CODING NOMADS

13) Sending Emails & Email Verification Lesson

Sending Emails with Python + Flask-Mail

20 min to complete · By Brandon Gigous

Contents

While email was invented decades ago, it's still around and in use today more than ever. The technology behind most email sending and receiving is called Simple Mail Transfer Protocol (SMTP). To keep your life simple, you don't have to get too technical with sending emails via your app. *Behold!* Flask-Mail is an extension that does most of that work for you.

In this lesson, you'll install and configure Flask-Mail to be able to send email through Python.

Install Flask-Mail

To install it, yet another simple pip command:

```
(env) $ pip install flask-mail
```

To add email support to your app, you'll only need two things:

1. An SMTP server
2. Settings configured for that server

As for the SMTP server, you *could* host one yourself. However, for development, it's *much* easier to use an external server. That's why in the previous lesson, you set up a secondary Gmail account so you can start playing around with emails right away.

Now let's configure your Flask-Mail config settings. In no time, you'll be sending emails from your app left and right. Ready?

The Configuration Settings

Alright, so what exactly do you need to do for the SMTP configuration stuff? There's plenty of settings, but this lesson will guide you through it all. In particular, Flask-Mail will need to have these configuration keys set:

- `MAIL_SERVER` : The hostname or IP of the email server. Default: `localhost`
- `MAIL_PORT` : The port of the email server. Default: 25
- `MAIL_USE_TLS` : To enable Transport Layer Security (TLS), or not to enable. That is the question. Default: `False`
- `MAIL_USE_SSL` : To enable Secure Sockets Layer (SSL), or not to enable. That is the other question. Default: `False`
- `MAIL_USERNAME` : Mail account username
- `MAIL_PASSWORD` : Mail account (app) password

Settings For Your Gmail Account

Note: See a previous lesson for how to enable SMTP on a Gmail account, be careful, though, you may not want to do this on your personal account.

To enable email support using your new Gmail account, you'll have to set most of the settings above. What better place than your `config.py` file? To take advantage of inheritance, you can place these in your base `Config` class.

For your new Gmail account, you can set your configuration settings like this:

```
class Config():
    # ...

    # Flask-Mail config
    MAIL_SERVER = 'smtp.gmail.com'
    MAIL_PORT = 587
    MAIL_USE_TLS = True
    MAIL_USERNAME = os.environ.get('MAIL_USERNAME')
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')

    # Other email settings
    RAGTIME_ADMIN = os.environ.get('RAGTIME_ADMIN')
    RAGTIME_MAIL_SUBJECT_PREFIX = 'Ragtime —'
    RAGTIME_MAIL_SENDER = 'Ragtime Admin <ragtime.flask@gmail.com'
```

The first three configuration key settings come straight from the horse's mouth, that is, from Google. Next is the username you signed up with, then your **app password** that you wrote down in the last lesson. These settings are taken from environment variables only since, y'know, it's pretty sensitive stuff. You can set them in the CLI like so:

```
export MAIL_USERNAME=<your Gmail username>
export MAIL_PASSWORD=<your Gmail app password>
```

Info: If you haven't already, now might be a good time to create a new file to instantiate your various environment variables.

The last three configuration key settings are for your app. They are more for convenience and not required for Flask-Mail, but configuration files are good at that, convenience. The first one `RAGTIME_ADMIN` is the email account of the administrator of the Flask app. That just means it's the email that gets *notifications* about various things that occur within your app, like when a new user signs up or when the server catches on fire. Yours doesn't have to have `RAGTIME` in the name, of course.

I cannot directly access or modify the content of images hosted on external servers or paths that are relative and outside the scope of available resources, such as "../images/server_on_fire.png". For a specific task like replacing an image link, I would need a full URL to the original image to provide an accurate replacement syntax. However, based on your instructions, here's how the output would look if the original image URL was provided:

The `RAGTIME_MAIL_SUBJECT_PREFIX` is just that, the subject prefix for when the app sends an email. When the app sends email to users, it will be *from* whatever you set the value of `RAGTIME_MAIL_SENDER` to.

Mentorship Makes the Difference!

You don't have to learn alone. Join the CodingNomads' Mentorship Bootcamp Program and get:

- A team of mentors and advisors dedicated to your success
- Weekly 1-on-1 screen share meetings with your professional mentor
- Constant, personal, in-depth support 7 days a week via Discord
- Accountability, feedback, encouragement, and success

Get Mentorship

Initializing Flask-Mail

To initialize Flask-Mail, you must place your lit candles on each point of the pentagram, then recite a blessing of Cthulhu. Just kidding, all you need to do is the usual extension initialization. Here's what you'll add in your app package's `__init__.py`.

```
from flask_mail import Mail
mail = Mail()

def create_app(config_name):
    # ...
    mail.init_app(app)
```

You've just configured Flask-Mail to help you send emails from your app. Now the next step? Sending emails from your app!

Sending Emails from Flask

Back in the early 1860's, the Pony Express operated in the United States to deliver all kinds of messages and mail, by horse, from California all the way to Missouri. At the time, it was a godsend because it reduced the amount of time for messages to be sent across the country. But today, you don't even need a pony. All you need is Flask-Mail, an email account, and a few lines of code to deliver thousands of messages a day. Knowing that, no one needs a pony.

Functional Email Sending

"Functional" meaning you'll encapsulate your email-sending functionality in a function! (There I go again with too many variations of the same word in one sentence...) Anyway, make a new file named `email.py` in your `app` package (`app/email.py`). Here's the code to make a convenient email-sending function called, what else? `send_email()` :

```
from flask import current_app, render_template
from flask_mail import Message
from . import mail

def send_email(to, subject, template, **kwargs):
    msg = Message(
        subject=current_app.config['RAGTIME_MAIL_SUBJECT_PREFIX'] +
        recipients=[to],
        sender=current_app.config['RAGTIME_MAIL_SENDER'])
    msg.body = render_template(template + '.txt', **kwargs)
    msg.html = render_template(template + '.html', **kwargs)
    mail.send(msg)
```

This function takes a recipient, a subject, the name of a template (without extension) to send as the body, and keyword arguments for the template parameters. It uses Flask-Mail's `Message` class to create an "email" object. The `Message` takes the subject, which is combined from the prefix defined in the last section and the subject passed in. The recipients argument accepts a list of email addresses. The sender is then simply the sender name and email address you also defined in the last lesson.

Then comes the message body, and you can utilize the `render_template()` to do the work for you. "Templates can be used to send emails, too?!" I heard you shout at your computer screen. That's right: Jinja2 is a *templating engine*, so you can make virtually any kind of document out of, including text files. There are two kinds of email bodies that are defined here, which is the text body and the HTML body. In case the HTML email gets rejected by an email provider's spam filter, the text email can be received by the recipient instead.

Make Your Templates

It's arts and crafts time! Well, for textual templates, anyway. This is your turn to make both:

1. a welcome email for registered users, and
2. a notification email for the app admin (you) that a user registered

You can have these emails say whatever you want. You'll create two files for each email, a `.txt` file and a `.html` file. Those will be your templates. Put them in `templates/mail/`. Call them `welcome.txt` / `html` and `new_user.txt` / `html`. They'll say something like "Welcome, {{user.username}}!" and "A new user, {{user.username}}, signed up at {{time}}". You are encouraged to use a `User` instance to get details about said user in your template. But again, it's your choice what these email messages say. :D

Making a text file template works the same way as an for HTML template. You use placeholders and create control structures the same way you would an HTML template. Once you're done, you'll get to try it out in a Flask shell session!

Note: Your templates should look very similar. The only difference really being that the text file doesn't have HTML tags. Use the same placeholders in your templates so that you don't get errors with your keyword arguments.

Send an email

Now it's time to give sending an email a shot. With your new `send_email()` function and a template message or two, you're ready.

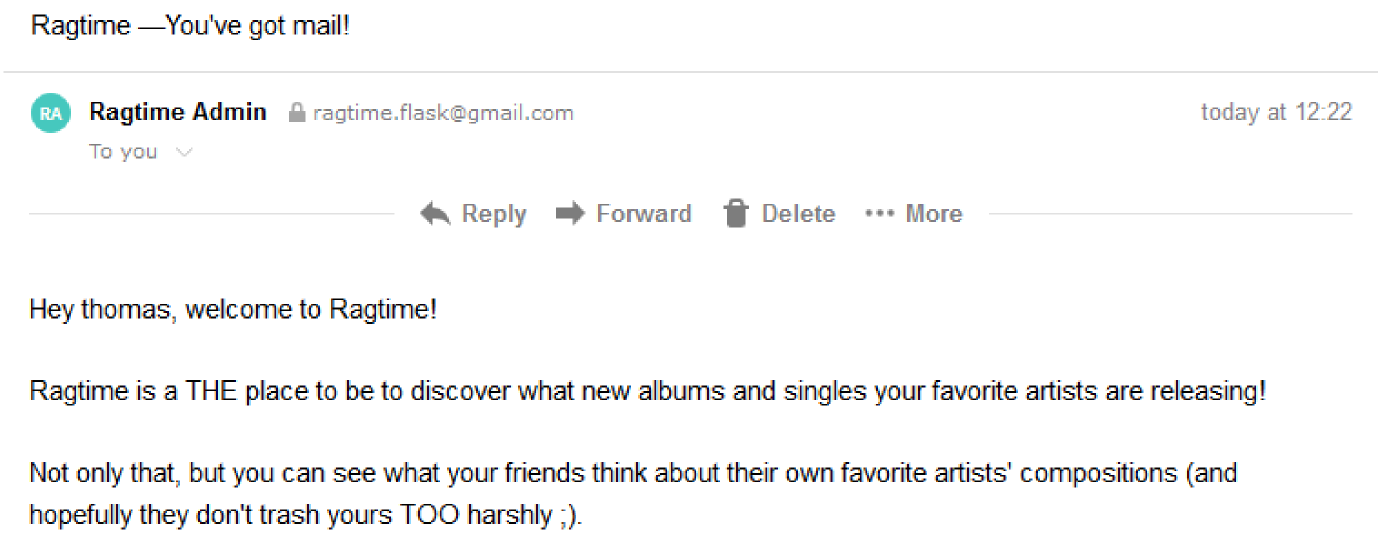
Whip open a flask shell session and try it out. To see that your email code works and actually sends an email, you can put an email address you have control of for the recipient. A fake one is used here:

```
(env) $ flask shell
>>> from app.email import send_email
>>> # If you don't already have a user in your database,
>>> # go ahead and make one on the spot
>>> u = User(username="thomas", email="thomas@example.com", password
>>> send_email(u.email, "You've got mail!", 'mail/welcome', user=u)
```

If it worked, you'll get output like this:

```
send: 'ehlo [127.0.1.1]\r\n'
reply: b'250-smtp.gmail.com at your service, [71.211.182.93]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-STARTTLS\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250 SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.googlemail.com at your service, [7
...
reply: b'221 2.0.0 closing connection t25sm2354630iog.19 - gsmtp\r\n
reply: retcode (221); Msg: b'2.0.0 closing connection t25sm2354630io
```

Notice that in the `send_email()` function, you use the `'mail/welcome'` template *without* an extension. That's because your new function automatically tacks on the extensions for sending the HTML version, or if that doesn't work, the text only version. But holy crap, you just send an email! Here's one I got as an example:


Ragtime —You've got mail!

Ragtime Admin ragtime.flask@gmail.com today at 12:22
To you

Hey thomas, welcome to Ragtime!

Ragtime is a THE place to be to discover what new albums and singles your favorite artists are releasing!

Not only that, but you can see what your friends think about their own favorite artists' compositions (and hopefully they don't trash yours TOO harshly ;).

That message actually went to spam, so be sure to check that folder. There are ways to prevent your messages from going to people's spam folders, but that's outside this course's scope.

Learn by Doing

Let's put that `send_email()` function to actual use, shall you? Because you're just starting out with a Flask web app, it would be exciting for you to get an email whenever someone registers an account with your totally rad site. But there are other reason for getting emails about the state of your app, as was alluded to above. And of course, why not welcome your user once they register?

Pull up your `register()` view function and take a look at it. Where would you put your `send_email()` functions? That's right, *functions*. You have two template messages, one for welcoming a new user called `welcome` and the other as a notification for yourself called `new_user` . The first will go to your user's email, and the second will go to your app's admin email.

Then once you put that functionality in, test it out! Try registering an account with your own email address for your user to make sure it works. If you're having trouble, remember you have the CodingNomads DIscord and perhaps your mentor to help you out.

Once you get to the point where you can send emails with the click of the registration submit button, well done! Now you're ready to learn about generating confirmation tokens for your users.

Summary: Sending Emails with Python + Flask-Mail

In this lesson you've:

- Installed **Flask-Mail**, the extension that makes handling email communication much simpler. No need to delve into the complexities of SMTP yourself.
- Understood the importance of having an SMTP server and appropriate settings.
- Familiarized yourself with basic Flask-Mail configuration by setting up key parameters like `MAIL_SERVER` , `MAIL_PORT` , `MAIL_USE_TLS` , `MAIL_USERNAME` , and `MAIL_PASSWORD` in `config.py` using environment variables.
- Implemented the initialization of the Flask-Mail extension within your Flask app.
- Created a `send_email` function that wraps Flask-Mail functionalities.
- Designed both text and HTML email templates that personalize emails for new users or admin notifications, demonstrating versatility.
- Tested email sending within the Flask shell to confirm that your setup works correctly and experienced the satisfaction of your app sending actual emails.

Previous Next → Generate Email Confirmation Tokens and Confirm User...