Logging

Flask uses standard Python **`logging`**. Messages about your Flask application are logged with **`app.logger`**, which takes the same name as **`app.name`**. This logger can also be used to log your own messages.

```
@app.route('/login', methods=['POST'])
def login():
    user = get_user(request.form['username'])

    if user.check_password(request.form['password']):
        login_user(user)
        app.logger.info('%s logged in successfully', user.username)
        return redirect(url_for('index'))
    else:
        app.logger.info('%s failed to log in', user.username)
        abort(401)
```

Basic Configuration

When you want to configure logging for your project, you should do it as soon as possible when the program starts. If **`app.logger`** is accessed before logging is configured, it will add a default handler. If possible, configure logging before creating the application object.

This example uses **`dictConfig()`** to create a logging configuration similar to Flask's default, except for all logs:

```
from logging.config import dictConfig

dictConfig({
    'version': 1,
    'formatters': {'default': {
        'format': '[%(asctime)s] %(levelname)s in %(module)s: %(message
    }},
    'handlers': {'wsgi': {
        'class': 'logging.StreamHandler',
        'stream': 'ext://flask.logging.wsgi_errors_stream',
        'formatter': 'default'
    }},
    'root': {
        'level': 'INFO',
        'handlers': ['wsgi']
    }
})
```

```
app = Flask(__name__)
```

Default Configuration

If you do not configure logging yourself, Flask will add a **`StreamHandler`** to **`app.logger`** automatically. During requests, it will write to the stream specified by the WSGI server in `environ['wsgi.errors']` (which is usually **`sys.stderr`**). Outside a request, it will log to **`sys.stderr`**.

Removing the Default Handler

If you configured logging after accessing **`app.logger`**, and need to remove the default handler, you can import and remove it:

```
from flask.logging import default_handler

app.logger.removeHandler(default_handler)
```

Email Errors to Admins

When running the application on a remote server for production, you probably won't be looking at the log messages very often. The WSGI server will probably send log messages to a file, and you'll only check that file if a user tells you something went wrong.

To be proactive about discovering and fixing bugs, you can configure a **`logging.handlers.SMTPHandler`** to send an email when errors and higher are logged.

```
import logging
from logging.handlers import SMTPHandler

mail_handler = SMTPHandler(
    mailhost='127.0.0.1',
    fromaddr='server-error@example.com',
    toaddrs=['admin@example.com'],
    subject='Application Error'
)
mail_handler.setLevel(logging.ERROR)
mail_handler.setFormatter(logging.Formatter(
    '[%(asctime)s] %(levelname)s in %(module)s: %(message)s'
))
```

```
if not app.debug:
    app.logger.addHandler(mail_handler)
```

This requires that you have an SMTP server set up on the same server. See the Python docs for more information about configuring the handler.

Injecting Request Information

Seeing more information about the request, such as the IP address, may help debugging some errors. You can subclass **`logging.Formatter`** to inject your own fields that can be used in messages. You can change the formatter for Flask's default handler, the mail handler defined above, or any other handler.

```
from flask import has_request_context, request
from flask.logging import default_handler

class RequestFormatter(logging.Formatter):
    def format(self, record):
        if has_request_context():
            record.url = request.url
            record.remote_addr = request.remote_addr
        else:
            record.url = None
            record.remote_addr = None

        return super().format(record)

formatter = RequestFormatter(
    '[%(asctime)s] %(remote_addr)s requested %(url)s\n'
    '%(levelname)s in %(module)s: %(message)s'
)
default_handler.setFormatter(formatter)
mail_handler.setFormatter(formatter)
```

Other Libraries

Other libraries may use logging extensively, and you want to see relevant messages from those logs too. The simplest way to do this is to add handlers to the root logger instead of only the app logger.

```
from flask.logging import default_handler

root = logging.getLogger()
```

```
root.addHandler(default_handler)
root.addHandler(mail_handler)
```

Depending on your project, it may be more useful to configure each logger you care about separately, instead of configuring only the root logger.

```
for logger in (
    app.logger,
    logging.getLogger('sqlalchemy'),
    logging.getLogger('other_package'),
):
    logger.addHandler(default_handler)
    logger.addHandler(mail_handler)
```

Werkzeug

Werkzeug logs basic request/response information to the `'werkzeug'` logger. If the root logger has no handlers configured, Werkzeug adds a **`StreamHandler`** to its logger.

Flask Extensions

Depending on the situation, an extension may choose to log to **`app.logger`** or its own named logger. Consult each extension's documentation for details.