Django-achievements Documentation

Release 0.0.8

Olivier Girardot

May 30, 2017

Contents

Contents:

CHAPTER 1

Introduction

Catchphrase

Django-achievement's goal is to provide a framework and an engine for handling User achievements in a django app.

What does it mean ?

It means two things that you can guess from the words i'm using :

- It's a framework : It means you can define classes that will be called and taken into account by the application.
- and it's an engine : It means that there's an entity responsible for checking and managing as a whole every achievement that can be handle by the system. Bonus : You can call it when you need to;

CHAPTER 2

Tutorial

Installing

Best way to install django-achievements is to user pip :

```
pip install django-achievements
```

of course you can also download it directly from PyPi (http://pypi.python.org/pypi/django-achievements) and install it :

```
python setup.py install
```

Then to install it on your Django project just add, '*achievements*' to your *INSTALLED_APPS* and synchronize your database with it. If you're using *South*, you need to execute *django-achievements*'s migrations :

```
./manage.py syncdb
# or with south :
./manage.py syncdb --migrate
```

And just with that, you're all set to create your first achievement.

Create your first achievement

To create your first achievement, you only need to create a Python class with the following attributes :

- key (unique) : the key string of your achievement, it's what will be used to identify uniquely your achievement.
- name : The full name of your achievement
- description : The full description of what your achievement is all about
- bonus : the bonus (integer) associated to unlocking this achievement

- evaluate(args): the evaluate function is a callback that must return a boolean in order to decide if the achievement has been unlocked.

Here is an example of Achievement definition :

```
class UsernameAchievement(object):
    name = "Username achivement"
    key = "username"
    description = "Handles when a user changes its username"
    bonus = 15.0
    def evaluate(self, user, *args, **kwargs):
        if create_username_from_email(user.email) != user.username:
            return True
        else:
            return False
```

The name of class must end with the string Achievement, this is in order to be able to define more than just *Achievements* classes into a given file.

Register it to the Engine

To register it into the engine just use the '*ACHIEVEMENT_CLASSES*' attribute in your settings like that :

```
#=============================================================================
# Achievements conf
#=============================================================================
ACHIEVEMENT_CLASSES = ['accounts.handlers', 'backend.handlers']
```

The file where the classes are defined are not important, but try to avoid conflicts of naming by avoiding to use the name *achievements.py*. When you have defined a new achievement, for the engine to create the proper objects into your database, execute :

```
./manage.py syncdb
# or if you're using south :
./manage.py migrate achievements
```

This step is mandatory when you're creating new achievement classes and when you update their properties.

Check it when you need

The engine now is aware of the achievements you want to handle. On startup it will create the achievements objects into your database, and store the path to the callback you want to associate it with.

Now the engine's running on startup of your application and you need a way to call it for it to check if given a precise context an achievement will be unlocked. Here's how you'll call it :

```
from achievements.engine import engine
engine.check_achievement(user=my_current_user, key="achievement_key")
```

Of course *my_current_user* can be a User object you extracted from database or your common *request.user*.

Don't hesitate to check achievements even for anonymous users, it won't be taken into account anyway.

The achievement *key* is the most important part as it will tell the engine with *callback* to call and if you have other arguments your achievement may need to evaluate, given the context if the achievement has been unlocked, you can pass them as optional arguments, they will be transmitted to the *Achievement* class.

Check automatic

Achievements includes a handy middleware class for automatic achievement checks.

Register the middleware class in your settings as usual :

```
MIDDLEWARE_CLASSES += ['achievements.middleware.AutoAchievementChecker']
```

Define the http request methods (get, post, put, delete) on which the automatic middleware checker should be run :

```
# for example check on changing http request methods only
ACHIEVEMENT_MIDDLEWARE_REQUEST_METHODS = ['post', 'put', 'delete']
```

Get the score of any user

It may be useful (or vital) to get at anytime a user's score given the achievements he may have unlocked, here's the way to do that :

```
from achievements.utils import get_user_score
score = get_user_score(request.user)
```

Handle the score yourself

If by any chance you want to add the score to your User profile, or just want to trigger a notification when the user has unlocked a new achievement, you can connect to the dedicated signal *achievement_unlocked* :

```
# first define your callback :
def update_score(sender, user, achievement, *args, **kwargs):
    print "Gotcha"

# then connect it to the django signal :
from achievements.signals import achievement_unlocked
achievement_unlocked.connect(update_score)
```

Use Celery

To use Celery, we assume you have installed the *django-celery* app, and you just need to set this attribute in your settings, and you don't have to change anything to your code :

```
ACHIEVEMENT_USE_CELERY = True
```

This attribute is false by default.

Enjoy.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search