django-emailhub Documentation

Release 0.0.4

Maxime Haineault

Dec 28, 2019

Contents:

Django EmailHub is a application that bring advanced email functionnalities to Django such as templates, batch sending and archiving.

Note: This is a work in progress and in early development stage.

CHAPTER 1

Getting started

1.1 Installation

The project is not stable enough yet to be on Pypy, to use it you will need to use the git repository:

```
pip install git+https://gitlab.com/h3/django-emailhub.git
```

Add *emailhub* to your project's settings:

```
INSTALLED_APPS = [
    ...
    'emailhub',
]
```

Run migrations:

```
(venv)$ python manage.py migrate
```

1.2 Configuration

In order to be able to log all outgoing emails, not just those sent from templates, it is necessary to use EmailHub's email backends:

```
EMAIL_BACKEND = 'emailhub.backends.smtp.EmailBackend'
# or
EMAIL_BACKEND = 'emailhub.backends.console.EmailBackend'
```

Refer to the *settings documentation* for all available backends.

CHAPTER 2

Settings

2.1 General configurations

EMAIL_BACKEND

In order to be able to log all outgoing emails, not just those sent from templates, it is necessary to use EmailHub's email backends.

```
EMAIL_BACKEND = 'emailhub.backends.smtp.EmailBackend'
```

They are essentially subclasses of the core django email backends.

Here's the conversion table for generic Django backends:

Django	EmailHub
`django.core.mail.backends.smtp.EmailBackend`	`emailhub.backends.smtp.EmailBackend`
`django.core.mail.backends.console.EmailBackend`	`emailhub.backends.console.EmailBackend`
`django.core.mail.backends.filebased.EmailBackend`	`emailhub.backends.filebased.EmailBackend`
`django.core.mail.backends.locmem.EmailBackend`	`emailhub.backends.locmem.EmailBackend`
`django.core.mail.backends.dummy.EmailBackend`	`emailhub.backends.dummy.EmailBackend`

2.1.1 Django-Anymail

Backends for django-anymail:

AnyMail	EmailHub
`anymail.backends.console.EmailBackend`	`emailhub.backends.anymail.console.EmailBackend`
`anymail.backends.mailgun.EmailBackend`	`emailhub.backends.anymail.mailgun.EmailBackend`
`anymail.backends.mailjet.EmailBackend`	`emailhub.backends.anymail.mailjet.EmailBackend`
`anymail.backends.mandrill.EmailBackend`	`emailhub.backends.anymail.mandrill.EmailBackend`
`anymail.backends.postmark.EmailBackend`	`emailhub.backends.anymail.postmark.EmailBackend`
`anymail.backends.sendgrid.EmailBackend`	`emailhub.backends.anymail.sendgrid.EmailBackend`
`anymail.backends.sendgrid_v2.EmailBackend`	`emailhub.backends.anymail.sendgrid_v2.EmailBackend`
`anymail.backends.sendinblue.EmailBackend`	`emailhub.backends.anymail.sendinblue.EmailBackend`
`anymail.backends.sparkpost.EmailBackend`	`emailhub.backends.anymail.sparkpost.EmailBackend`

2.1.2 Django-Secure-Mail

Backends for django-secure-mail:

Secure Mail	EmailHub
`secure_mail.backends.EncryptingSmtpEmailBackend`	`emailhub.backends.secure_mail.smtp.EmailBackend`
`secure_mail.backends.EncryptingConsoleEmailBackend`	`emailhub.backends.secure_mail.console.EmailBackend`
`secure_mail.backends.EncryptingFilebasedEmailBackend`	`emailhub.backends.secure_mail.filebased.EmailBackend`
`secure_mail.backends.EncryptingLocmemEmailBackend`	`emailhub.backends.secure_mail.locmem.Emailbackend`

2.1.3 Django-Celery-Email

Backends for django-celery-email:

Django Celery Email	EmailHub
`djcelery_email.backends.CeleryEmailBackend`	`emailhub.backends.djcelery_email.celery.EmailBackend`

EMAILHUB_DRAFT_MODE

Defaul: `True`

Activate or deactivate draft mode.

EMAILHUB_SEND_HTML

Default: `True`

Send also the HTML version with the text version of the email body (multi-parts)

EMAILHUB_PAGINATE_BY

Default: `20`

Pagination count used by EmailMessage ListView.

EMAILHUB_USER_LANGUAGE_DETECTION

Default: `True`

If set to `True`, templates rendering will be rendered with the user's language if it can be resoved. See *EMAILHUB_USER_LANGUAGE_RESOLVER* to see how language is resolved or customize it.

If set to `Fales`, the `settings.LANGUAGE_CODE` will be used to render email templats if no language is provided.

EMAILHUB_USER_LANGUAGE_RESOLVER

Default: `'emailhub.utils.i18n.guess_user_language'`

This is a function used to guess a user's preferred language according to common models patterns, **you should provide your own function to resolve the language**.

This is the default resolver:

```
def guess_user_language(user):
    if hasattr(user, 'profile') and hasattr(user.profile, 'language'):
        return user.profile.language
    elif hasattr(user, 'profile') and hasattr(user.profile, 'lang'):
        return user.profile.lang
    elif hasattr(user, 'language'):
        return user.profile.language
    elif hasattr(user, 'lang'):
        return user.lang
    elif hasattr(settings, 'LANGUAGE_CODE'):
        return settings.LANGUAGE_CODE.split('-')[0]
    else:
        return 'en'
```

This what your custom resolver should look like:

```
def my_custom_resolver(user):
    return user.customerprofile.lang
```

2.2 Outgoing emails

EMAILHUB_DEFAULT_FROM

Default: `'no-reply@domain.com'`

If email_from isn't specified when sending the email or if the template does not provide a value for it, this setting is used.

EMAILHUB_SEND_BATCH_SLEEP

Default: `2`

Sleep N seconds between sending each batches

EMAILHUB_SEND_BATCH_SIZE

Default: `20`

Limit the number of Email objects will be sent

EMAILHUB_SEND_MAX_RETRIES

Default: `3`

Maximum send retries before giving up.

2.3 Email templates

EMAILHUB_PRELOADED_TEMPLATE_TAGS

Default:

```
[
    'i18n',
]
```

These template tags will be preloaded for email templates rendering.

EMAILHUB_TEXT_TEMPLATE

Default: `"""{{% load {template_tags} %}}{content}"""`

Template used to render text email templates

EMAILHUB_HTML_TEMPLATE

Default:

```
{{% load {template_tags} %}}
{{% language lang|default:"en" %}}
<!DOCTYPE html>
<html lang="{{ lang }}">
<head><meta charset="utf-8"></head>
<body>{content}</body>
</html>
```

Template used to render HTML email templates

CHAPTER 3

Python API

3.1 Sending from Python

You can send email in two ways, first the conventional way described in the Django documentation and second by using EmailHub's template feature.

The conventional method is suited for email for which you don't need editable templates or draft mode. The EmailHub email backend handle the linking with users if a user email is found in the `to`, `cc` or `bcc` fields.

The EmailHub template method uses the `EmailFromTemplate` class to create an email instance from a template:

```
from emailhub.utils.email import EmailFromTemplate

msg = EmailFromTemplate(
    'template-slug-name', lang='en').send_to(user)
```

With custom context variables:

```
from emailhub.utils.email import EmailFromTemplate

msg = EmailFromTemplate('template-slug-name',
        lang='en',
        extra_context={
            'somevar': some_var,
            'someothervar': some_other_var,
        }).send_to(user)
```

At this point the message isn't really sent, it is wither in `draft` or in `pending` state. You can the email right away by calling `send`.

You can force send a message like so:

```
msg.send(force=True)
```

If the `force` argument is `False` (default) it will just mark the email as pending so it will be sent in batch with the cron job.

3.2 Email states

Email messages are always in one of the following state:

- **draft**: message is still editable, will not be sent.
- **pending**: message is waiting to be sent (via cron job)
- **locked**: message is being sent, will result in either `sent` or `error` state.
- **sent**: message has been sent (is an archive)
- **error**: message has been sent, but the server returned an error. Sending will be retried until `EMAILHUB_SEND_MAX_RETRIES` has been reached.

```
>>> print(msg.state)
'draft'
>>> print(msg.is_draft)
True
>>> print(msg.is_pending)
False
>>> print(msg.is_locked)
False
>>> print(msg.is_sent)
False
>>> print(msg.is_error)
False
```

CHAPTER 4

Management commands

4.1 Create template

```
(venv)$ python manage.py emailhub --create-template
```

4.2 Diff

Performs a diff between a JSON fixture file and template present in database.

```
(venv)$ python manage.py emailhub --diff emailtemplates-backup.json
```

By default only changed field are shown, to get a full word level diff you can set verbosity at 2.

```
(venv)$ python manage.py emailhub --diff emailtemplates-backup.json -v2
```

4.3 Dump

Dumps specific email templates specified by slug in JSON format.

Multiple slug can be passed using comas as separators.

```
(venv)$ python manage.py emailhub --dump request-accepted,request-received
```

4.4 Dump all

Dumps all email template in JSON format.

4.5 List templates

List available templates and their corresponding translations.

```
(venv)$ python manage.py emailhub --list-templates

request-accepted
  - EN) [{{ site }}] We have accepted your request!
  - FR) [{{ site }}] Nous aons accepté votre demande!

request-received
  - EN) [{{ site }}] New request from {{ user.get_fullname }}
  - FR) [{{ site }}] Nouvelle demande de {{ user.get_fullname }}
```

4.6 Send

Send unsent emails:

```
(venv)$ python manage.py emailhub --send
```

Since email batch sending is throttled, you can set up a cron job to run every minutes to send unsent emails. This way an email will never wait more than one minute before being sent in a optimal situation.

```
*  * * * * root /venv/path/bin/python /project/path/manage.py emailhub --send >> /var/
↪log/emailhub.log 2>&1
```

4.7 Send test

Sends an email test, accepts a destination email address or a user ID.

```
(venv)$ python manage.py emailhub --send-test bob@test.com
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Subject: Test email
From: no-reply@domain.com
To: bob@test.com
Date: Tue, 06 Mar 2018 19:01:16 -0000
Message-ID: <20180306190116.2915.53062@singularity>

This is a test.
-------------------------------------------------------------------------------
```

4.8 Status

```
(venv)$ python manage.py emailhub --status

    Unsent                      14
    Drafts                      7
```

(continues on next page)

(continued from previous page)

```
Is sent                         7
Errors                          0
```

CHAPTER 5

Templates

Email templates use Django's built-in template renderer, which means you can load and use any available templatetags.

5.1 Base context

The base context contains global variables.

CHAPTER 6

Generic Views

Generic views can be used as is or be subclassed for customization.

Example:

```
from emailhub.views import InboxListView


class MessageIndexView(InboxListView):
    """
    EmailMessage inbox view
    """
    pass
```

6.1 InboxListView

The `InboxListView` view is used to create an Inbox view for the email recipient (must be a registred used).

6.2 EmailMessageDetailView

The `EmailMessageDetailView` view is used to display a specific message.

The message state must be in `['locked', 'sent', 'error']`.

6.3 EmailMessageUpdateView

The `EmailMessageUpdateView` view is used to edit a specific message.

The message state must be `draft`.

6.4 process_message

Used manipulate message via ajax calls.

Actions:

- send
- delete

CHAPTER 7

Signals

7.1 on_email_process

Sent when an `EmailMessage` is created from an outgoing email.

```
from emailhub.signals import on_email_process

on_email_process.connect(email_process_callback,
                         dispatch_uid="emailhub.on_email_process")
```

7.2 on_email_out

Sent when an email is going out.

```
from emailhub.signals import on_email_out

on_email_out.connect(email_out_callback, dispatch_uid="emailhub.on_email_out")
```

CHAPTER 8

Integration

8.1 Django REST Regristration

The django-rest-registration is not really flexible in term of template engine for emails.

There are however workarounds.

8.1.1 Registration view

The registration view can use emailhub template if you disable *REGISTER_VERIFICATION_ENABLED* and use signal to send the notification instead:

```
# settings.py
REST_REGISTRATION = {
  'REGISTER_VERIFICATION_ENABLED': False,
}
```

```
# signals.py
from django.conf import settings
from django.dispatch import receiver

from rest_registration.api.views.register import RegisterSigner
from rest_registration.utils.users import get_user_verification_id
from emailhub.utils.email import EmailFromTemplate

@receiver(user_registered, sender=None)
def user_registred(sender, **kwargs):
    user, request = kwargs.get('user'), kwargs.get('request')
    with transaction.atomic():
        signer = RegisterSigner({
            'user_id': get_user_verification_id(user),
        }, request=request)
        context = {
```

(continues on next page)

(continued from previous page)

```
            'site': settings.FRONTEND_URL.split('//').pop(-1),
            'site_url': settings.FRONTEND_URL,
            'params_signer': signer,
            'verification_url': settings.REST_REGISTRATION.get(
                'REGISTER_VERIFICATION_URL'),
        }
        EmailFromTemplate(
            'register-verify', extra_context=context,
            lang=user.language).send_to(user)
```

```
# views.py
from django.conf import settings

from rest_registration.decorators import api_view_serializer_class_getter
from rest_registration.settings import registration_settings
from rest_registration.notifications.enums import NotificationType
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import AllowAny
from rest_registration.utils.responses import get_ok_response
from rest_registration.utils.users import get_user_verification_id
from rest_registration.exceptions import UserNotFound
from rest_registration.api.views.reset_password import ResetPasswordSigner
from emailhub.utils.email import EmailFromTemplate


@api_view_serializer_class_getter(
    lambda: registration_settings.SEND_RESET_PASSWORD_LINK_SERIALIZER_CLASS)
@api_view(['POST'])
@permission_classes([AllowAny])
def send_reset_password_link(request):
    '''
    Send email with reset password link.
    '''
    if not registration_settings.RESET_PASSWORD_VERIFICATION_ENABLED:
        raise Http404()
    serializer_class = registration_settings.SEND_RESET_PASSWORD_LINK_SERIALIZER_
↪CLASS  # noqa: E501
    serializer = serializer_class(
        data=request.data,
        context={'request': request},
    )
    serializer.is_valid(raise_exception=True)
    user = serializer.get_user_or_none()
    if not user:
        raise UserNotFound()
    signer = ResetPasswordSigner({
        'user_id': get_user_verification_id(user),
    }, request=request)

    EmailFromTemplate(
        'password-reset', extra_context={
            'site': settings.FRONTEND_URL.split('//').pop(-1),
            'site_url': settings.FRONTEND_URL,
            'params_signer': signer,
            'verification_url': signer.get_url(),
            'user': user,
```

(continues on next page)

(continued from previous page)

```
        }, lang=user.language).send_to(user)

    return get_ok_response('Reset link sent')
```

CHAPTER 9

Overview

9.1 Inboxes

Not actual inboxes, but emails are (optionally) linked to users model.

This makes it possible to build an inbox view for users where they can see a copy of all emails sent to them.

This is accomplished in two ways, first when using the EmailHub API:

```
EmailFromTemplate('welcome-message').send_to(user)
```

And when using EmailHub's email backends, it will look for user emails that matches the destination email and link them.

9.2 Batch sending

Sending email right away is rarely a good idea, having a batch sending approach prevents many headaches down the road.

It won't hang your frontend process if the SMTP is slow to respond.

It allows to have throttling rules to avoid flooing the SMTP.

Finally, it allow to introduce the draft state feature.

9.3 Draft state

The draft mode works somewhat like standard email draft, but with automated emails.

When a template email is sent and draft mode is enabled, the email isn't sent right away. It is only saved in db where it can be edited and sent at a later time.

This allows to create new email templates and review / correct outgoing emails before they are actually sent to actual customers.

When the template is stable, draft mode can be disabled and be sent directly.

9.4 Email templates

Email templates are can be defined in the admin. They support:

- translations
- variables (they are actual django templates)
- preset signatures
- overriding default send from email
- allow or block draft mode
- django-material theme

They also integrate CodeMirror to highlight template variables and HTML:

Note: This screenshot is with the Django-material theme.

9.5 Generic views

EmailHub provide generic views for `EmailMessage` for viewing, editing and listing. You can consult the *Generic Views documentation here*.

9.6 Signature templates

Email templates can (or not) use signature templates defined in the admin.

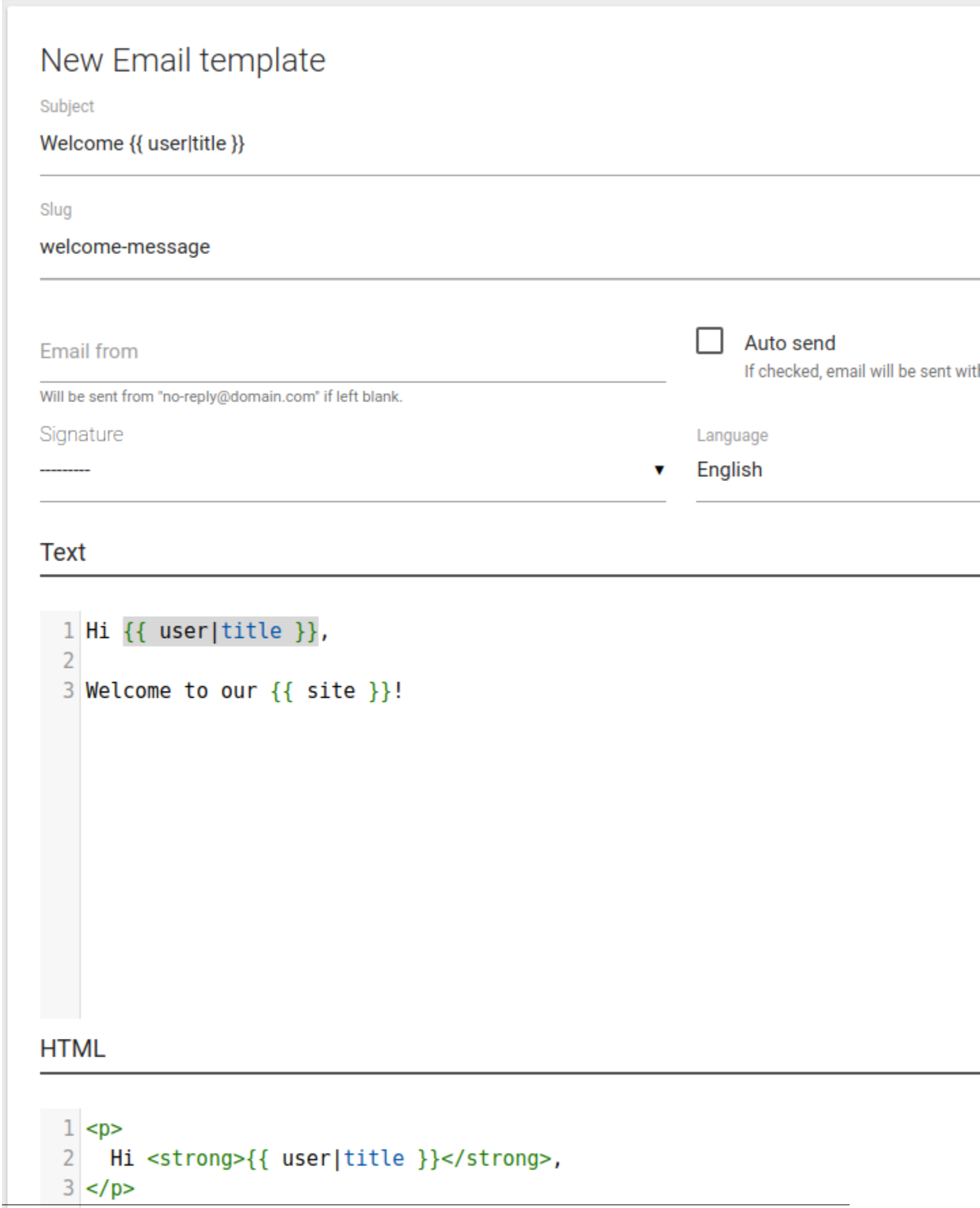
9.7 Ecosystem

Django EmailHub tries to stay compatible with the followings apps:

- django-material
- django-anymail (refer to backend settings for integration)
- django-herald
- django-secure-mail

Don't hesitate to create a pull request to integrate you django app, I will gladly merge it!

You might also want to consider django-post-office which has overlapping features with EmailHub.



New Email template

Subject

Welcome {{ user|title }}

Slug

welcome-message

Email from

Will be sent from "no-reply@domain.com" if left blank.

Auto send

If checked, email will be sent wit

Signature

Language

English

Text

```
Hi {{ user|title }},

Welcome to our {{ site }}!
```

HTML

```
<p>
  Hi <strong>{{ user|title }}</strong>,
</p>
<p>
  Welcome to our <a href="{{ site_url }}">{{ site }}</a>!
</p>
```

CHAPTER 10

Indices and tables

- genindex
- modindex
- search