django-spark Documentation

Release 0.3

Feinheit AG

Oct 29, 2018

Contents

Version 0.3

This is not supposed to be real documentation; it's more a reminder for myself.

The idea is that there are event sources and event handlers. Event sources may create a stream of `spark.api.Event` instances, where each event must have a `group` and a `key`. Additional data may be added to the `Event` as well. Keys are globally unique – events with the same key are still only processed exactly once. Groups are used to determine which handlers handle a certain event.

Event handlers are functions which are called once per `spark.api.Event` instance if the event's group matches the event handler's regex.

CHAPTER 1

Some usage example code

Given a challenge, create events for the challenge (the specifics do not matter):

```
from datetime import date
from spark import api

def events_from_challenge(challenge):
    if not challenge.is_active:
        return

    yield {
        "group": 'challenge_created',
        "key": 'challenge_created_%s' % challenge.pk,
        "context": {"challenge": challenge},
    }

    if (date.today() - challenge.start_date).days > 2:
        if challenge.donations.count() < 2:
            yield {
                "group": 'challenge_inactivity_2d',
                "key": 'challenge_inactivity_2d_%s' % challenge.pk,
                "context": {"challenge": challenge},
            }

    if (challenge.end_date - date.today()).days <= 2:
        yield {
            "group": 'challenge_ends_2d',
            "key": 'challenge_ends_2d_%s' % challenge.pk,
            "context": {"challenge": challenge},
        }

    if challenge.end_date < date.today():
        yield {
            "group": 'challenge_ended',
            "key": 'challenge_ended_%s' % challenge.pk,
```

(continues on next page)

(continued from previous page)

```
            "context": {"challenge": challenge},
        }
```

Send mails related to challenges (uses django-authlib's `render_to_mail`):

```
from authlib.email import render_to_mail

def send_challenge_mails(event):
    challenge = event["context"]["challenge"]
    render_to_mail(
        # Different mail text per event group:
        "challenges/mails/%s" % event["group"],
        {
            "challenge": challenge,
        },
        to=[challenge.user.email],
    ).send(fail_silently=True)
```

Register the handlers:

```
class ChallengesConfig(AppConfig):
    def ready(self):
        # Prevent circular imports:
        from spark import api

        api.register_group_handler(
            handler=send_challenge_mails,
            group=r'^challenge',
        )

        Challenge = self.get_model('Challenge')

        # All this does right now is register a post_save signal
        # handler which runs the challenge instance through
        # events_from_challenge:
        api.register_model_event_source(
            sender=Challenge,
            source=events_from_challenge,
        )
```

Now, events are generated and handled directly in process. Alternatively, you might want to handle events outside the request-response cycle. This can be achieved by only registering the model event source e.g. in a management command, and then sending all model instances through all event sources, and directly processing those events, for example like this:

```
from spark import api

api.register_model_event_source(...)

# Copied from the process_spark_sources management command inside
# this repository
for model, sources in api.MODEL_SOURCES.items():
    for instance in model.objects.all():
        for source in sources:
            api.process_events(api.only_new_events(source(instance)))
```

- Documentation

- Github

CHAPTER 2

Change log

2.1 Next version

2.2 0.3 (2018-10-29)

- Changed API events to be dictionaries instead of `types.SimpleNamespace` objects. The top level of the dictionary normally contains `key` and `group` keys used by django-spark and an additional `context` dictionary with arbitrary data.
- Added a new `Event.objects.create_if_new` queryset method which understands event dictionaries.
- Added a new `spark.spark_generators` app for configuring spark generators using Django's administration interface.
- Changed the API contract for sources and sinks: Sources and sinks are both **NOT** responsible for only letting new events through. A new `spark.api.only_new_events` filtering iterator has been added which only yields events that haven't been seen yet.
- Added a new `spark.spark_mails` app for transactional mails.

2.3 0.2 (2018-10-16)

- Reformatted the code using black.
- Added a testsuite and some documentation.

2.4 0.1 (2017-12-19)

- Initial public version.