dictionary containing the current `HttpRequest` (as the key *request*) and, if `recipient_list` was defined, its value (as the key *recipient_list*).

> **Warning:** If you override `get_form_kwargs()`, you **must** ensure that, at the very least, the keyword argument *request* is still provided, or `ContactForm` initialization will raise `TypeError`. The easiest approach is to use `super()` to call the base implementation in `ContactFormView`, and modify the dictionary it returns.

Return type dict

CHAPTER 5

Frequently asked questions

The following notes answer some common questions, and may be useful to you when installing, configuring or using django-contact-form.

5.1 What versions of Django and Python are supported?

As of django-contact-form 1.8.1, Django 2.2 and 3.0 are supported, on Python 3.5 (Django 2.2 only) 3.6, 3.7, and 3.8.

5.2 What license is django-contact-form under?

django-contact-form is offered under a three-clause BSD-style license; this is an OSI-approved open-source license, and allows you a large degree of freedom in modifying and redistributing the code. For the full terms, see the file *LICENSE* which came with your copy of django-contact-form; if you did not receive a copy of this file, you can view it online at <https://github.com/ubernostrum/django-contact-form/blob/master/LICENSE>.

5.3 Why aren't there any default templates I can use?

Usable default templates, for an application designed to be widely reused, are essentially impossible to produce; variations in site design, block structure, etc. cannot be reliably accounted for. As such, django-contact-form provides bare-bones (i.e., containing no HTML structure whatsoever) templates in its source distribution to enable running tests, and otherwise just provides good documentation of all required templates and the context made available to them.

5.4 Why am I getting a bunch of *BadHeaderError* exceptions?

Most likely, you have an error in your `ContactForm` subclass. Specifically, one or more of `from_email`, `recipient_list` or `subject()` are returning values which contain newlines.

As a security precaution against email header injection attacks (which allow spammers and other malicious users to manipulate email and potentially cause automated systems to send mail to unintended recipients), Django's email-sending framework does not permit newlines in message headers. `BadHeaderError` is the exception Django raises when a newline is detected in a header. By default, *`contact_form.forms.ContactForm.subject()`* will forcibly condense the subject to a single line.

Note that this only applies to the headers of an email message; the message body can (and usually does) contain newlines.

5.5 I found a bug or want to make an improvement!

The canonical development repository for django-contact-form is online at <https://github.com/ubernostrum/django-contact-form>. Issues and pull requests can both be filed there.

If you'd like to contribute to django-contact-form, that's great! Just please remember that pull requests should include tests and documentation for any changes made, and that following PEP 8 is mandatory. Pull requests without documentation won't be merged, and PEP 8 style violations or test coverage below 100% are both configured to break the build.

5.6 I'm getting errors about "akismet" when trying to run tests?

The full test suite of django-contact-form exercises all of its functionality, including the spam-filtering `AkismetContactForm`. That class uses the Wordpress Akismet spam-detection service to perform spam filtering, and so requires the Python *akismet* module to communicate with the Akismet service, and some additional configuration (in the form of a valid Akismet API key and associated URL).

By default, the tests for `AkismetContactForm` will be skipped unless the required configuration (in the form of either a pair of Django settings, or a pair of environment variables) is detected. However, if you have supplied Akismet configuration but do *not* have the Python *akismet* module, you will see test errors from attempts to import *akismet*. You can resolve this by running:

```
pip install akismet
```

or (if you do not intend to use `AkismetContactForm`) by no longer configuring the Django settings/environment variables used by Akismet.

Additionally, if the `AkismetContactForm` tests are skipped, the default code-coverage report will fail due to the relevant code not being exercised during the test run.

Python Module Index

Index