asphalt-mailer

Release 3.0.3

Nov 21, 2018

Contents

This Asphalt framework component provides a means for sending email from Asphalt applications.

Three mechanisms are currently supported:

- SMTP (using aiosmtplib)
- Sendmail
- Mock (just stores sent mails; useful for testing applications)

Third party libraries may provide additional backends.

CHAPTER 1

Configuration

To configure a mailer for your application, you need to choose a backend and then specify any necessary configuration values for it. The following backends are provided out of the box:

- `smtp` (**recommended**)
- `sendmail`
- `mock` (for testing only)

Other backends may be provided by other components.

Once you've selected a backend, see its specific documentation to find out what configuration values you need to provide, if any. Configuration values are expressed as constructor arguments for the backend class:

```
components:
  mailer:
    backend: smtp
    host: primary-smtp.company.com
    username: foo
    password: bar
```

This configuration uses `primary-smtp.company.com` as the server hostname. Because it has a user name and password defined, the mailer will automatically use port 587 and STARTTLS before authenticating itself with the server.

The above configuration can be done directly in Python code as follows:

```
class ApplicationComponent(ContainerComponent):
    async def start(ctx: Context):
        self.add_component(
            'mailer', backend='smtp', host='primary-smtp.company.com', username='foo',
            password='bar')
        await super().start()
```

1.1 Multiple mailers

If you need multiple mailers, you need to specify them via the `mailers` argument, which is a dictionary of resource names to their backend configuration options:

```
components:
  mailer:
    mailers:
      smtp_a:
        backend: smtp
        context_attr: mailer1
        host: primary-smtp.company.com
        username: foo
        password: bar
      smtp_b:
        backend: smtp
        context_attr: mailer2
        host: isp-smtp.provider.com
      sendmail:
        backend: sendmail
        context_attr: mailer3
```

This configures three mailer resources, named `smtp_a`, `smtp_b` and `sendmail`. Their corresponding context attributes are `mailer1`, `mailer2` and `mailer3`. If you omit the `context_attr` option for a mailer, its resource name will be used.

CHAPTER 2

Using mailers

The primary tools for sending email with asphalt-mailer are the `EmailMessage` class and the `deliver()` method. The workflow is to first construct one or more messages and then using the mailer to deliver them.

Two convenience methods are provided to this end: `create_message()` and `create_and_deliver()`. Both methods take the same arguments, but the former only creates a message (for further customization), while the latter creates and delivers a message in one shot, as the name implies.

Email messages can have plain and/or HTML content, along with attachments. The full power of the new standard library email API is at your disposal.

In addition to the examples below, some runnable examples are also provided in the `examples` directory of the source distribution. The same code is also available on Github.

2.1 Simple example

This sends a plaintext message with the body "Greetings from Example!" to `recipient@company.com`, addressed as coming from `Example Person <example@company.com>`:

```
async def handler(ctx):
    await ctx.mailer.create_and_deliver(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body='Greetings from Example!')
```

2.2 HTML content

Users may want to send styled emails using HTML. This can be done by passing the HTML content using the `html_body` argument:

```
async def handler(ctx):
    html = "<h1>Greetings</h1>Greetings from <strong>Example Person!</strong>"
    plain = "Greetings!\n\nGreetings from Example Person!"
    await ctx.mailer.create_and_deliver(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body=plain, html_body=html)
```

Note: It is highly recommended to provide a plaintext fallback message (as in the above example) for cases where the recipient cannot display HTML messages for some reason.

2.3 Attachments

To add attachments, you can use the handy `add_file_attachment()` and `add_attachment()` methods.

The following example adds the file `/path/to/file.zip` as an attachment to the message. The file will be displayed as `file.zip` with the autodetected MIME type `application/zip`:

```
async def handler(ctx):
    message = ctx.mailer.create_message(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body='See the attached file.')
    await ctx.mailer.add_file_attachment(message, '/path/to/file.zip')
    await ctx.mailer.deliver(message)
```

If you need more fine grained control, you can directly pass the attachment contents as bytes to `add_attachment()`, but then you will have to explicitly specify the file name and MIME type:

```
async def handler(ctx):
    message = ctx.mailer.create_message(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body='See the attached file.')
    ctx.mailer.add_attachment(message, b'file contents', 'attachment.txt')
    await ctx.mailer.deliver(message)
```

Warning: Most email servers today have strict limits on the size of the message, so it is recommended to keep the size of the attachments small. A maximum size of 2 MB is a good rule of thumb.

2.4 Multiple messages at once

To send multiple messages in one shot, you can use `create_message()` to create the messages and then use `deliver()` to send them. This is very useful when sending personalized emails for multiple recipients:

```
from email.headerregistry import Address


async def handler(ctx):
    messages = []
    for recipient in [Address('Some Person', 'some.person', 'company.com'),
```

(continues on next page)

(continued from previous page)

```
                    Address('Other Person', 'other.person', 'company.com')]:
        message = ctx.mailer.create_message(
            subject='Hi there, %s!' % recipient.display_name,
            sender='Example Person <example@company.com>',
            to=recipient, plain_body='How are you doing, %s?' % recipient.display_
↪name)
        messages.append(message)

    await ctx.mailer.deliver(messages)
```

2.5 Handling errors

If there is an error, a `DeliveryError` will be raised. Its `message` attribute will contain the problematic `EmailMessage` instance if the error is specific to a single message:

```
async def handler(ctx):
    try:
        await ctx.mailer.create_and_deliver(
            subject='Hi there!', sender='Example Person <example@company.com>',
            to='recipient@company.com', plain_body='Greetings from Example!')
    except DeliveryError as e:
        print('Delivery to {} failed: {}'.format(e.message['To'], e.error))
```

CHAPTER 3

Testing with mailers

When you test an application that uses asphalt-mailer, you don't want it to actually send any emails outside of your testing environment. To that end, it is recommended that you use `MockMailer` as the mailer backend in your testing configuration. This mailer simply stores the sent messages which you can then verify in your test function:

```
from asphalt.core.component import ContainerComponent
from asphalt.core.context import Context


@pytest.fixture(scope='session')
def container():
    container = ContainerComponent()
    container.add_component('mailer', backend='mock')
    return container


@pytest.fixture
def context(container, event_loop):
    context = Context()
    event_loop.run_until_complete(container.start(context))
    return context


@pytest.mark.asyncio
async def test_foo(context):
    # (do something with the application here that should cause a mail to be sent)

    # check that exactly one message was sent, to intended.recipient@example.org
    assert len(context.mailer.messages) == 1
    assert context.mailers.messages[0]['To'] == 'intended.recipient@example.org'
```

CHAPTER 4

Writing new mailer backends

If you wish to implement an alternate method of sending email, you can do so by subclassing the `Mailer` class. There are two methods implementors typically override:

- `start()` (optional)
- `deliver()`

The `start` method is a coroutine that is called by the component from its own `start()` method. You can handle any necessary resource related setup there.

The `deliver` method must be overridden and needs to:

1. handle both a single `EmailMessage` and an iterable of them
2. remove any `Bcc` header from each message to avoid revealing the hidden recipients

If you want your mailer to be available as a backend for the `MailerComponent`, you need to add the corresponding entry point for it. Suppose your mailer class is named `AwesomeMailer`, lives in the package `foo.bar.awesome` and you want to give it the alias `awesome`, add this line to your project's `setup.py` under the `entrypoints` argument in the `asphalt.mailer.mailers` namespace:

```
setup(
    # (...other arguments...)
    entry_points={
        'asphalt.mailer.mailers': [
            'awesome = foo.bar.awesome:AwesomeMailer'
        ]
    }
)
```

CHAPTER 5

Version history

This library adheres to Semantic Versioning.

3.0.3 (2018-11-21)

- Retry of the previous release, with all reported code style errors fixed

3.0.2 (2018-09-27, never uploaded to PyPI)

- Fixed deprecation warning about importing ABCs directly from `collections`

3.0.1 (2016-06-04)

- Added compatibility with Asphalt 4.0

3.0.0 (2017-04-17)

- **BACKWARD INCOMPATIBLE** Migrated to Asphalt 3.0
- **BACKWARD INCOMPATIBLE** Replaced home grown SMTP implementation with aiosmtplib
- **BACKWARD INCOMPATIBLE** Implicit TLS support in SMTPMailer was replaced with STARTTLS support
- **BACKWARD INCOMPATIBLE** The `ssl` option in SMTPMailer was replaced with the `tls` and `tls_context` options
- **BACKWARD INCOMPATIBLE** Default port selection in SMTPMailer was changed; see the class docstring for details
- **BACKWARD INCOMPATIBLE** Renamed the `asphalt.mailer.util` module to `asphalt.mailer.utils`

2.0.1 (2017-01-09)

- Fixed occasional missing dots in the messages (due to not quoting leading dots)

2.0.0 (2016-05-09)

- **BACKWARD INCOMPATIBLE** Migrated to Asphalt 2.0
- **BACKWARD INCOMPATIBLE** Renamed `Mailer.defaults` to `Mailer.message_defaults`
- Allowed combining `mailers` with default parameters

- Fixed default message parameters not being applied in `Mailer.create_message()`

1.1.0 (2016-01-02)

- Added typeguard checks to fail early if arguments of wrong types are passed to functions

1.0.0 (2015-10-31)

- Initial release
- API reference