django-webhook

Release 0.0.7

Dani Hodovic

Apr 21, 2024

CONTENTS

A plug-and-play Django app for sending outgoing webhooks on model changes.

Django has a built-in signal system which allows programmers to schedule functions to be executed on model changes. django-webhook leverages the signal system together with Celery to send HTTP requests when models change.

Suppose we have a User model:

```
class User(models.Model):
    name = models.CharField(max_length=50)
    age = models.PositiveIntegerField()
```

If a webhook is configured, any time the above model is created, updated or deleted django-webhook will send an outgoing HTTP request to a third party:

```
POST HTTP/1.1
host: webhook.site
user-agent: python-urllib3/2.0.3
django-webhook-uuid: 5e2ee3ba-905e-4360-94bf-18ef21c0e844
django-webhook-signature-v1:
django-webhook-request-timestamp: 1697818014

{
  "topic": "users.User/create",
  "object": {
    "id": 3,
    "name": "Dani Doo",
    "age": 30
  },
  "object_type": "users.User",
  "webhook_uuid": "5e2ee3ba-905e-4360-94bf-18ef21c0e844"
}
```

CHAPTER

ONE

FEATURES

- Automatically sends webhooks on model changes
- Leverages Celery for processing
- Webhook authentication using HMAC
- Retries with exponential backoff
- Admin integration
- Audit log with past webhook events
- Protection from replay attacks

CHAPTER

TWO

TABLE OF CONTENTS

2.1 Quickstart

2.1.1 Requirements

Django Webhook depends on Celery for background processing. Celery is the de-facto background processing system for Django.

Django-Webhook sends each webhook within the context of a Celery task. This allows us to offload webhook logic from Django and automatically retry failed requests.

To install Celery in your Django project see: https://docs.celeryq.dev/en/stable/django/first-steps-with-django.html

Make sure that your project has a Celery worker running. This component is in charge of sending webhooks.

2.1.2 Installation

To demonstrate and example the below code assumes we have a model called `Product` in an application called `Core`.

You don't have to include the code sample, any of your own models could work.

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=50)
```

Install the python package

```
pip install django-webhook
```

Add the app to your settings.py and whitelist models for which you want to send webhooks

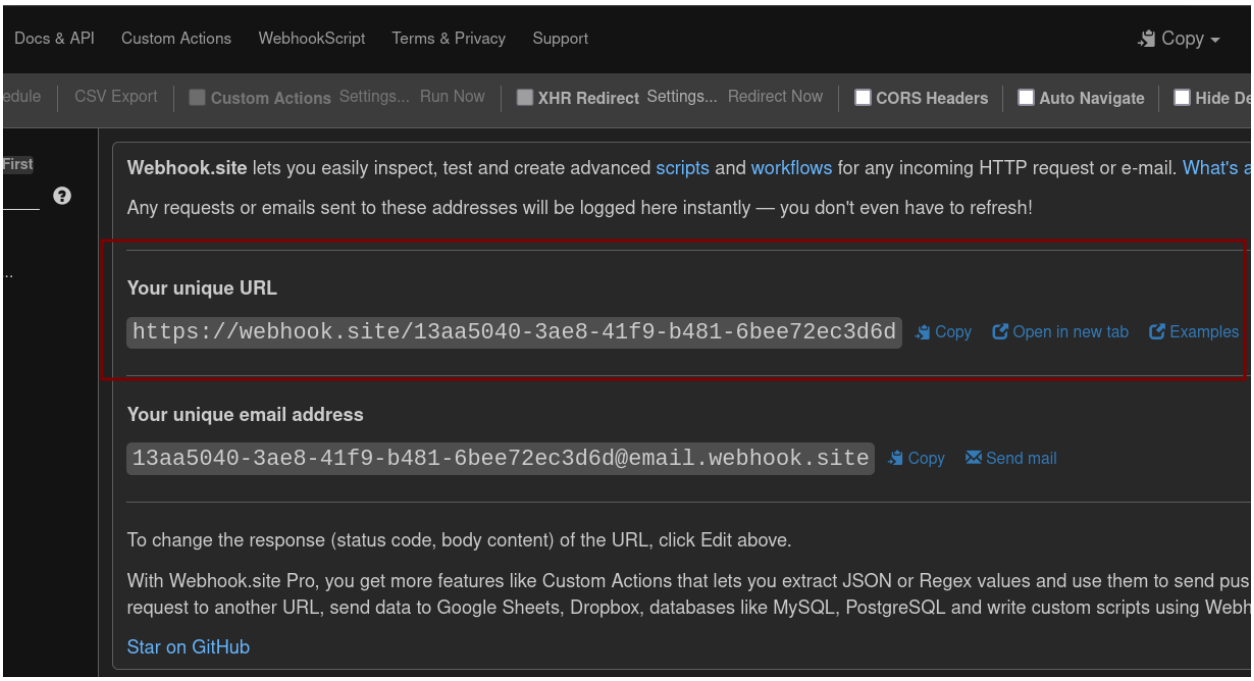
```
INSTALLED_APPS = [
    "django_webhook"
]

# Whitelist models for which we send webhooks
DJANGO_WEBHOOK = dict(MODELS=["core.Product", "users.User"])
```

Run the migrations

```
./manage.py migrate
```

2.1.3 Test outgoing webhooks

Visit https://webhook.site to create an inbox for your webhooks. Copy the unique URL which will be the destination for our webhook.



Configure an outgoing webhook for one of your models

```
./manage.py shell

>>> from django_webhook.models import Webhook, WebhookTopic
>>> webhook = Webhook(url="https://webhook.site/13aa5040-3ae8-41f9-b481-6bee72ec3d6d")
>>> webhook.save()
```

Set the topic to be triggered on create and update for your model.

```
>>> topics = [
        WebhookTopic.objects.get(name="core.Product/create"),
        WebhookTopic.objects.get(name="core.Product/update")
    ]
>>> webhook.topics.set(topics)
```

Finally create a new Product instance to trigger the webhook.

```
>>> from core.models import Product
>>> Product.objects.create(name="test")
```

django-webhook should send an outgoing HTTP request in the following format:

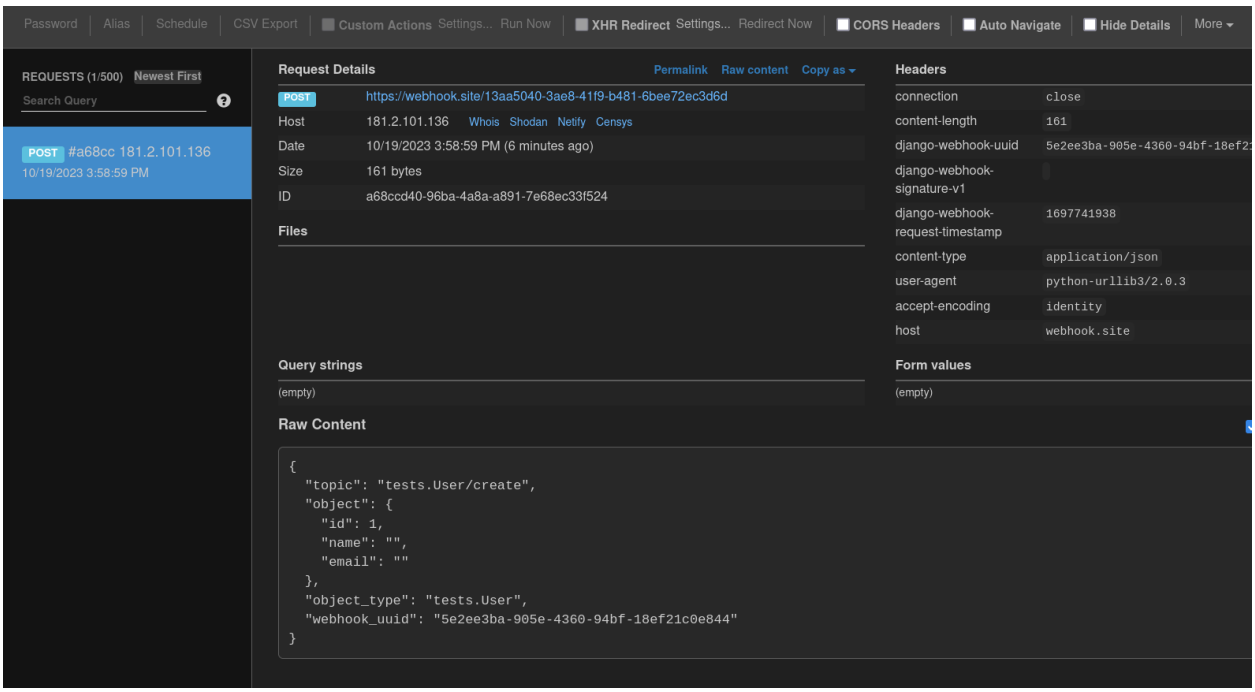
```
POST HTTP/1.1
host: webhook.site
user-agent: python-urllib3/2.0.3
django-webhook-uuid: 5e2ee3ba-905e-4360-94bf-18ef21c0e844
django-webhook-signature-v1:
```

(continues on next page)

(continued from previous page)

```
django-webhook-request-timestamp: 1697818014

{
  "topic": "core.Product/create",
  "object": {
    "id": 3,
    "name": "test",
  },
  "object_type": "core.Product",
  "webhook_uuid": "5e2ee3ba-905e-4360-94bf-18ef21c0e844"
}
```

Visit the page for your unique webhook where you can inspect the incoming HTTP request.



2.2 Configuring Celery

Celery is an open-source asynchronous task queue. It's a popular choice for processing tasks in the background in Django projects.

Django-Webhook uses Celery to offload the work of sending HTTP requests to a separate worker process. This ensures that we don't bog down the web workers and that we can retry failing webhooks.

Running a Celery worker is a requirement for running Django-Webhook. Without a Celery worker no webhooks will be sent.

By default Django-Webhook places tasks on the default Celery queue, called "celery". To consume tasks run a worker:

```
celery -A config.settings.celery:app worker -Q celery
```

2.2.1 Using a dedicated worker

If you want to process Django-Webhook tasks separately from your other tasks you need to use task routing, a separate queue and start a separate worker. This could be to:

- process webhook tasks faster by running multiple workers for the webhooks queue
- avoid clogging up the default tasks queue with webhook tasks
- rate limit how many webhooks to send per minute

In your Celery configuration file configure the `task_routes` property:

```
import os

from celery import Celery

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "config.settings.production")

app = Celery("myapp")
app.config_from_object("django.conf:settings", namespace="CELERY")

# Configure task routes
app.conf.task_routes = [
    [
        ("django_webhook.tasks.fire_webhook", {"queue": "webhooks"}),
    ],
]
```

With this route enabled webhook tasks will be routed to the "webhooks" queue, while all other tasks will be routed to the default queue (named "celery"). You now have to run a worker that consumes from the webhooks queue:

```
celery -A config.settings.celery:app worker -Q webhooks
```

The worker could also consume from multiple queues:

```
celery -A config.settings.celery:app worker -Q celery,webhooks,email
```

2.3 Example integrations

- Flask server integration
- Express.js server integration