django-request-id Documentation

Release 0.1.0

Filip Wasilewski

September 29, 2014

Contents

CHAPTER 1

django-request-id

Augments each request with unique `request_id` attribute and provides request id logging helpers.

Developed and used at en.ig.ma software shop.

1.1 Quickstart

1. Include `django-request-id` in your `requirements.txt` file.
2. Add `request_id` to `INSTALLED_APPS` (necessary only if you are going to use the `{% request_id %}` template tag).
3. Add `request_id.middleware.RequestIdMiddleware` to the top of `MIDDLEWARE_CLASSES`.
4. The app integrates with the standard Python/Django logging by defining a filter that puts a `request_id` variable in scope of every log message.

   First add a filter definition to the Django `LOGGING` settings:

   ```
   "filters": {
       "request_id": {
           "()": "request_id.logging.RequestIdFilter"
       }
   }
   ```

   Then enable the filter for related handlers:

   ```
   "handlers": {
       "console": {
           ...
           "filters": ["request_id"],
       }
    }
   ```

   And finally modify formatter output format to include the `%(request_id)` placeholder:

   ```
   "formatters": {
       "console": {
           "format": "%(asctime)s - %(levelname)-5s [%(name)s] request_id=%(request_id)s %(message)
       }
   }
   ```

   A full Django logging config example may look like this:

```
LOGGING= {
    "version": 1,
    "disable_existing_loggers": False,
    "filters": {
        "request_id": {
            "()": "request_id.logging.RequestIdFilter"
        }
    },
    "formatters": {
        "console": {
            "format": "%(asctime)s - %(levelname)-5s [%(name)s] request_id=%(request_id)s %(mess
            "datefmt": "%H:%M:%S"
        }
    },
    "handlers": {
        "console": {
            "level": "DEBUG",
            "filters": ["request_id"],
            "class": "logging.StreamHandler",
            "formatter": "console"
        }
    },
    "loggers": {
        "": {
            "level": "DEBUG",
            "handlers": ["console"]
        }
    }
}
```

5. Make sure that your web server adds a `X-Request-ID` header to each request (and logs it in the server log for further matching of the server and app log entries).

   Heroku handles this automatically. On Nginx you may require a separate module (see nginx_requestid or nginx-x-rid-header). On Apache you need to `a2enmod` the unique_id module and set `REQUEST_ID_HEADER = "UNIQUE_ID"` in the Django project settings.

   If you can't generate the X-Request-Id header at the web server level then simply set `REQUEST_ID_HEADER = None` in your project settings and the app will generate a unique id value automatically instead of retrieving it from the wsgi environment.

   For more info on server configs see server-config.

1.2 Dependencies

`django-request-id` depends on `django-appconf>=0.6`.

1.3 Documentation

The full documentation is at http://django-request-id.rtfd.org.

There's also an instant demo example that can be run from the cloned repository:

```
python demo.py
```

1.4 License

`django-request-id` is released under the MIT license.

1.5 Other Resources

- GitHub repository - https://github.com/nigma/django-request-id
- PyPi Package site - http://pypi.python.org/pypi/django-request-id

1.6 Commercial Support

This app and many other help us build better software and focus on delivering quality projects faster. We would love to help you with your next project so get in touch by dropping an email at en@ig.ma.

1.7 Content

1.7.1 Web server configs

Heroku

If the X-Request-ID header is not passed automatically you may need to enable it using the labs command:

```
heroku labs:enable http-request-id
```

See http-request-id for more info.

The `request_id` will also appear in the Heroku Dyno logs so it is easy to match application logs with Heroku request logs.

Nginx

There's no built-in option in Nginx to generate a unique request id, but there are several modules that provide this functionality:

- nginx_requestid
- nginx-x-rid-header

Unfortunately this require Nginx binary recompilation.

Alternatively, if your Nginx has Lua scripting enabled, you can generate a random id and add it to server logs using the following snippets:

```
http {
    ...

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for" '
                      'request_id=$request_id';
```

```
    ...
}

server {
    listen       80;
    server_name  localhost;

    access_log  logs/host.access.log  main;

    location / {
        set_by_lua $request_id '
            local function random_id()
                local charset = "0123456789abcdefghijklmnopqrstuvwxyz"
                local template = "xxxx-xxxxxxxx-xxxxxxxx"
                local range = charset:len()
                return string.gsub(template, "x", function (c)
                    return string.char(charset:byte(math.random(1, range)))
                end)
            end
            local request_id = random_id()
            ngx.req.set_header("X-Request-Id", request_id)
            return request_id
        ';

        ...
    }
```

Apache

On Apache you need to `a2enmod` the unique_id module and set `REQUEST_ID_HEADER = "UNIQUE_ID"` in the Django project settings.

Standalone

If you can't generate the X-Request-Id header at the web server level then simply set `REQUEST_ID_HEADER = None` in your project settings and the app will generate a unique id value automatically instead of retrieving it from the wsgi environment.

You can also use the `request_id.wsgi.AddRequestIdHeaderMiddleware` WSGI middleware for that purpose.

1.7.2 Credits

Development Lead

- Filip Wasilewski <en@ig.ma>

Contributors

None yet. Why not be the first?

1.7.3 History

0.1.0 (2014-01-30)

- First release

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*