django-guid

Jun 16, 2023

Contents

1 Installation **3**

2 Configuration **5**
- 2.1 1. Installed Apps . . . 5
- 2.2 2. Middleware . . . 5
- 2.3 3. Logging Configuration . . . 5
- 2.4 4. Django GUID Logger (Optional) . . . 6

3 Settings **7**
- 3.1 GUID_HEADER_NAME . . . 7
- 3.2 VALIDATE_GUID . . . 7
- 3.3 RETURN_HEADER . . . 8
- 3.4 EXPOSE_HEADER . . . 8
- 3.5 INTEGRATIONS . . . 8
- 3.6 IGNORE_URLS . . . 8
- 3.7 UUID_LENGTH . . . 8
- 3.8 UUID_LENGTH . . . 8

4 API **11**
- 4.1 Getting started . . . 11
- 4.2 get_guid() . . . 11
- 4.3 set_guid() . . . 11
- 4.4 clear_guid() . . . 12
- 4.5 Example usage . . . 12

5 Integrations **13**
- 5.1 Available integrations . . . 13
  - 5.1.1 Sentry . . . 13
  - 5.1.2 Celery . . . 15
- 5.2 Writing your own integration . . . 17
  - 5.2.1 Method descriptions . . . 18

6 Extended example **21**
- 6.1 Experiment . . . 21

7 Troubleshooting **23**
- 7.1 Turn on Django debug logging . . . 23

7.2 Run Django with warnings enabled . . . 23
7.3 Use the demo project as a reference . . . 23
7.4 Read the official logging docs . . . 23
7.5 Ask for help . . . 24

8 Upgrading Django-GUID 2.x.x to 3.x.x **25**
8.1 1. Change Middleware . . . 25
8.2 2. Change API functions (if you used them) . . . 25

9 Publish django-guid **27**
9.1 Publishing to test-PyPi . . . 27
9.2 Publishing to PyPi . . . 27
9.3 Read the docs . . . 27

10 Changelog **29**
10.1 3.2.1 - 13.12.2021 . . . 29
10.2 3.2.0 - 04.12.2020 . . . 29
10.3 3.1.0 - 18.11.2020 . . . 29
10.4 3.0.1 - 12.11.2020 . . . 29
10.5 3.0.0 - 28.10.2020 - Full Django3.1+(ASGI/async) support! . . . 30
10.6 2.2.0 - 04.11.2020 . . . 30
10.7 2.1.0 - 03.11.2020 . . . 30
10.8 2.0.0 - 02.03.2020 . . . 30
10.9 1.1.1 - 12.02.2020 . . . 31
10.10 1.1.0 - 10.02.2020 . . . 31
10.11 1.0.1 - 08.02.2020 . . . 31
10.12 1.0.0 - 14.01.2020 . . . 31
10.13 0.3.1 - 13.01.2020 . . . 32
10.14 0.3.0 - 10.01.2020 . . . 32
10.15 0.2.3 - 09.01.2020 . . . 32
10.16 0.2.2 - 21.12.2019 . . . 32
10.17 0.2.1 - 21.12.2019 . . . 32
10.18 0.2.0 - 21.12.2019 . . . 32
10.19 20.10.2019 . . . 33

Django GUID attaches a unique correlation ID/request ID to all your log outputs for every request. In other words, all logs connected to a request now has a unique ID attached to it, making debugging simple.

Which version of Django GUID you should use depends on your Django version and whether you run `ASGI` or `WSGI` servers. To determine which Django-GUID version you should use, please see the table below.

Django version	Django-GUID version
3.1.1 or above	3.x.x - ASGI and WSGI
3.0.0 - 3.1.0	2.x.x - Only WSGI
2.2.x	2.x.x - Only WSGI

Django GUID >= 3.0.0 uses `ContextVar` to store and access the GUID. Previous versions stored the GUID to an object, making it accessible by using the ID of the current thread.

Resources:

- Free software: BSD License
- Documentation: https://django-guid.readthedocs.io
- Homepage: https://github.com/snok/django-guid

Examples

Log output with a GUID:

```
INFO ... [773fa6885e03493498077a273d1b7f2d] project.views This is a DRF view log, and␣
↪should have a GUID.
WARNING ... [773fa6885e03493498077a273d1b7f2d] project.services.file Some warning in␣
↪a function
INFO ... [0d1c3919e46e4cd2b2f4ac9a187a8ea1] project.views This is a DRF view log, and␣
↪should have a GUID.
INFO ... [99d44111e9174c5a9494275aa7f28858] project.views This is a DRF view log, and␣
↪should have a GUID.
WARNING ... [0d1c3919e46e4cd2b2f4ac9a187a8ea1] project.services.file Some warning in␣
↪a function
WARNING ... [99d44111e9174c5a9494275aa7f28858] project.services.file Some warning in␣
↪a function
```

Log output without a GUID:

```
INFO ... project.views This is a DRF view log, and should have a GUID.
WARNING ... project.services.file Some warning in a function
INFO ... project.views This is a DRF view log, and should have a GUID.
INFO ... project.views This is a DRF view log, and should have a GUID.
WARNING ... project.services.file Some warning in a function
WARNING ... project.services.file Some warning in a function
```

CHAPTER 1

Installation

Install using pip:

```
pip install django-guid
```

Install using poetry:

```
poetry add django-guid
```

CHAPTER 2

Configuration

Once django guid has been installed, add the following to your projects' `settings.py`:

2.1 1. Installed Apps

Add `django_guid` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    ...
    'django_guid',
]
```

2.2 2. Middleware

Add the `django_guid.middleware.guid_middleware` to your `MIDDLEWARE`:

```
MIDDLEWARE = [
    'django_guid.middleware.guid_middleware',
    ...
 ]
```

It is recommended that you add the middleware at the top, so that the remaining middleware loggers include the requests GUID.

2.3 3. Logging Configuration

Add `django_guid.log_filters.CorrelationId` as a filter in your `LOGGING` configuration:

```
LOGGING = {
    ...
    'filters': {
        'correlation_id': {
            '()': 'django_guid.log_filters.CorrelationId'
        }
    }
}
```

Put that filter in your handler:

```
LOGGING = {
    ...
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'medium',
            'filters': ['correlation_id'],
        }
    }
}
```

And make sure to add the new `correlation_id` filter to one or all of your formatters:

```
LOGGING = {
    ...
    'formatters': {
        'medium': {
            'format': '%(levelname)s %(asctime)s [%(correlation_id)s] %(name)s
↪%(message)s'
        }
    }
}
```

If these settings were confusing, please have a look in the demo projects' settings.py file for a complete example.

2.4 4. Django GUID Logger (Optional)

If you wish to see the Django GUID middleware outputs, you may configure a logger for the module. Simply add django_guid to your loggers in the project, like in the example below:

```
LOGGING = {
    ...
    'loggers': {
        'django_guid': {
            'handlers': ['console', 'logstash'],
            'level': 'WARNING',
            'propagate': False,
        }
    }
}
```

This is especially useful when implementing the package, if you plan to pass existing GUIDs to the middleware, as misconfigured GUIDs will not raise exceptions, but will generate warning logs.

CHAPTER 3

Settings

Package settings are added in your `settings.py`:

Default settings are shown below:

```
DJANGO_GUID = {
    'GUID_HEADER_NAME': 'Correlation-ID',
    'VALIDATE_GUID': True,
    'RETURN_HEADER': True,
    'EXPOSE_HEADER': True,
    'INTEGRATIONS': [],
    'UUID_LENGTH': 32,
    'UUID_FORMAT': 'hex',
}
```

3.1 GUID_HEADER_NAME

- **Default**: `Correlation-ID`
- **Type**: `string`

The name of the GUID to look for in a header in an incoming request. Remember that it's case insensitive.

3.2 VALIDATE_GUID

- **Default**: `True`
- **Type**: `boolean`

Whether the `GUID_HEADER_NAME` should be validated or not. If the GUID sent to through the header is not a valid GUID (`uuid.uuid4`).

3.3 RETURN_HEADER

- **Default**: `True`
- **Type**: `boolean`

Whether to return the GUID (Correlation-ID) as a header in the response or not. It will have the same name as the `GUID_HEADER_NAME` setting.

3.4 EXPOSE_HEADER

- **Default**: `True`
- **Type**: `boolean`

Whether to return `Access-Control-Expose-Headers` for the GUID header if `RETURN_HEADER` is `True`, has no effect if `RETURN_HEADER` is `False`. This is allows the JavaScript Fetch API to access the header when CORS is enabled.

3.5 INTEGRATIONS

- **Default**: `[]`
- **Type**: `list`

Whether to enable any custom or available integrations with `django_guid`. As an example, using `SentryIntegration()` as an integration would set Sentry's `transaction_id` to match the GUID used by the middleware.

3.6 IGNORE_URLS

- **Default**: `[]`
- **Type**: `list`

URL endpoints where the middleware will be disabled. You can put your health check endpoints here.

3.7 UUID_LENGTH

- **Default**: `32`
- **Type**: `int`

If a full UUID hex is too long for you, this settings lets you specify the length you wish to use. The chance of collision in a UUID is so low, that most systems will get away with a lot fewer than 32 characters.

3.8 UUID_LENGTH

- **Default**: `hex`

- **Type**: `string`

If a UUID hex is not suitable for you, this settings lets you specify the format you wish to use. The options are: * `hex`: The default, a 32 character hexadecimal string. e.g. ee586b0fba3c44849d20e1548210c050 * `str`: A 36 character string. e.g. ee586b0f-ba3c-4484-9d20-e1548210c050

CHAPTER 4

API

4.1 Getting started

You can either use the `contextvar` directly by importing it with `django_guid.middleware import guid`, or use the API which also logs changes. If you want to use the contextvar, please see the official Python docs.

To use the API import the functions you'd like to use:

```
from django_guid import get_guid, set_guid, clear_guid
```

4.2 get_guid()

- **Returns**: `str` or `None`, if set by Django-GUID.

Fetches the GUID.

```
guid = get_guid()
```

4.3 set_guid()

- **Parameters**: `guid`: `str`

Sets the GUID to the given input.

```
set_guid('My GUID')
```

4.4 clear_guid()

Clears the guid (sets it to `None`)

```
clear_guid()
```

4.5 Example usage

```
import requests
from django.conf import settings

from django_guid import get_guid

requests.get(
    url='http://localhost/api',
    headers={
        'Accept': 'application/json',
        settings.DJANGO_GUID['GUID_HEADER_NAME']: get_guid(),
    }
)
```

CHAPTER 5

Integrations

Integrations are optional add-ins used to extend the functionality of the Django GUID middleware.

To enable an integration, simply add an integration instance to the `INTEGRATIONS` field in `settings.py`, and the relevant integration logic will be executed in the middleware:

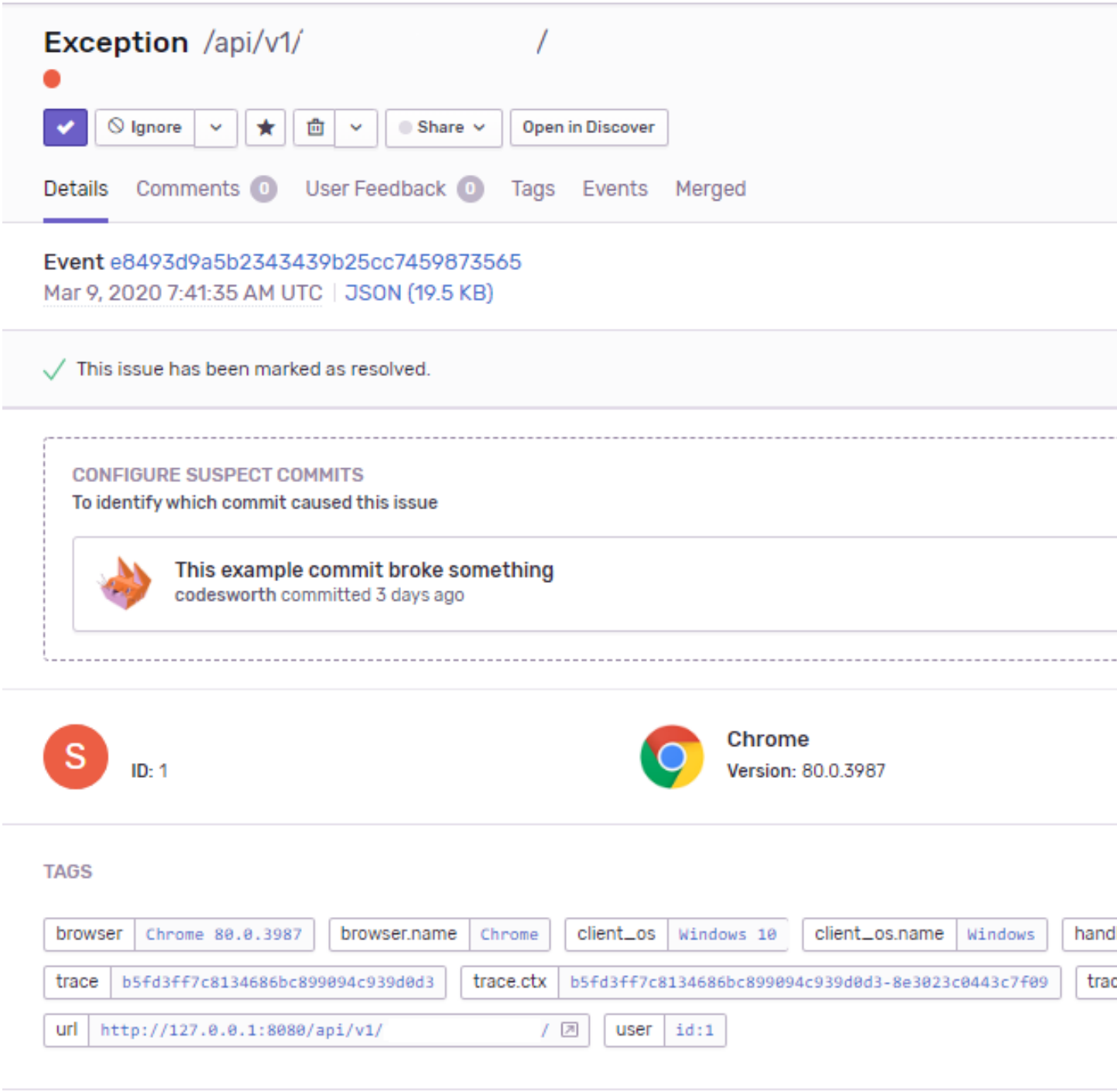
```
from django_guid.integrations import SentryIntegration

DJANGO_GUID = {
    ...
    'INTEGRATIONS': [SentryIntegration()],
}
```

Integrations are a new addition to Django GUID, and we plan to expand selection in the future. If you are looking for specific functionality that is not yet available, consider creating an issue, making a pull request, or writing your own private integration. Custom integrations classes are simple to write and can be implemented just like package integrations.

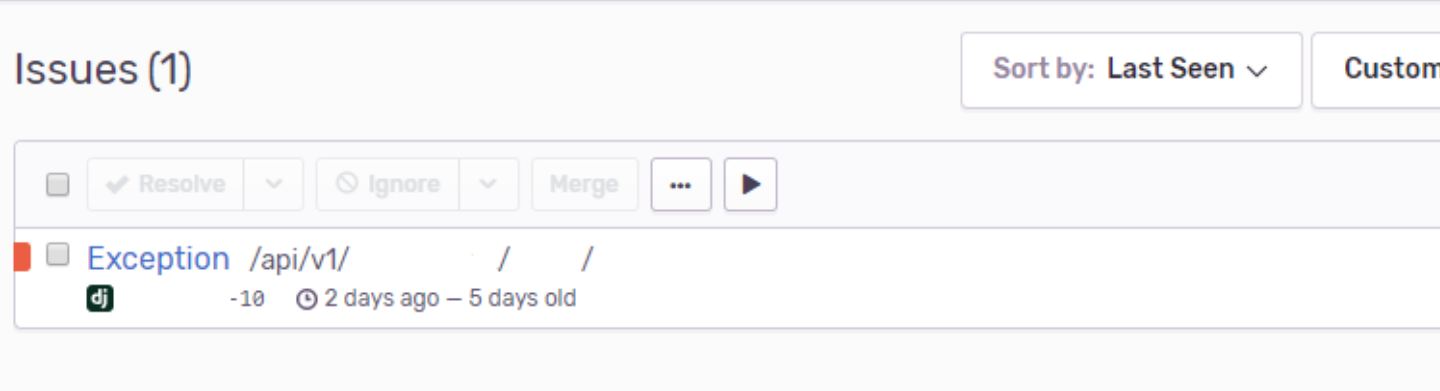
5.1 Available integrations

5.1.1 Sentry

Integrating with Sentry, lets you tag Sentry-issues with a `transaction_id`. This lets you easily connect an event in Sentry to your logs.



Rather than changing how Sentry works, this is just an additional piece of metadata that you can use to link sources of information about an exception. If you know the GUID of an exception, you can find the relevant Sentry issue by searching for the tag:



To add the integration, simply import `SentryIntegration` from the integrations folder and add it to your settings:

```
from django_guid.integrations import SentryIntegration

DJANGO_GUID = {
    ...
    'INTEGRATIONS': [SentryIntegration()],
}
```

5.1.2 Celery

The Celery integration enables tracing for Celery workers. There's three possible scenarios:

1. A task is published from a request within Django
2. A task is published from another task
3. A task is published from Celery Beat

For scenario 1 and 2 the existing correlation IDs is transferred, and for scenario 3 a unique ID is generated.

To enable this behavior, simply add it to your list of integrations:

```
from django_guid.integrations import CeleryIntegration

DJANGO_GUID = {
    ...
    'INTEGRATIONS': [
        CeleryIntegration(
            use_django_logging=True,
            log_parent=True,
        )
    ],
}
```

Integration settings

These are the settings you can pass when instantiating the `CeleryIntegration`:

- **use_django_logging**: Tells celery to use the Django logging configuration (formatter).
- **log_parent**: Enables the `CeleryTracing` log filter described below.
- **uuid_length**: Lets you optionally trim the length of the integration generated UUIDs.

- **sentry_integration**: If you use Sentry, enabling this setting will make sure `transaction_id` is set (like in the SentryIntegration) for Celery workers.

Celery integration log filter

Out of the box, the CeleryIntegration will make sure a correlation ID is present for any Celery task; but how do you make sense of duplicate logs in subprocesses? Given these example tasks, what happens if we a worker picks up `debug_task` as scheduled by Celery beat?

```
@app.task()
def debug_task() -> None:
    logger.info('Debug task 1')
    second_debug_task.delay()
    second_debug_task.delay()


@app.task()
def second_debug_task() -> None:
    logger.info('Debug task 2')
    third_debug_task.delay()
    fourth_debug_task.delay()


@app.task()
def third_debug_task() -> None:
    logger.info('Debug task 3')
    fourth_debug_task.delay()
    fourth_debug_task.delay()


@app.task()
def fourth_debug_task() -> None:
    logger.info('Debug task 4')
```

It will be close to impossible to make sense of the logs generated, simply because the correlation ID tells you nothing about how subprocesses are linked. For this, the integration provides an additional log filter, `CeleryTracing` which logs the ID of the current process and the ID of the parent process. Using the log filter, the log output of the example tasks becomes:

```
   correlation-id               current-id
          |        parent-id        |
          |            |            |
INFO [3b162382e1] [   None   ] [93ddf3639c] demoproj.celery - Debug task 1
INFO [3b162382e1] [93ddf3639c] [24046ab022] demoproj.celery - Debug task 2
INFO [3b162382e1] [93ddf3639c] [cb5595a417] demoproj.celery - Debug task 2
INFO [3b162382e1] [24046ab022] [08f5428a66] demoproj.celery - Debug task 3
INFO [3b162382e1] [24046ab022] [32f40041c6] demoproj.celery - Debug task 4
INFO [3b162382e1] [cb5595a417] [1c75a4ed2c] demoproj.celery - Debug task 3
INFO [3b162382e1] [08f5428a66] [578ad2d141] demoproj.celery - Debug task 4
INFO [3b162382e1] [cb5595a417] [21b2ef77ae] demoproj.celery - Debug task 4
INFO [3b162382e1] [08f5428a66] [8cad7fc4d7] demoproj.celery - Debug task 4
INFO [3b162382e1] [1c75a4ed2c] [72a43319f0] demoproj.celery - Debug task 4
INFO [3b162382e1] [1c75a4ed2c] [ec3cf4113e] demoproj.celery - Debug task 4
```

At the very least, this should provide a mechanism for linking parent/children processes in a meaningful way.

To set up the filter, add `django_guid.integrations.celery.log_filters.CeleryTracing` as a fil-

ter in your `LOGGING` configuration:

```
LOGGING = {
    ...
    'filters': {
        'celery_tracing': {
            '()': 'django_guid.integrations.celery.log_filters.CeleryTracing'
        }
    }
}
```

Put that filter in your handler:

```
LOGGING = {
    ...
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'medium',
            'filters': ['correlation_id', 'celery_tracing'],
        }
    }
}
```

And then you can **optionally** add `celery_parent_id` and/or `celery_current_id` to you formatter:

```
LOGGING = {
    ...
    'formatters': {
        'medium': {
            'format': '%(levelname)s [%(correlation_id)s] [%(celery_parent_id)s-
↪%(celery_current_id)s] %(name)s - %(message)s'
        }
    }
}
```

However, if you use a log management tool which lets you interact with `log.extra` value, leaving the filters out of the formatter might be preferable.

If these settings were confusing, please have a look in the demo projects' settings.py file for a complete example.

5.2 Writing your own integration

Creating your own custom integration requires you to inherit the `Integration` base class (which is found here).

The class is quite simple and only contains four methods and a class attribute:

```
class Integration(object):
    """
    Integration base class.
    """

    identifier = None  # The name of your integration

    def __init__(self) -> None:
        if self.identifier is None:
```

(continues on next page)

(continued from previous page)

```
            raise ImproperlyConfigured('`identifier` cannot be None')

    def setup(self) -> None:
        """
        Holds validation and setup logic to be run when Django starts.
        """
        pass

    def run(self, guid: str, **kwargs) -> None:
        """
        Code here is executed in the middleware, before the view is called.
        """
        raise ImproperlyConfigured(f'The integration `{self.identifier}` is missing a
↪`run` method')

    def cleanup(self, **kwargs) -> None:
        """
        Code here is executed in the middleware, after the view is called.
        """
        pass
```

To extend this into a fully functioning integration, all you need to do is

1. Create a new class that inherits the base class
2. Set the identifier to a string, naming your integration
3. Add the logic you wish to be executed to the `run` method
4. Add logic to each of the remaining methods as required

A fully functioning integration can be as simple as this:

```
from django_guid.integrations import Integration

class CustomIntegration(Integration):

    identifier = 'CustomIntegration'  # Should be a string

    def run(self, guid, **kwargs):
        print('This is a functioning Django GUID integration')
```

There are four built in methods which are always called. You can chose to override these in your custom integration.

5.2.1 Method descriptions

Setup

The `setup` method is run when Django starts, and is a good place to keep your integration-specific validation logic, like, e.g., making sure all dependencies are installed:

```
from third_party_sdk import start_service

class CustomIntegration(Integration):

    identifier = 'CustomIntegration'
```

(continues on next page)

(continued from previous page)

```
    def setup(self):
        try:
            import third_party_sdk
        except ModuleNotFoundError:
            raise ImproperlyConfigured(
                'Package third_party_sdk must be installed'
            )
```

Run

The `run` method is required, and is designed to hold code that should be executed each time the middleware is run (for each request made to the server), before the view is called.

This function **must** accept both `guid` and `**kwargs`. Additional arguments are likely be added in the future, and so the function must be able to handle those new arguments.

```
from third_party_sdk import send_guid_to_system

class CustomIntegration(Integration):

    identifier = 'CustomIntegration'

    def setup(self):
        ...

    def run(self, guid, **kwargs):
        send_guid_to_system(guid=guid)
```

Cleanup

The `cleanup` method is the final method called in the middleware, each time the middleware, each time the middleware is run, after a view has been called.

This function **must** accept `**kwargs`. Additional arguments are likely be added in the future, and so the function must be able to handle those new arguments.

```
from third_party_sdk import clean_up_guid

class CustomIntegration(Integration):

    identifier = 'CustomIntegration'

    def setup(self):
        ...

    def run(self, guid, **kwargs):
        ...

    def cleanup(self, **kwargs):
        clean_up_guid()
```

CHAPTER 6

Extended example

Using tools like `ab` (Apache Benchmark) we can benchmark our application with concurrent requests, simulating heavy load. This is an easy way to display the strength of `django-guid`.

6.1 Experiment

First, we run our application like we would in a production environment:

```
gunicorn demoproj.wsgi:application --bind 127.0.0.1:8080 -k gthread -w 4
```

Then, we do 3 concurrent requests to one of our endpoints:

```
ab -c 3 -n 3 http://127.0.0.1:8080/api
```

This results in these logs:

```
django-guid git:(master)  gunicorn demoproj.wsgi:application --bind 127.0.0.1:8080 -k␣
↪gthread -w 4

[2020-01-14 16:36:15 +0100] [8624] [INFO] Starting gunicorn 20.0.4
[2020-01-14 16:36:15 +0100] [8624] [INFO] Listening at: http://127.0.0.1:8080 (8624)
[2020-01-14 16:36:15 +0100] [8624] [INFO] Using worker: gthread
[2020-01-14 16:36:15 +0100] [8627] [INFO] Booting worker with pid: 8627
[2020-01-14 16:36:15 +0100] [8629] [INFO] Booting worker with pid: 8629
[2020-01-14 16:36:15 +0100] [8630] [INFO] Booting worker with pid: 8630
[2020-01-14 16:36:15 +0100] [8631] [INFO] Booting worker with pid: 8631

# First request
INFO 2020-01-14 15:40:48,953 [None] django_guid.middleware No Correlation-ID found in␣
↪the header. Added Correlation-ID: 773fa6885e03493498077a273d1b7f2d
INFO 2020-01-14 15:40:48,954 [773fa6885e03493498077a273d1b7f2d] demoproj.views This␣
↪is a DRF view log, and should have a GUID.
WARNING 2020-01-14 15:40:48,954 [773fa6885e03493498077a273d1b7f2d] demoproj.services.
↪useless_file Some warning in a function
```

(continues on next page)

(continued from previous page)

```
DEBUG 2020-01-14 15:40:48,954 [773fa6885e03493498077a273d1b7f2d] django_guid.
↪middleware Deleting 773fa6885e03493498077a273d1b7f2d from _guid

# Second and third request arrives at the same time
INFO 2020-01-14 15:40:48,955 [None] django_guid.middleware No Correlation-ID found in␣
↪the header. Added Correlation-ID: 0d1c3919e46e4cd2b2f4ac9a187a8ea1
INFO 2020-01-14 15:40:48,955 [None] django_guid.middleware No Correlation-ID found in␣
↪the header. Added Correlation-ID: 99d44111e9174c5a9494275aa7f28858
INFO 2020-01-14 15:40:48,955 [0d1c3919e46e4cd2b2f4ac9a187a8ea1] demoproj.views This␣
↪is a DRF view log, and should have a GUID.
INFO 2020-01-14 15:40:48,955 [99d44111e9174c5a9494275aa7f28858] demoproj.views This␣
↪is a DRF view log, and should have a GUID.
WARNING 2020-01-14 15:40:48,955 [0d1c3919e46e4cd2b2f4ac9a187a8ea1] demoproj.services.
↪useless_file Some warning in a function
WARNING 2020-01-14 15:40:48,955 [99d44111e9174c5a9494275aa7f28858] demoproj.services.
↪useless_file Some warning in a function
DEBUG 2020-01-14 15:40:48,955 [0d1c3919e46e4cd2b2f4ac9a187a8ea1] django_guid.
↪middleware Deleting 0d1c3919e46e4cd2b2f4ac9a187a8ea1 from _guid
DEBUG 2020-01-14 15:40:48,955 [99d44111e9174c5a9494275aa7f28858] django_guid.
↪middleware Deleting 99d44111e9174c5a9494275aa7f28858 from _guid
```

If we have a close look, we can see that the first request is completely done before the second and third arrives. However, the second and third request arrives at the exact same time, and since `gunicorn` is run with multiple workers, they are also handled concurrently. The result is logs that get mixed together, making them impossible to differentiate.

Now, depending on how you view your logs you can easily track a single request down. In these docs, try using `ctrl + f` and search for `99d44111e9174c5a9494275aa7f28858`

If you're logging to a file you could use `grep`:

```
  ~ cat demoproj/logs.log | grep 99d44111e9174c5a9494275aa7f28858

INFO 2020-01-14 15:40:48,955 [None] django_guid.middleware No Correlation-ID found in␣
↪the header. Added Correlation-ID: 99d44111e9174c5a9494275aa7f28858
INFO 2020-01-14 15:40:48,955 [99d44111e9174c5a9494275aa7f28858] demoproj.views This␣
↪is a DRF view log, and should have a GUID.
WARNING 2020-01-14 15:40:48,955 [99d44111e9174c5a9494275aa7f28858] demoproj.services.
↪useless_file Some warning in a function
DEBUG 2020-01-14 15:40:48,955 [99d44111e9174c5a9494275aa7f28858] django_guid.
↪middleware Deleting 99d44111e9174c5a9494275aa7f28858 from _guid
```

CHAPTER 7

Troubleshooting

7.1 Turn on Django debug logging

Set the logger to log DEBUG logs from django-guid:

```
LOGGING = {
    'loggers': {
        'django_guid': {
            'handlers': ['console'],
            'level': 'DEBUG',
        },
    },
}
```

7.2 Run Django with warnings enabled

Start `manage.py runserver` with the `-Wd` parameter to enable warnings that normally are suppressed.

```
python -Wd manage.py runserver
```

7.3 Use the demo project as a reference

There is a simple demo project available in the `demoproj` folder, have a look at that to see best practices.

7.4 Read the official logging docs

Read the official docs about logging.

7.5 Ask for help

Still no luck? Create an issue on GitHub and ask for help.

CHAPTER 8

Upgrading Django-GUID 2.x.x to 3.x.x

Upgrading to `Django>=3.1.1` and using async/ASGI requires you to use `Django-GUID` version 3 or higher.

In order to upgrade, you need to do the following:

8.1 1. Change Middleware

- **From:** `django_guid.middleware.GuidMiddleware`
- **To:** `django_guid.middleware.guid_middleware`

```
MIDDLEWARE = [
    'django_guid.middleware.guid_middleware',
    ...
 ]
```

8.2 2. Change API functions (if you used them)

From:

```
from django_guid.middleware import GuidMiddleware
GuidMiddleware.get_guid()
GuidMiddleware.set_guid('x')
GuidMiddleware.delete_guid()
```

To:

```
from django_guid import clear_guid, get_guid, set_guid
get_guid()
set_guid('x')
clear_guid()  # Note the name change from delete to clear
```

CHAPTER 9

Publish django-guid

This site is intended for the contributors of `django-guid`.

9.1 Publishing to test-PyPi

Before publishing a new version of the package, it is advisable that you publish a test-package. Among other things, this will flag any possible issues the current interation of the package might have.

Please note, to publish a test-package, you need to have a test-pypi API token.

Using the API token, you can publish a test-package by running:

```
poetry config repositories.test https://test.pypi.org/legacy/
poetry config pypi-token.test <api-token>
poetry publish --build --no-interaction --repository test
```

9.2 Publishing to PyPi

Publishing `django-guid` can be done by creating a github release in the `django-guid` repository. Before publishing a release, make sure that the version is consistent in `django_guid/__init__.py`, `pyproject.toml` and in the title of the actual publication. The title of the release should simply be the version number and the release body should contain the changelog for the patch.

9.3 Read the docs

Read the docs documentation can be built locally by entering the `docs` folder and writing `make html`. It requires that you have installed `sphinx` and the theme we're using, which is `sphinx_rtd_theme`. Both can be installed through `pip`.

CHAPTER 10

Changelog

10.1 3.2.1 - 13.12.2021

Changes can be seen here going forward.

10.2 3.2.0 - 04.12.2020

Features

- Added a new setting, `sentry_integration` to the Celery integration, which sets `transaction_id` for Celery workers.

10.3 3.1.0 - 18.11.2020

Features

- Added a new setting, `UUID_LENGTH`, which lets you crop the UUIDs generated for log filters.
- Added a new integration for tracing with Celery.

10.4 3.0.1 - 12.11.2020

Bugfix

- Importing an integration before a `SECRET_KEY` was set would cause a circular import.

10.5 3.0.0 - 28.10.2020 - Full Django3.1+(ASGI/async) support!

Brings full async/ASGI (as well as the old WSGI) support to Django GUID using ContextVars instead of thread locals.

Breaking changes

This version requires `Django>=3.1.1`. For previous versions of Django, please use `django-guid<3.0.0` (Such as `django-guid==2.2.0`).

If you've already implemented `django-guid` in your project and are currently upgrading to `Django>=3.1.1`, please see the upgrading docs.

10.6 2.2.0 - 04.11.2020

Features

- `IGNORE_URLS` setting which disables the middleware on a list of URLs.

Other

- Added docs for the new setting

10.7 2.1.0 - 03.11.2020

Features

- Integration module, which enables the users of `django_guid` to extend functionality.
- Added a integration for Sentry, tagging the Sentry issue with the GUID used for the request.

Other

- Added docs for integrations

10.8 2.0.0 - 02.03.2020

This version contains backwards incompatible changes. Read the entire changelog before upgrading

Deprecated

- `SKIP_CLEANUP`: After a request is finished, the Correlation ID is cleaned up using the `request_finished` Django signal.

Incompatible changes

- `django_guid` must be in `INSTALLED_APPS` due to usage of signals.

Improvements

- Restructured README and docs.

10.9 1.1.1 - 12.02.2020

Improvements

- Fixed `EXPOSE_HEADER` documentation issue. New release has to be pushed to fix PyPi docs.

10.10 1.1.0 - 10.02.2020

Features

- Added a `EXPOSE_HEADER` setting, which will add the `Access-Control-Expose-Headers` with the `RETURN_HEADER` as value to the response. This is to allow the JavaScript Fetch API to access the header with the GUID

10.11 1.0.1 - 08.02.2020

Bugfix

- Fixed validation of incoming GUID

Improvements

- Changed the `middleware.py` logger name to `django_guid`
- Added a WARNING-logger for when validation fails
- Improved README

Other

- Added `CONTRIBUTORS.rst`

10.12 1.0.0 - 14.01.2020

Features

- Added a `RETURN_HEADER` setting, which will return the GUID as a header with the same name

Improvements

- Added a Django Rest Framework test and added DRF to the `demoproj`
- Improved tests to also check for headers in the response
- Added tests for the new setting
- Added examples to `README.rst` and docs, to show how the log messages get formatted
- Added an API page to the docs
- Fixed the `readthedocs` menu bug

10.13 0.3.1 - 13.01.2020

Improvements

- Changed logging from f'strings' to %strings
- Pre-commit hooks added, including `black` and `flake8`
- Added `CONTRIBUTING.rst`
- Added github actions to push to `PyPi` with github tags

10.14 0.3.0 - 10.01.2020

Features

- Added a SKIP_CLEANUP setting

Improvements

- Improved all tests to be more verbose
- Improved the README with more information and a list of all the available settings

10.15 0.2.3 - 09.01.2020

Improvements

- Added tests written in *pytests*, 100% codecov
- Added Django2.2 and Django3 to github workflow as two steps
- Improved logging

10.16 0.2.2 - 21.12.2019

Improvements

- Removed the mandatory DJANGO_GUID settings in settings.py. Added an example project to demonstrate how to set the project up

10.17 0.2.1 - 21.12.2019

Improvements

- Workflow added, better docstrings, easier to read flow

10.18 0.2.0 - 21.12.2019

Features

- Header name and header GUID validation can be specified through Django settings

10.19 20.10.2019

- Initial release