Django Mail Builder Documentation

Release 0.3

Curtis Maloney

March 07, 2016

Build email messages from templates.

Contents:

CHAPTER 1

Overview

The EmailMessage (https://docs.djangoproject.com/en/1.9/topics/email/#the-emailmessage-clas class in Django accepts a number of arguments, and when you're sending emails from your app, you need to supply them. Sometimes this is simple, as the values are fixed (i.e. from_email), whilst others are more dynamic (such as the message body).

One way or another you need to produce these values and pass them to the class.

And sometimes you want `EmailMultipartMessage` instead, for instane when you want to send plain text _and_ html.

The `build_message` function helps you do this by letting you pass arguments, as well as use blocks from a template to render others.

1.1 The Arguments

As shown in the Django docs, the `EmailMessage` class takes the following arguments:

- subject
- body
- from_email
- to
- bcc
- connection
- attachments
- headers
- cc
- reply_to

The `build_message` function will accept and and all of them as keyword arguments, but will also try to render any blocks with those names from the provided template and update the values from there.

Any mix of keyword and template supplied arguments is valid, as long as there are enough to satisfy the `EmailMessage` class.

The unsent message instance is returned, so you can update fields, override them, add attachments or headers, or anything else you like before sending.

CHAPTER 2

Templates

Templates are used to supply blocks, which are rendered to provide values to be passed to the `EmailMessage` constructor.

2.1 Blocks

The following blocks are used:

- subject
- from_email
- body
- html
- to
- cc
- bcc
- reply_to

For the meaning of these fields (except `html`), see the *EmailMessage* docs.

All values are stripped, and the email address fields (*to*, *cc*, *bcc*, and *reply_to*) are split on newlines into a list, then stripped.

Reference

`mail_builder.`**`build_message`**(*template_names*, *extra_context=None*, *force_multipart=False*, ***defaults*)

Constructs a `EmailMessage` using the template to provide arguments.

Parameters

- **`template_names`** (*`sequence`*) – A list of template names to pass to `select_template`. If a single string is passed, it will be wrapped in a list
- **`extra_context`** (*`dict`*) – Extra context to pass to the template blocks.
- **`force_multipart`** (*`bool`*) – Ensure a `EmailMultipartMessage` is built, even when no *hmtl* content is provided.
- **`defaults`** (*`varied`*) – All extra arguments will be passed to the `EmailMessage`

Returns `EmailMessage` instance.

class `mail_builder.views.`**`EmailFormMixin`**

A mixin intended for `FormView` which renders and sends an email on form valid.

`email_template`

The value to pass as *email_templates* to *build_message*

`fail_silently`

(Default: True)

Passed to `EmailMessage.send`

`email_kwargs`

(Default: {})

Arguments to pass when calling *build_message*

`get_email_context`(*form*, ***kwargs*)

Hook to build the context to be used when rendering email template blocks. The default implementation will return `kwargs`, after setting 'form' to the form's `cleaned_data`, if it's not set.

`get_email_kwargs`(*form*, ***kwargs*)

Builds the dict of keyword arguments to pass to *build_message*.

The default implementation updates *kwargs* from `self.email_kwargs`.

`form_valid`(*form*)

Calls *self.get_email_context* and *self.get_email_kwargs*, then builds a message using *build_message*. Then calls `send(fail_silently=self.fail_silently)` on the message. Finally calls the super-class's `form_valid` method.

CHAPTER 4

Quick Start

1. Install the package

```
$ pip install django-mail-builder
```

2. Write a template

```
{% block subject %}Thanks for signing up to Awesome Site!{% endblock %}
{% block to %}{{ user.email }}{% endblock %}
{% block body %}
Thanks for joining our site!

We hope you love how awesome it is!
{% endblock %}
{% block html %}
<h1> Thanks for joining our site! </h1>

<p> We hope you love how awesome it is!
{% endblock %}
```

3. In your view, build the message and send it.

```
msg = build_message('email/welcome.email', {'user': request.user})
msg.send()
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index