django-email-pal Documentation

Release 0.0.1

18F

Jun 12, 2017

Contents:

CHAPTER 1

Introduction

django-email-pal attempts to solve a number of issues we've had when iterating on projects at 18F.

"What kinds of emails are we sending out?"

Throughout the life of a project, designers and content authors often have a difficult time figuring out what kinds of emails their project sends out, because they're not easily discoverable or well-documented.

django-email-pal attempts to solve this by providing a built-in "example email gallery" that showcases example versions of every kind of email that can be sent out, in both HTML and plaintext formats, with accompanying documentation.

This gallery also makes it very easy to *iterate* on the content and design of an email: instead of constantly sending oneself an email through the app, a designer can just make changes to a template and see the immediate effects in their browser, iterating on it just as they'd do with any other Django view.

"Wait, we have to write our email's HTML like it's 1995?"

Simply getting started with HTML email can be daunting due to the lack of standards and wide variety of email clients one must support. It should be easy to send HTML emails that look nice, without having to figure out a bunch of arcane tricks to ensure they don't break on popular email clients.

Solutions to this problem are varied. django-email-pal gives you the freedom to choose your own HTML email framework if you need to, but it also comes with a nice solution out-of-the-box: Lee Munroe's Really Simple Responsive HTML Email Template.

"There are people who can't read HTML email?"

Email is often designed in HTML form first, with the plaintext alternative left as an afterthought.

Some projects just strip the HTML tags to generate the plaintext. However, this doesn't always result in an intelligible email, as there may have been buttons or hyperlinked phrases in the original HTML version that no longer make much sense.

Other projects have completely separate HTML and plaintext versions that are specified independently of one another. Yet this is cumbersome too, as both versions are *mostly* identical and can easily get out of sync.

django-email-pal attempts to solve this by providing a simple two-pass template rendering strategy that allows both the HTML and plaintext versions of an email to be generated from the same template. This allows *most* of the content to be reused, but also allows for minor modifications based on the email's content type.

CHAPTER 2

Quick start guide

Prerequisites

- You'll need Django 1.11.1 or later.
- Your project needs to use either Django's default `DjangoTemplates` or `Jinja2` template engine.
- Your project needs to use Python 3.5 or later.

Installation

This package isn't on PyPI yet, so you'll need to install it directly from GitHub for now:

```
pip install git+git://github.com/18F/django-email-pal
```

Required settings

Add `emailpal.apps.EmailPalConfig` to your `INSTALLED_APPS` setting, e.g.:

```
INSTALLED_APPS = (
    # ...
    'emailpal.apps.EmailPalConfig',
    # ...
)
```

Then add the following to your project's `urls.py`:

```
from django.conf.urls import include, url

urlpatterns = [
    # ...
```

```
    url(r'^examples/', include('emailpal.urls')),
    # ...
]
```

This sets up the email example gallery at `/examples/` on your app. You can change it to something else if you want, or you can hide it behind some logic if you only want it to be exposed during development.

Your first email

Let's get started by adding an email example to your project. We're going to assume that your project has an app called `example` in it.

Create a file at `example\emails.py` and put the following in it:

```
from emailpal import SendableEmail


class MySendableEmail(SendableEmail):
    """
    This is a simple example email.
    """

    template_name = 'example/my_template.html'
    subject = 'Check this out, {full_name}!'
    example_ctx = {'full_name': 'Jane Doe'}
```

Then create a file at `example\templates\example\my_template.html` and put this in it:

```
{% extends "emailpal/really_simple/base.html" %}

{% block content %}
<p>Hello {{ full_name }},</p>

<p>This is a simple email which uses Lee Munroe's <em>Really Simple
Responsive HTML Email Template</em> to be easily viewable across a
wide range of mail clients.</p>

{% include "emailpal/really_simple/cta.html" with action="learn more about the␣
↪template" url="https://github.com/leemunroe/responsive-html-email-template" %}

<p>Hopefully it will be useful.</p>
{% endblock %}
```

Important: If you're using Jinja2, you'll want to put the template at `example\jinja2\example\my_template.html`.

Also, replace the line containing the `{% include %}` directive with the following:

```
{% with action="learn more about the template",
        url="https://github.com/leemunroe/responsive-html-email-template" %}
{% include "emailpal/really_simple/cta.html" %}
{% endwith %}
```

As you can probably guess, the email expects the context variable `full_name` to contain the full name of the recipient. The example version of the email will use "Jane Doe".

The email will also contain a call-to-action (CTA) that directs the user to a website.

Registering the email with the gallery

Now we just need to let the email example gallery know about the existence of your new template. Do this by adding the following to your project's `settings.py`:

```
SENDABLE_EMAILS = [
    'example.emails.MySendableEmail',
]
```

Now you're set! Start your app and visit `/examples/`; you should see the email gallery with a single entry, and be able to view your example email as HTML and plaintext.

Sending the email

You can create a Django `EmailMessage` with your email's *`create_message()`* method like so:

```
msg = MySendableEmail().create_message(
    {'full_name': 'boop jones'},
    from_email='foo@example.org',
    to=['bar@example.org'],
    headers={'Message-ID': 'blah'},
)
```

Then you can send the message with `msg.send()`.

Adding smoke tests

Since your email has an example context, it's straightforward to add smoke tests for it: just render the email with the example context and make sure nothing explodes. In fact, django-email-pal comes with tooling that makes this particularly easy.

Just create a new test module and add the following to it:

```
from unittest import TestCase
from emailpal import EmailSmokeTestsMixin


class EmailTests(TestCase, EmailSmokeTestsMixin):
    pass
```

Now when you run `manage.py test` (or whatever your choice of test runners is), all the emails you've listed in `settings.SENDABLE_EMAILS` will be rendered with their example context to ensure that they don't throw any exceptions.

CHAPTER 3

Reference

Sendable emails

class emailpal.**SendableEmail**

This abstract base class represents a template-based email that can be sent in HTML and plaintext formats.

When generating the email, the template is actually rendered *twice*: once as HTML, and again as plain text. As explained in *"There are people who can't read HTML email?"*, this allows both formats to share most of their content, yet also deviate where necessary.

So, aside from the context your code provides, the following context variables are provided when rendering your template:

- `is_html_email` is `True` if (and only if) the template is being used to render the email's HTML representation.
- `is_plaintext_email` is `True` if (and only if) the template is being used to render the email's plaintext representation.

Note that when rendering the email as plaintext, HTML tags are automatically stripped from the generated content.

create_message(*ctx: T*, *from_email=None*, *to=None*, *bcc=None*, *connection=None*, *attachments=None*, *headers=None*, *alternatives=None*, *cc=None*, *reply_to=None*) → django.core.mail.message.EmailMessage

Creates and returns a `django.core.mail.EmailMessage` which contains the plaintext and HTML versions of the email, using the context specified by `ctx`.

Aside from `ctx`, arguments to this method are the same as those for `EmailMessage`.

example_ctx

An example context with which the email can be rendered.

subject

The subject line of the email. This is processed by `str.format()` and passed the same context that is passed to templates when rendering the email, so you can include context variables via brace notation, e.g. `"Hello {full_name}!"`.

template_name
The path to the template used to render the email, e.g. `"my_app/my_email.html"`.

Really simple template

This package comes with an optional template based on Lee Munroe's Really Simple Responsive HTML Email Template that makes it easy to get started with sending HTML emails that look nice.

For an example of this template in use, see *Your first email*.

Base template

Emails can use the base template by extending `emailpal/really_simple/base.html`.

Variables used

This template has no special variables aside from the ones you include in your context and the ones defined by *`emailpal.SendableEmail`*.

Blocks defined

These can be overridden by templates that inherit from the base. Unless otherwise stated, all blocks default to empty content.

content The content for the email.

preheader The contents of a `<span>` with a `class` of `preheader`, which some email clients will show as a preview.

title The HTML title of the email (not the subject line).

footer The footer of the email.

Call-to-action (CTA)

CTAs can be included via the `emailpal/really_simple/cta.html` template.

Here's an example of using the CTA with Django templates:

```
{% include "emailpal/really_simple/cta.html" with action="view the site" url="https://
↪example.org" %}
```

And here's the equivalent in Jinja2:

```
{% with action="view the site", url="https://example.org" %}
  {% include "emailpal/really_simple/cta.html" %}
{% endwith %}
```

In the HTML version of the email, the above snippet will appear as a large button with the text "View The Site" on it; clicking the button will take the user to example.org.

In the plaintext version of the email, the snippet will appear like this:

```
To view the site, visit:
https://example.org
```

Variables required

`action` The human-readable name of the action the reader is being asked to take, e.g. `"view the website"`.

`url` The URL the user should visit to take the action.

CHAPTER 4

Developing django-email-pal

Important: This section is about developing django-email-pal itself, not using it in your Django project. For details on the latter, see the *Quick start guide*.

First, clone the git repository:

```
git clone https://github.com/18F/django-email-pal
```

Then create a virtualenv for the project and install development dependencies:

```
virtualenv -p python3 venv
source venv/bin/activate
pip install -r requirements-dev.txt
```

Then install django-email-pal in development mode:

```
python setup.py develop
```

Running the example app

An example Django project provides basic integration with django-email-pal. It can be used to manually ensure that everything works as expected.

To use it, run the following from the root of the repository:

```
cd example
python manage.py migrate
python manage.py runserver
```

At this point you should be able to visit the locally-hosted project.

Running tests

You can run all the tests with code coverage:

```
pytest
```

You can also ensure that there aren't any linting errors:

```
flake8
```

To run all tests, linters, and other automated QA against all supported runtimes and dependencies, run:

```
tox
```

Writing documentation

If you want to work on documentation, you can run the development documentation server with:

```
python setup.py devdocs
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index