django-contactme-form Documentation

Release 1.3.0

Daniel Rus Morales

May 27, 2017

Contents

1 **Demo projects** **3**
- 1.1 Demo quick setup . . . 3
- 1.2 Email settings . . . 3
- 1.3 Register a signal receiver . . . 4

2 **Tutorial** **5**
- 2.1 Installation . . . 5
- 2.2 Configuration . . . 5
- 2.3 Workflow . . . 6
- 2.4 Signals and receivers . . . 7

3 **Signals** **9**
- 3.1 Confirmation will be requested . . . 9
- 3.2 Confirmation has been requested . . . 9
- 3.3 Confirmation has been received . . . 9

4 **Templatetags** **11**
- 4.1 `render_contact_form` . . . 11

5 **Settings** **13**
- 5.1 `CONTACTME_MSG_MAX_LEN` . . . 13
- 5.2 `CONTACTME_SALT` . . . 13
- 5.3 `CONTACTME_NOTIFY_TO` . . . 13

6 **Templates** **15**

7 **Quick start** **17**

8 **Workflow in short** **19**

django-contactme provides a simple contact form that only hits the database when the user has visited the confirmation URL. Emails are threaded to avoid response blocking. It comes with a complete unittest set for both the backend functionality and the Jquery plugin.

Version 1.3 is compatible with:

- Django 1.9 under Python 2.7, 3.4, 3.5
- Django 1.8 under Python 2.7, 3.4, 3.5

Version 1.2 is compatible with:

- Django 1.8 under Python 2.7, 3.4
- Django 1.7 under Python 2.7, 3.4
- Django 1.6 under Python 2.7, 3.4
- Django 1.5 under Python 2.7, 3.4
- Django 1.4 under PYthon 2.7

Table of contents:

CHAPTER 1

Demo projects

The source package of django-contactme comes with several demo projects to see the application in action:

- **bare_demo** is the simplest demo possible.
- **bare_demo_with_ajax** is the same previous example plus Ajax functionality provided by `jquery.djcontactme.js`, the jquery plugin that comes with the application.
- **crispy_forms_demo** is an example of how to use django-contactme with django-crispy-forms.

Demo quick setup

Demo projects live inside the `example` project in app's root directory.

The simplest and less interfeing way to run the demo projects is by creating a virtualenv for django-contactme. Then:

1. `cd` into the any of the demo directories.
2. Run `python manage migrate` to create a minimal SQLite db for the demo.
3. Run `python manage runserver` and browse http://localhost:8000

In addition, **crispy_forms_demo** requires the **crispy_forms** package:

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

Email settings

By default the demo project send email messages to the standard output. You can customize the email settings to send actual emails.

Edit the `settings.py` module, go to the end of the file and customize the following entries:

```
EMAIL_HOST          = "" # for gmail it would be: "smtp.gmail.com"
EMAIL_PORT          = "" # for gmail: "587"
EMAIL_HOST_USER     = "" # for gmail: user@gmail.com
EMAIL_HOST_PASSWORD = ""
EMAIL_USE_TLS       = True # for gmail

DEFAULT_FROM_EMAIL  = "Your site name <user@gmail.com>"
SERVER_EMAIL        = DEFAULT_FROM_EMAIL

# Fill in actual EMAIL settings above, and comment out the
# following line to let the django demo sending actual emails
# EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'

CONTACTME_NOTIFY_TO = "Your name <user@gmail.com>"
```

The domain used in the links sent by email refers to *example.com* and thus are not associated with your django development web server. Change the domain name through the admin interface, sites application, to something like *localhost:8000* so that URLs in email messages match your development server.

Register a signal receiver

After trying the demo site you may like to add a receiver for any of the signals sent during the workflow.

Read the entry on *Signals* to know more about django-contactme signals. The section *Signals and receivers* in the Tutorial shows a use case.

CHAPTER 2

Tutorial

django-contactme is a simple reusable app. This is a tutorial as simple as the app itself.

Installation

Installing django-contactme is just a matter of checking out the package and adding it to your project or `PYTHONPATH`.

Use git, pip or easy_install to check out django-contactme from Github or get a release from PyPI:

1. Use **git** to clone the repository, and then install the package (read more about git):
   - `git clone git://github.com/danirus/django-contactme.git` and
   - `python setup.py install`
2. Or use **pip** (read more about pip):
   - Do `pip install django-contactme`
3. Or use **easy_install** (read more about easy_install):
   - Do `easy_install django-contactme`

Configuration

1. Add `'django_contactme'` to your `INSTALLED_APPS` setting.
2. Add `url(r'^contact/', include('django_contactme.urls'))` to your `urls.py`.
3. Create a `django_contactme` directory in your templates directory and copy the default templates from django-contactme into the new created directory.
4. Run `python manage.py migrate` that creates the `contactme_contact_msg` table.

Customization

1. Optionally you can add some settings to control django-contactme behaviour (see *Settings*), but they all have sane defaults.
2. Customize the templates (see *Templates*) in your `django_contactme` templates directory to make them fit in your design. Look at the *crispy_forms_demo* to see an example of templates customisation.

Workflow

Workflow described in 3 actions:

1. Get the Contact Form.

1. Render the Contact Form page. Omit this at will by using the `render-contact-form` templatetag (see *Templatetags*) in your own templates.

2. Post the Contact Form.

1. Check if there are *form security errors*. django_contactme forms are protected with `timestamp`, `security_hash` and `honeypot` field, following the same approach as the built-in Django Comments Framework. In case of *form security errors* send a 400 code response and stop.
2. Check whether there are other *form errors* (fields `name`, `email` and `message`) or whether the user clicked on the *preview* button. In such a case render the *Contact Form* again, with the *form errors* if any, and stop.
3. Send signal `django_contactme.signals.confirmation_will_be_requested`. If any receiver returns `False`, send a discarded contact message response to the user and stop.
4. Send a confirmation email to the user with a confirmation URL.
5. Send signal `django_contactme.signals.confirmation_requested`.
6. Render a *"confirmation has been sent to you by email"* template.

3. Visit the Confirmation URL.

1. Check whether the token in the confirmation URL is correct. If it isn't raise a 404 code response and stop.
2. Create a `ContactMsg` model instance with the message data secured in the URL.
3. Send signal `confirmation_received`. If any receiver return False, send a discarded contact message response to the user and stop.
4. Send an email to `settings.CONTACTME_NOTIFY_TO` addresses indicating that a new Contact Message has been received.
5. Render a *"your contact request has been received, thank you"* template.

Creating the secure token for the confirmation URL

The Confirmation URL sent by email to the user has a secured token with the contact form data. To create the token django-contactme uses the module `signed.py` authored by Simon Willison and provided in Django-OpenID.

`django_openid.signed` offers two high level functions:

- **dumps**: Returns URL-safe, sha1 signed base64 compressed pickle of a given object.
- **loads**: Reverse of dumps(), raises ValueError if signature fails.

A brief example:

```
>>> signed.dumps("hello")
'UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E'

>>> signed.loads('UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E')
'hello'

>>> signed.loads('UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E-modified')
BadSignature: Signature failed: QLtjWHYe7udYuZeQyLlafPqAx1E-modified
```

There are two components in dump's output `UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E`, separatad by a '.'. The first component is a URLsafe base64 encoded pickle of the object passed to dumps(). The second component is a base64 encoded hmac/SHA1 hash of "$first_component.$secret".

Calling signed.loads(s) checks the signature BEFORE unpickling the object -this protects against malformed pickle attacks. If the signature fails, a ValueError subclass is raised (actually a BadSignature).

Signals and receivers

The workflow mentions that django-contactme sends 3 signals:

1. **confirmation_will_be_requested**: Sent just before a confirmation message is requested.
2. **confirmation_requested**: Sent just after a confirmation message is requested.
3. **confirmation_received**: Sent just after a confirmation has been received.

See *Signals* to know more.

You may want to extend django-contactme by registering a receiver for any of this signals.

An example function receiver might check the datetime a user submitted a contact message and the datetime the confirmation URL has been clicked. If the difference between them is over 7 days the message could be discarded with a graceful *"sorry, too old message"* template.

Extending the demo site with the following code would do the job:

```
#-----------------------------------------
# append the code below to any demo project views.py module:

from datetime import datetime, timedelta
from django_contactme import signals

def check_submit_date_is_within_last_7days(sender, data, request, **kwargs):
    plus7days = timedelta(days=7)
    if data["submit_date"] + plus7days < datetime.now():
        return False
signals.confirmation_received.connect(check_submit_date_is_within_last_7days)


#-----------------------------------------------------
# change get_instance_data in django_contactme/forms.py to cheat a bit and
# make django believe that the contact form was submitted 7 days ago:

def get_instance_data(self):
    """
    Returns the dict of data to be used to create a contact message.
    """
    from datetime import timedelta                              # ADD THIS
    return dict(
        name        = self.cleaned_data["name"],
        email       = self.cleaned_data["email"],
        message     = self.cleaned_data["message"],
#        submit_date = datetime.datetime.now(),                    # COMMENT THIS
        submit_date = datetime.datetime.now() - timedelta(days=8), # ADD THIS
    )
```

Try the demo site again and see that the *django_contactme/discarded.html* template is rendered after clicking on the confirmation URL.

CHAPTER 3

Signals

List of signals sent by the django-contactme app.

Confirmation will be requested

django_contactme.signals.confirmation_will_be_requested Sent just before a confirmation message is requested.

A message is sent to the user right after the contact form is been posted and validated to verify the user's email address. This signal may be used to ban email addresses or check message content. If any receiver returns False the process is discarded and the user receives a discarded message.

Confirmation has been requested

django_contactme.signals.confirmation_requested Sent just after a confirmation message is requested.

A message is sent to the user right after the contact form is been posted and validated to verify the user's email address. This signal may be uses to trace contact messages posted but never confirmed.

Confirmation has been received

django_contactme.signals.confirmation_received Sent just after a confirmation has been received.

A confirmation is received when the user clicks on the link provided in the confirmation message sent by email. This signal may be used to validate that the submit date stored in the URL is no older than a certain time. If any receiver returns False the process is discarded and the user receives a discarded message.

See a simple example of a receiver for this signal: *Signals and receivers*, in the Tutorial.

CHAPTER 4

Templatetags

django-contactme has a templatetag to render the contact form.

`render_contact_form`

Many sites use a hidden div that fadeIn/slideUp when the user clicks on the **contact me/us** link. Use `render_contact_form` templatetag to render the contact form anywhere in your template. It uses the `django_contactme/form.html` template to render the form.

CHAPTER 5

Settings

This is the comprehensive list of settings django-contactme recognizes.

CONTACTME_MSG_MAX_LEN

Optional

This setting establish the maximum length of the message a user may write in the form.

An example:

```
CONTACTME_MSG_MAX_LEN = 3000
```

Defaults to 3000.

CONTACTME_SALT

Optional

This setting establish the ASCII string extra_key used by `signed.dumps` to salt the contact form hash. As `signed.dumps` docstring says, just in case you're worried that the NSA might try to brute-force your SHA-1 protected secret.

An example:

```
CONTACTME_SALT = 'G0h5gt073h6gH4p25GS2g5AQ25hTm256yGt134tMP5TgCX$&HKOYRV'
```

Defaults to an empty string.

CONTACTME_NOTIFY_TO

Optional

This setting establish the email address that will be notified on new contact messages. May be a list of email addresses separated by commas.

An example:

```
CONTACTME_NOTIFY_TO = 'Alice <alice@example.com>, Joe <joe@example.com>'
```

Defaults to `settings.ADMINS`.

CHAPTER 6

Templates

List of template files coming with django-contactme.

django_contactme/contactme.html Entry point for the django-contactme form. Template rendereded when visiting the `/contact/` URL. It makes use of the `render_contact_form` templatetag (see *Templatetags*).

django_contactme/form.html Used by the templatetag `render_contact_form` (see *Templatetags*).

django_contactme/preview.html Rendered either when the contact form has errors or when the user click on the `preview` button.

django_contactme/confirmation_email.txt Email message sent to the user when the contact form is clean, after the user clicks on the `post` button.

django_contactme/confirmation_sent.html Rendered if the contact form is clean when the user clicks on the `post` button and right after sending the confirmation email.

django_contactme/discarded.html Rendered if a receiver of the `confirmation_received` signal returns False. The signal `confirmation_received` is sent when the user click on the URL sent by email to confirm the contact message. See *Signals*.

django_contactme/accepted.html Rendered when the user click on the URL sent by email to confirm the contact message. If there are no receivers of the signal `confirmation_received` or none of the receivers returns False, the template is rendered and a `ContactMsg` model instance is created.

CHAPTER 7

Quick start

1. Add `django_contactme` to `INSTALLED_APPS`.
2. Add `url(r'^contact/', include('django_contactme.urls'))` to your root URLconf.
3. Run the `migrate` command to apply migrations.
4. Run the `runserver` command and check the new contact form at http://localhost:8000/contact/

CHAPTER 8

Workflow in short

The user...

1. Clicks on the *contact me/us* link of your site.
2. Fills in the contact form data with her `name`, `email address` and `message`, and clicks on *preview*.
3. She finally clicks on *post* and submit the form.
4. Then django-contactme:

- Creates a token with the contact form data.
- Sends an email to the user with a confirmation URL containing the token.
- And shows a template informing the user that she must click on the link to confirm the message.

5. The user receives the email, opens it, and clicks on the confirmation link.
6. Then django-contactme:

- Verifies the token and creates a `ContactMsg` model instance.
- Sends an email to the addresses listed in *CONTACTME_NOTIFY_TO*, to notify that a new contact message has reached the database.
- And finally shows a template being grateful for the message.

Read a longer workflow description in the *Workflow* section of the Tutorial.