django-envelope Documentation

Release 1.3

Zbigniew Siciarz

Sep 27, 2017

Contents

1 Basic usage **3**

2 Resources **5**

3 Authors **7**

4 License **9**

5 Gratipay **11**

6 Documentation **13**
- 6.1 Installation . . . 13
- 6.2 Usage . . . 13
- 6.3 Configuration . . . 14
- 6.4 Customization . . . 14
- 6.5 Cookbook . . . 16
- 6.6 Development . . . 19
- 6.7 Reference . . . 20
- 6.8 Changelog . . . 22
- 6.9 Indices and tables . . . 25

Python Module Index **27**

`django-envelope` is a simple contact form app for Django web framework.

CHAPTER 1

Basic usage

1. Install with `pip install django-envelope`.
2. Add `envelope` to your `INSTALLED_APPS`.
3. Create a template `envelope/contact.html` that contains somewhere a call to `{% render_contact_form %}` template tag. This tag can be imported by placing `{% load envelope_tags %}` at the top of your template.
4. Hook the app's URLconf in your `urls.py` like this:

```
urlpatterns = patterns('',
    #...
    (r'^contact/',    include('envelope.urls')),
    #...
)
```

See the docs for more customization options.

CHAPTER 2

Resources

- Documentation
- Issue tracker
- CI server

CHAPTER 3

Authors

django-envelope is maintained by Zbigniew Siciarz. See AUTHORS.rst for a full list of contributors.

CHAPTER 4

License

This work is released under the MIT license. A copy of the license is provided in the LICENSE file.

The HTML template comes from Open Source Template Project by sendwithus.com, distributed under the Apache 2.0 license (see the APACHE_LICENSE file for the full text).

CHAPTER 5

Gratipay

Like this project? You can support it via Gratipay!

CHAPTER 6

Documentation

Installation

Make sure you have Django installed. Then install the package from PyPI:

```
pip install django-envelope
```

If you like living on the edge, grab the development version from Github:

```
git clone https://github.com/zsiciarz/django-envelope.git
cd django-envelope
python setup.py install
```

To enable a simple antispam check, install django-honeypot. Envelope will automatically pick that one up and use in the contact form.

Usage

Add `envelope` to your `INSTALLED_APPS` in `settings.py`. The application does not define any models, so a `manage.py syncdb` is *not needed*. If you installed `django-honeypot`, add also `honeypot` to `INSTALLED_APPS`.

For a quick start, simply include the app's `urls.py` in your main URLconf, like this:

```
urlpatterns = patterns('',
    #...
    (r'^contact/',    include('envelope.urls')),
    #...
)
```

The view that you just hooked into your URLconf will try to render a `envelope/contact.html` template. Create that file in some location where Django would be able to find it (see the Django template docs for details).

Note: Changed in version 1.0: `django-envelope` used to ship with one such template by default. However, it made too opinionated assumptions about your templates and site layout. For that reason it was removed and you *must* now create the template explicitly.

This template file can (and possibly should) extend your base site template. The view will pass to the context a `form` variable, which is an instance of *`ContactForm`*. You can write your own HTML code for the form or use the provided `{% render_contact_form %}` template tag for simplicity. For example (assuming `base.html` is your main template):

```
{% extends "base.html" %}
{% load envelope_tags %}

{% block content %}
    {% render_contact_form %}
{% endblock %}
```

That's basically it. Navigate to the given URL and see the contact form in action. See *Customization* for more customization options.

Configuration

These values defined in `settings.py` affect the application:

- `DEFAULT_FROM_EMAIL`: This is the sender of the email sent with your contact form.

  Note: (Some mail servers do not allow sending messages from an address that is different than the one used for SMTP authentication.)

- `ENVELOPE_EMAIL_RECIPIENTS`: A list of e-mail addresses of people who will receive the message. For backwards compatibility reasons, the default value is a list where the only element is `DEFAULT_FROM_EMAIL`.
- `ENVELOPE_SUBJECT_INTRO`: The prefix for subject line of the email message. This is different than `EMAIL_SUBJECT_PREFIX` which is global for the whole project. `ENVELOPE_SUBJECT_INTRO` goes after the global prefix and is followed by the actual subject entered in the form by website's user.

  Default value: *Message from contact form:*
- `ENVELOPE_USE_HTML_EMAIL`: Whether to send an HTML email along with the plaintext one. Defaults to `True`.

Customization

Most of the time, including `envelope.urls` is just fine. But if you want more control over the contact form, you need to hook the view into your URLconf yourself. Just import *`envelope.views.ContactView`*, and call the `as_view` classmethod when defining URL patterns.

Example:

```
# urls.py
from django.conf.urls import patterns, url
from envelope.views import ContactView

```

```
urlpatterns = patterns('',
    url(r'^contact/', ContactView.as_view()),
)
```

If you want some more fine-grained control over the contact form, you can customize the view class. You can inherit from *envelope.views.ContactView* and set class attributes in your derived view class, or simply pass the values for these attributes when calling `as_view` in your URLconf.

Example (using a subclass):

```
# some_app/views.py
from envelope.views import ContactView

class MyContactView(ContactView):
    template_name = "my_contact.html"
    success_url = "/thank/you/kind/sir/"

# urls.py
from django.conf.urls import patterns, url
from some_app.views import MyContactView

urlpatterns = patterns('',
    url(r'^contact/', MyContactView.as_view()),
)
```

Example (setting attributes in place):

```
# urls.py
from django.conf.urls import patterns, url
from envelope.views import ContactView

urlpatterns = patterns('',
    url(r'^contact/', ContactView.as_view(
        template_name="my_contact.html",
        success_url="/thank/you/kind/sir/"
    )),
)
```

The following options (as well as those already in Django's FormView) are recognized by the view:

- `form_class`: Which form class to use for contact message handling. The default (*envelope.forms.ContactForm*) is often enough, but you can subclass it if you want, or even replace with a totally custom class. The only requirement is that your custom class has a `save()` method which should send the message somewhere. Stick to the default, or its subclasses.
- `template_name`: Full name of the template which will display the form. By default it is `envelope/contact.html`.
- `success_url`: View name or a hardcoded URL of the page with some kind of a "thank you for your feedback", displayed after the form is successfully submitted. If left unset, the view redirects to itself.
- `form_kwargs`: Additional kwargs to be used in the creation of the form. Use with *envelope.forms.ContactForm* form arguments for dynamic customization of the form.

You can also subclass *envelope.forms.ContactForm* to further customize your form processing. Either set the following options as keyword arguments to `__init__`, or override class attributes.

- `subject_intro`: Prefix used to create the subject line. Default is `settings.ENVELOPE_SUBJECT_INTRO`.

- `from_email`: Used in the email from. Defaults to `settings.DEFAULT_FROM_EMAIL`.
- `email_recipients`: List of email addresses to send the email to. Defaults to `settings.ENVELOPE_EMAIL_RECIPIENTS`.
- `template_name`: Template used to render the plaintext email message. Defaults to `envelope/email_body.txt`. You can use any of the form field names as template variables.
- `html_template_name`: Template used to render the HTML email message. Defaults to `envelope/email_body.html`.

Example of a custom form:

```
# forms.py
from envelope.forms import ContactForm

class MyContactForm(ContactForm):
    subject_intro = "URGENT: "
    template_name = "plaintext_email.txt"
    html_template_name = "contact_email.html"

# urls.py
from django.conf.urls import patterns, url
from envelope.views import ContactView
from forms import MyContactForm

urlpatterns = patterns('',
    url(r'^contact/', ContactView.as_view(form_class=MyContactForm)),
)
```

Cookbook

Success and error messages

Starting from release 1.0, *envelope.views.ContactView* does not set any messages since these were customized by most users anyway. We encourage you to use the excellent django-braces app which provides a FormMessagesMixin designed specifically for this purpose.

The following example shows how to add the mixin to `ContactView`:

```
from braces.views import FormMessagesMixin
from envelope.views import ContactView

from django.utils.translation import ugettext_lazy as _


class MyContactView(FormMessagesMixin, ContactView):
    form_valid_message = _(u"Thank you for your message.")
    form_invalid_message = _(u"There was an error in the contact form.")
```

See the *customization section* on how to plug the subclassed view into your URLconf.

Check out Django messages documentation to make sure messages are enabled in your project.

Bootstrap integration

Embedding the contact form

From our personal experience with Bootstrap-powered websites, the easiest way to embed the contact form is to use django-crispy-forms. Install it with:

```
pip install django-crispy-forms
```

and add `crispy_forms` to `INSTALLED_APPS`. From there it's as simple as adding a `crispy` template tag to display the form. For example:

```
{% load envelope_tags crispy_forms_tags %}

...

<form action="{% url 'envelope-contact' %}" method="post">
    {% csrf_token %}
    {% antispam_fields %}
    {% crispy form %}
</form>
```

To add a submit button, create a custom form using `django-crispy-forms` helper:

```
# forms.py
from envelope.forms import ContactForm
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Submit


class MyContactForm(ContactForm):
    def __init__(self, *args, **kwargs):
        super(MyContactForm, self).__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.add_input(Submit('submit', 'Submit', css_class='btn-lg'))
```

And finally link this form to your view:

```
# views.py
from braces.views import FormMessagesMixin
from envelope.views import ContactView

from django.utils.translation import ugettext_lazy as _

from .forms import MyContactForm


class MyContactView(FormMessagesMixin, ContactView):
    form_invalid_message = _(u"There was an error in the contact form.")
    form_valid_message = _(u"Thank you for your message.")
    form_class = MyContactForm
```

or just use it in your urls.py if you directly reference *ContactView* `as_view()` method:

```
# urls.py
from django.conf.urls import patterns, url
from envelope.views import ContactView

from .forms import MyContactForm
```

```
urlpatterns = patterns('',
    url(r'^contact/', ContactView.as_view(form_class=MyContactForm)),
)
```

Displaying form messages nicely

GETting the contact form page after POSTing it will give you access to either a success message (form_valid_message) or an error message (form_invalid_message) thanks to django-braces' `FormMessagesMixin`. These messages use Django messages tag level so you can use the right Bootstrap class.

We recommend you first override Django's default message tags as following:

```
# settings.py
MESSAGE_TAGS = {
    messages.DEBUG: 'debug',
    messages.INFO: 'info',
    messages.SUCCESS: 'success',
    messages.WARNING: 'warning',
    messages.ERROR: 'danger' # 'error' by default
}
```

Then you can use Django's tip to display messages with Bootstrap CSS classes such as text-info or alert-warning:

```
{% if messages %}
    <ul class="messages">
        {% for message in messages %}
            <li {% if message.tags %} class="text-{{ message.tags }}"{% endif %}>
                {{ message }}
            </li>
        {% endfor %}
    </ul>
{% endif %}
```

Categorized contact form

Although the `category` field was removed from the default form class in 1.0, you can bring it back to your form using the following subclass:

```
from envelope.forms import ContactForm

from django import forms
from django.utils.translation import ugettext_lazy as _


class CategorizedContactForm(ContactForm):
    CATEGORY_CHOICES = (
        ('', _("Choose")),
        (10, _("A general question regarding the website")),
        # ... any other choices you can imagine
        (None, _("Other")),
    )
    category = forms.ChoiceField(label=_("Category"), choices=CATEGORY_CHOICES)

    def __init__(self, *args, **kwargs):
```

```
        """
        Category choice will be rendered above the subject field.
        """
        super(CategorizedContactForm, self).__init__(*args, **kwargs)
        self.fields.keyOrder = [
            'sender', 'email', 'category', 'subject', 'message',
        ]

    def get_context(self):
        """
        Adds full category description to template variables in order
        to display the category in email body.
        """
        context = super(CategorizedContactForm, self).get_context()
        context['category'] = self.get_category_display()
        return context

    def get_category_display(self):
        """
        Returns the displayed name of the selected category.
        """
        try:
            category = int(self.cleaned_data['category'])
        except (AttributeError, ValueError, KeyError):
            category = None
        return dict(self.CATEGORY_CHOICES).get(category)
```

Development

Contributing

Report bugs

Use the issue tracker on GitHub to file bugs.

Hack on the code

Fork the repository on GitHub, do your work in your fork (rhymes, eh?) and send me a pull request. Try to conform to **PEP 8** and make sure the tests pass (see below).

Running tests

Note: It is recommended to work in a virtualenv.

All dependencies required for running tests are specified in the file `test_requirements.txt`.

Note: If you get errors such as `ImportError: No module named mock` while running tests, you're probably on Python 2 (Python 3 has `mock` in standard library). To fix that, run `pip install mock`.

To get the tests up and running, follow these commands:

```
virtualenv envelope
cd envelope
source bin/activate
git clone https://github.com/zsiciarz/django-envelope.git
cd django-envelope
pip install -r test_requirements.txt
make test
```

Note: First three steps can be simplified by using virtualenvwrapper.

To get a coverage report, replace the last command with:

```
make coverage
```

CI Server

The GitHub repository is hooked to Travis CI. Travis worker pushes code coverage to coveralls.io after each successful build.

Reference

Views

class `envelope.views.`**`ContactView`**(***kwargs*)

Contact form view (class-based).

Displays the contact form upon a GET request. If the current user is authenticated, `sender` and `email` fields are automatically filled with proper values.

When the form is submitted and valid, a message is sent and afterwards the user is redirected to a "thank you" page (by default it is the page with the form).

`form_class` Which form class to use for contact message handling. The default (*`envelope.forms.ContactForm`*) is often enough, but you can subclass it if you want, or even replace with a totally custom class. The only requirement is that your custom class has a `save()` method which should send the message somewhere. Stick to the default, or its subclasses.

`form_kwargs` Additional kwargs to be used in the creation of the form. Use with *`envelope.forms.ContactForm`* form arguments for dynamic customization of the form.

`template_name` Full name of the template which will display the form. By default it is "envelope/contact.html".

`success_url` URL of the page with some kind of a "thank you for your feedback", displayed after the form is successfully submitted. If left unset, the view redirects to itself.

`form_class`

alias of `ContactForm`

`form_invalid`(*form*)

When the form has errors, display it again.

form_valid(*form*)
Sends the message and redirects the user to `success_url`.

get_initial()
Automatically fills form fields for authenticated users.

get_success_url()
Returns the URL where the view will redirect after submission.

`envelope.views.`**filter_spam**(*sender*, *request*, *form*, ***kwargs*)
Handle spam filtering.

This function is called when the `before_send` signal fires, passing the current request and form object to the function. With that information in hand, all available spam filters are called.

TODO: more spam filters

Forms

class `envelope.forms.`**ContactForm**(**args*, ***kwargs*)
Base contact form class.

The following form attributes can be overridden when creating the form or in a subclass. If you need more flexibility, you can instead override the associated methods such as *get_from_email()* (see below).

subject_intro Prefix used to create the subject line. Default is `settings.ENVELOPE_SUBJECT_INTRO`.

from_email Used in the email from. Defaults to `settings.ENVELOPE_FROM_EMAIL`.

email_recipients List of email addresses to send the email to. Defaults to `settings.ENVELOPE_EMAIL_RECIPIENTS`.

template_name Template used to render the (plaintext) email message. Defaults to `envelope/email_body.txt`.

html_template_name Template used to render the HTML email message. Defaults to `envelope/email_body.html`.

get_context()
Returns context dictionary for the email body template.

By default, the template has access to all form fields' values stored in `self.cleaned_data`. Override this method to set additional template variables.

get_email_recipients()
Returns a list of recipients for the message.

Override to customize how the email recipients are determined.

get_from_email()
Returns the from email address.

Override to customize how the from email address is determined.

get_subject()
Returns a string to be used as the email subject.

Override this method to customize the display of the subject.

get_template_names()
Returns a template_name (or list of template_names) to be used for the email message.

Override to use your own method choosing a template name.

save()
Sends the message.

Template tags

Add `{% load envelope_tags %}` to your template before using any of these.

envelope.templatetags.envelope_tags.**antispam_fields**()
Returns the HTML for any spam filters available.

envelope.templatetags.envelope_tags.**render_contact_form**(*context*)
Renders the contact form which must be in the template context.

The most common use case for this template tag is to call it in the template rendered by *ContactView*. The template tag will then render a sub-template `envelope/contact_form.html`.

Spam filters

envelope.spam_filters.**check_honeypot**(*request*, *form*)
Make sure that the hidden form field is empty, using django-honeypot.

Signals

before_send

Sent after the form is submitted and valid, but before sending the message.

Arguments:

sender View class.

request The current request object.

form The form object (already valid, so `cleaned_data` is available).

after_send

This signal is sent after sending the message.

Arguments:

sender Form class.

message An instance of `EmailMessage` that was used to send the message.

form The form object.

Changelog

1.3.0

- added Greek translation, thanks raratiru!
- Python 3.6 and Django 1.11 compatibility

1.2.0

- added Latvian and Russian translations, thanks wildd!
- added Spanish translations, thanks javipalanca!

1.1.0

- added Brazilian Portuguese translation, thanks aleprovencio!
- Python 3.5 and Django 1.9 compatibility

1.0.0

Improvements and fixes:

- HTML email support
- subject field is optional by default
- support for custom User model
- docs: added *Cookbook*

Backwards incompatible changes:

- removed category field from `ContactForm`
- `BaseContactForm` no longer exists; to customize form processing, subclass `ContactForm` directly
- `ContactView` does not create any flash messages; use FormMessagesMixin from django-braces (see the *Cookbook* for an example)
- dropped Django 1.4 compatibility
- dropped Python 2.6 compatibility; use 2.7 or 3.3+
- message rejection reason from signal handlers isn't sent to the user in HTTP 400 response's body
- the default `envelope/contact.html` template is removed; one must create the template explicitly

0.7.0

- added `{% render_contact_form %}` template tag
- Django 1.6 compatibility
- settled on 3.3 as the minimum supported Python 3 version
- moved to Travis CI as the continuous integration solution

0.6.1

- fixed `NameError` in example project

0.6.0

- Python 3 compatibility!

0.5.1

- fixed template loading in tests

0.5.0

- contact form class is more customizable
- the `Reply-To` header in the message is set to whatever the submitted email was
- added `after_send` signal
- django-honeypot is now just an optional dependency
- `example_project` is no longer incorrectly distributed with the application

0.4.1

- security bugfix regarding initial form values

0.4.0

- removed the function-based view
- removed `ContactForm.send()` method
- application signals (`before_send`)
- updated documentation
- reworked settings
- Continous Integration server, thanks to ShiningPanda

0.3.2

- omit the brackets if the user doesn't have a full name
- honeypot is mentioned in the usage docs

0.3.1

- configurable recipients
- better logging hierarchy
- the code is more PEP-8 compliant

0.3.0

- introduced a class-based *`envelope.views.ContactView`* (requires Django >= 1.3)
- deprecated the function-based view `envelope.views.contact`
- improved test coverage

- more and better documentation (also hosted on Read The Docs)

0.2.1

- French translation added

0.2.0

- deprecated the `ContactForm.send()` method, use *envelope.forms.ContactForm.save()* instead for more consistency with Django coding style
- localization support

0.1.4

- added a more descriptive README file

0.1.3

- added the `redirect_to` optional argument to view function

0.1.2

- added the `extra_context` argument to view function

0.1.1

- improved setup script, added dependencies

0.1.0

- initial version

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

envelope.forms, 21
envelope.spam_filters, 22
envelope.templatetags.envelope_tags, 22
envelope.views, 20

Index

A

antispam_fields() (in module envelope.templatetags.envelope_tags), 22

C

check_honeypot() (in module envelope.spam_filters), 22
ContactForm (class in envelope.forms), 21
ContactView (class in envelope.views), 20

E

envelope.forms (module), 21
envelope.spam_filters (module), 22
envelope.templatetags.envelope_tags (module), 22
envelope.views (module), 20

F

filter_spam() (in module envelope.views), 21
form_class (envelope.views.ContactView attribute), 20
form_invalid() (envelope.views.ContactView method), 20
form_valid() (envelope.views.ContactView method), 20

G

get_context() (envelope.forms.ContactForm method), 21
get_email_recipients() (envelope.forms.ContactForm method), 21
get_from_email() (envelope.forms.ContactForm method), 21
get_initial() (envelope.views.ContactView method), 21
get_subject() (envelope.forms.ContactForm method), 21
get_success_url() (envelope.views.ContactView method), 21
get_template_names() (envelope.forms.ContactForm method), 21

P

Python Enhancement Proposals
    PEP 8, 19

R

render_contact_form() (in module envelope.templatetags.envelope_tags), 22

S

save() (envelope.forms.ContactForm method), 22