CODING NOMADS

13) Sending Emails & Email Verification Lesson

Send Email Asynchronously in a Flask App

5 min to complete · By Brandon Gigous

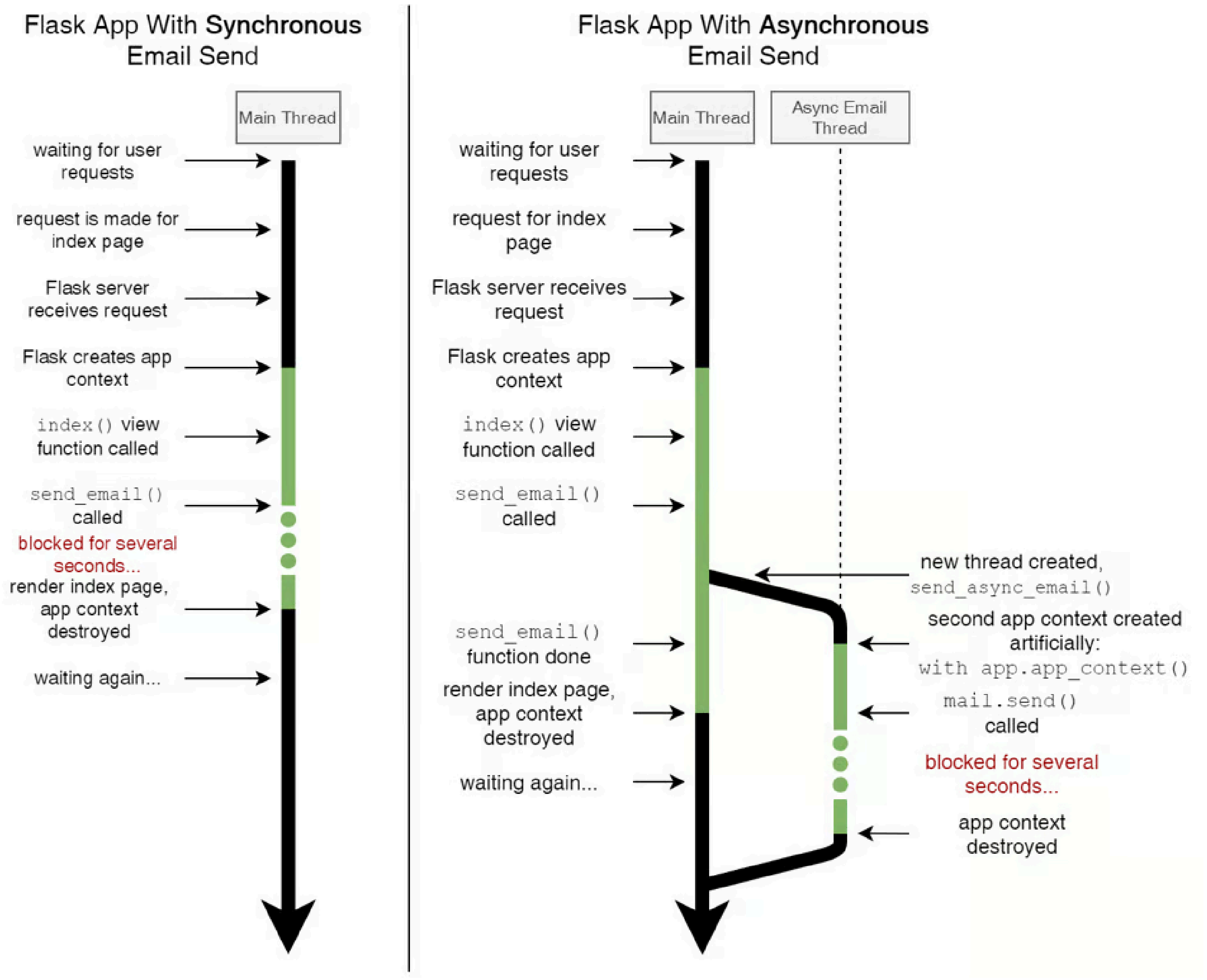
Contents

In the last lesson, you completed all your email functionality and user verification was implemented. There's one thing you can do to make this email stuff a good bit faster.

Threading

No no, that's not the same as "blocking" on Twitter. What is meant by the heading is that the `mail.send()` function in your `email.py` blocks your app until the email is sent, which takes a few seconds. Clearly you don't want that. But why does it block?

Like many Flask extensions, Flask-Mail relies on an application context to be present. You've used the `current_app` object before, and you can use it to reference things like your app's configuration settings *because* there is an application context. Otherwise, using `current_app` wouldn't be possible. The `mail.send()` itself uses `current_app` to send an email. The app, at the point where it invokes `send_email()`, and the `mail.send()` function exist on the same thread. That means Flask-Mail has to finish sending the email before `send_email()` can itself return control to its caller function. Emails take *at least* a few seconds to send because a delay is actually part of the technology to help prevent spam.


Flask App With Synchronous Email Send
Main Thread
waiting for user requests
request is made for index page
Flask server receives request
Flask creates app context
index() view function called
send_email() called
blocked for several seconds...
render index page, app context destroyed
waiting again...
Flask App With Asynchronous Email Send
Main Thread
Async Email Thread
waiting for user requests
request for index page
Flask server receives request
Flask creates app context
index() view function called
send_email() called
new thread created, send_async_email()
second app context created artificially: with app.app_context()
send_email() function done
mail.send() called
render index page, app context destroyed
blocked for several seconds...
waiting again...
app context destroyed

To allow emails to be sent in parallel with the app's normal operation—in other words, to prevent blocking of the app by `mail.send()` —you'll need to create another thread just for sending the email. You can make another thread easily with the `threading.Thread` class, but your `mail.send()` will still need an app context!

Send Async Email

To give `mail.send()` the best of both worlds, a new thread and an application context that it so desires, you can start a new thread and pass the application instance to the new function. That way, you can create a new application context artificially:

```
def send_async_email(app, msg):
    with app.app_context():
        mail.send(msg)
```

```
def send_email(to, subject, template, **kwargs):
    app = current_app._get_current_object()
    msg = Message(
        subject=app.config['RAGTIME_MAIL_SUBJECT_PREFIX'] + subject,
        recipients=[to],
        sender=config['RAGTIME_MAIL_SENDER'])
    msg.body = render_template(template + '.txt', **kwargs)
    msg.html = render_template(template + '.html', **kwargs)
    thread = Thread(target=send_async_email, args=[app, msg])
    thread.start()
```

The `current_app._get_current_object()` function does just that: gives you the *actual* application instance instead of just a proxy. Then, that application instance is passed to `send_async_email()` along with the message, where a new application context is created.

You are now hereby graduated from the School of Sending Emails with Flask! And the College of User Verification with Flask. The next section will introduce you to user roles, and what your role will be in implementing them.

Note: This is another reminder to perform a database migration!

Summary: What is Python Async Email

- To allow emails to be sent in parallel with the app's normal operation—in other words, to prevent blocking of the app by `mail.send()` —you'll need to create another thread just for sending the email. You can make another thread easily with the `threading.Thread` class, but your `mail.send()` will still need an app context!

- You can start a new thread and pass the application instance to the new function. That way, you can create a new application context artificially:

```
def send_async_email(app, msg):
    with app.app_context():
        mail.send(msg)
```

```
def send_email(to, subject, template, **kwargs):
    app = current_app._get_current_object()
    msg = Message(
        subject=app.config['RAGTIME_MAIL_SUBJECT_PREFIX'] + subje
        recipients=[to],
        sender=config['RAGTIME_MAIL_SENDER'])
    msg.body = render_template(template + '.txt', **kwargs)
    msg.html = render_template(template + '.html', **kwargs)
    thread = Thread(target=send_async_email, args=[app, msg])
    thread.start()
```

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