
brabeion Documentation

Release 0.1.dev11

Eldarion

Aug 10, 2017

Contents

1	Using brabeion	3
1.1	Getting Started	3
1.2	Asynchronous Badges	4
2	Models in brabeion	7
3	Templatetags in brabeion	9
3.1	badge_count	9
3.2	badges_for_user	9
4	Signals in brabeion	11
	Python Module Index	13

`brabeion` is a powerful, extensible, reusable application that provides support for awarding badges to users in Django. It provides simple abstractions over awarding users badges for completing tasks, including multi-level badges, and repeatable badges, making it super simple to add badges to a Django project.

Getting Started

`class brabeion.base.Badge`

brabeion works by allowing you to define your badges as subclasses of a common `Badge` class and registering them with `brabeion`. For example if your site gave users points, and you wanted to award three ranks of badges based on how many points a user had your badge might look like this:

```
from brabeion import badges
from brabeion.base import Badge, BadgeAwarded

class PointsBadge(Badge):
    slug = "points"
    levels = [
        "Bronze",
        "Silver",
        "Gold",
    ]
    events = [
        "points_awarded",
    ]
    multiple = False

    def award(self, **state):
        user = state["user"]
        points = user.get_profile().points
        if points > 10000:
            return BadgeAwarded(level=3)
        elif points > 7500:
            return BadgeAwarded(level=2)
        elif points > 5000:
            return BadgeAwarded(level=1)
```

```
badges.register(PointsBadge)
```

There are a few relevant attributes and methods here.

slug

The unique identifier for this `Badge`, it should never change.

levels

A list of the levels available for this badge (if this badge doesn't have levels it should just be a list with one item). It can either be a list of strings, which are the names of the levels, or a list of `brabeion.base.BadgeDetail` which have both names and description.

events

A list of events that can possibly trigger this badge to be awarded. How events are triggered is described in further detail below.

multiple

A boolean specifying whether or not this badge can be awarded to the same user multiple times, currently if this badge has multiple levels this must be `False`.

award (*self*, ***state*)

This method returns whether or not a user should be awarded this badge. `state` is guaranteed to have a `"user"` key, as well as any other custom data you provide. It should return either a `BadgeAwarded` instance, or `None`. If this `Badge` doesn't have multiple levels `BadgeAwarded` doesn't need to be provided an explicit level.

Note: `BadgeAwarded.level` is 1-indexed.

Now that you have your `PointsBadge` class you need to be able to tell `brabeion` when to try to give it to a user. To do this, any time a user *might* have received a badge just call `badges.possibly_award_badge` with the name of the event, and whatever state these events might need and `brabeion` will handle the details of seeing what badges need to be awarded to the user:

```
from brabeion import badges

def my_view(request):
    if request.method == "POST":
        # do some things
        request.user.profile.award_points(15)
        badges.possibly_award_badge("points_awarded", user=request.user)
    # more view
```

By default badges will be awarded at the current time, if you need to override the award time of the badge you can pass a `force_timestamp` keyword argument to `possibly_award_badge()`.

Asynchronous Badges

Note: To use asynchronous badges you must have `celery` installed and configured.

If your `Badge.award()` method takes a long time to compute it may be prohibitively expensive to call during the request/response cycle. To solve this problem `brabeion` provides an `async` option to `Badges`. If this is `True` `brabeion` will defer calling your `award()` method, using `celery`, and it will be called at a later time, by another process (read the [celery docs](#) for more information on how `celery` works).

Because `award()` won't be called until later you can define a `freeze()` method which allows you to provide and additional state that you'll need to compute `award()` correctly. This may be necessary because your `Badge` requires some mutable state.

```
class AddictBadge(Badge):
    # stuff
    async = True

    def freeze(self, **state):
        state["current_day"] = datetime.today()
        return state
```

In this example badge the user will be awarded the `AddictBadge` when they've visited the site every day for a month, this is expensive to calculate so it will be done outside the request/response cycle. However, what happens if they visit the site right before midnight, and then the `award()` method isn't actually called until the next day? Using the `freeze` method you can provide additional state to the `award()` method.

CHAPTER 2

Models in brabeion

```
class brabeion.models.BadgeAward(models.Model)

    user
        The user who was awarded this badge.

    awarded_at
        The datetime that this badge was awarded at.

    slug
        The slug for the Badge this refers to.

    name
        The name for the Badge this refers to, for the appropriate level.

    description
        The description for the Badge this refers to, for the appropriate level.
```

Templatetags in brabeion

brabeion offers a number of templatetags for your convenience, which are available in the `brabeion_tags` library.

badge_count

This tag returns the number of badges that have been awarded to this user, it can either set a value in context, or simple display the count. To display the count its syntax is:

```
{% badge_count user %}
```

To get the count as a template variable:

```
{% badge_count user as badges %}
```

badges_for_user

This tag provides a `QuerySet` of all of a user's badges, ordered by when they were awarded, descending, and makes them available as a template variable. The `QuerySet` is composed of `BadgeAwarded` instances.

```
{% badges_for_user user as badges %}
```


CHAPTER 4

Signals in brabeion

brabeion makes one signal available to developers.

`brabeion.signals.badge_awarded`

This signal is sent whenever a badge is awarded to a user. It provides a single argument, `badge`, which is an instance of `BadgeAwarded`.

b

`brabeion.models`, [7](#)

`brabeion.signals`, [11](#)

`brabeion.templatetags.brabeion_tags`, [9](#)

A

[award\(\)](#), 4
[awarded_at](#) ([brabeion.models.BadgeAward](#) attribute), 7

B

[badge_awarded](#) (in module [brabeion.signals](#)), 11
[BadgeAward](#) (class in [brabeion.models](#)), 7
[brabeion.base.Badge](#) (built-in class), 3
[brabeion.models](#) (module), 7
[brabeion.signals](#) (module), 11
[brabeion.templatetags.brabeion_tags](#) (module), 9

D

[description](#) ([brabeion.models.BadgeAward](#) attribute), 7

E

[events](#), 4

L

[levels](#), 4

M

[multiple](#), 4

N

[name](#) ([brabeion.models.BadgeAward](#) attribute), 7

S

[slug](#), 4
[slug](#) ([brabeion.models.BadgeAward](#) attribute), 7

U

[user](#) ([brabeion.models.BadgeAward](#) attribute), 7