django-simplekeys Documentation

Release 0.5.3

James Turk

Dec 20, 2018

Contents

1 Features **3**

2 Requirements **5**

3 Further Reading **7**
- 3.1 Getting Started . . . 7
- 3.2 Advanced Usage . . . 9
- 3.3 Changelog . . . 12

4 Indices and tables **15**

Python Module Index **17**

django-simplekeys is a reusable Django app that provides a simple way to add API keys to an existing Django project, regardless of API framework.

- GitHub: https://github.com/jamesturk/django-simplekeys
- Documentation: https://django-simplekeys.readthedocs.io/en/latest/

CHAPTER 1

Features

- Token bucket rate limiting, for limiting requests/second with optional bursting behavior.
- Quota-based rate limiting (e.g. requests/day)
- Ability to configure different usage tiers, to give different users different rates/quotas.
- Ability to configure different 'zones' so that different API methods can have different limits. (e.g. some particularly computationally expensive queries can have a much lower limit than cheap GET queries)
- Provided views for very simple email-based API key registration.

CHAPTER 2

Requirements

- simplekeys currently supports Django 1.8 through Django 1.11. Future versions will drop support for Django <= 1.10.
- simplekeys is tested against Python 2.7, Python 3.5, and Python 3.6.

CHAPTER 3

Further Reading

3.1 Getting Started

3.1.1 Step 1- Configure Settings

- Add `simplekeys` to `INSTALLED_APPS` as you would any app.
- Be sure to run the `migrate` command after adding the app to your project.
- If you plan on using the provided registration view be sure you've set DEFAULT_FROM_EMAIL.
- Unless otherwise configured simplekeys will use Django's `CACHE['default']` to store ephemeral information used for rate-limiting. Depending on your use case it may be desirable to configure Django's cache with this in mind.
- simplekeys doesn't require any other settings, but there are plenty of other things you can configure. See *Advanced Settings* for details.

3.1.2 Step 2- Configure Default Zones & Tiers

Typically this will be done via the Django admin, if you're not using the Django admin it is possible to do this via the shell but that is beyond the scope of this documentation.

For the simplest usage it is sufficient to create a `Tier` and a `Zone` with the slug `default`. You should then edit the `Tier` to have a `Limit` configuration for the default zone.

For more detail on these concepts, see *Models*.

Warning: If you do not create an association between a `Tier` and a `Zone` then users will not be able to access any views that you define as being within a given `Zone`.

3.1.3 Step 3- Protect API Views

There are two ways to let Django know which views are protected and assign them to particular zones:

- `simplekeys.middleware.SimpleKeysMiddleware` allows you to define views by regex, similar to a urlconf. The downside is that this check has to happen on every request.
- You can also use the `key_required()` decorator to annotate certain views, this will be more efficient, but requires you to decorate views individually- which may be difficult depending upon your setup.

If you add `simplekeys.middleware.SimpleKeysMiddleware` to your installed middleware, by default it will protect every view. Unless your app is very simple (and you don't use the Django admin, etc.) you probably also want to add the `SIMPLEKEYS_ZONE_PATHS` setting.

`SIMPLEKEYS_ZONE_PATHS` is a list of tuples that looks like:

```
SIMPLEKEYS_ZONE_PATHS = [
    ('/api/v1/legislators/geo/', 'geo'),
    ('/api/v1/', 'default'),
]
```

This would place the `/api/v1/legislators/geo/` method into the 'geo' zone and all other `/api/v1/` methods into the default zone. These strings are matched with `re.match`- so you can design complex rules as needed.

Alternatively, if you choose to use the `key_required()` decorator, it might look like:

```
@key_required()
def simple_api_view(request):
    ...
```

`simplekeys.decorators.`**`key_required`**(*zone=None*)

Decorator that specifies that a view should require an API key and will be throttled according to the rules of a specified zone.

If `zone` parameter is omitted `SIMPLEKEYS_DEFAULT_ZONE` will be used (`default` unless overriden)

3.1.4 Step 4- Add Registration Views (optional)

simplekeys provides two class-based views that can be used together to provide a simple email-based workflow for obtaining API keys.

You can also create `Key` instances via the Django admin or within your own app, but these views are provided to accomodate a common flow out of the box.

You can use these two views by adding them to your `urls.py` like so:

```
url(r'^register/$', RegistrationView.as_view()),
url(r'^confirm/$', ConfirmationView.as_view()),
```

These two views are designed to work without any parameters but take quite a few optional parameters should you wish to customize their behavior.

See *Class-based Views* for more details on overiding the defaults.

3.2 Advanced Usage

3.2.1 Understanding Tiers & Zones

Some of the real power of simplekeys comes from using multiple zones and tiers.

- Zones allow you to give different levels of access & rate limits to different API methods.
- Tiers allow you to give different types of users different access across zones.
- Limits connect these two, each Tier defines appropriate rate limits and quotas for Zones it has access to.

As noted in the introduction, be sure each tier has an association with each zone unless you intend to prevent a given tier from accessing that zone at all.

3.2.2 Models

class simplekeys.models.**Tier**

slug
internal name of the Tier (e.g. `default`)

name
human-readable name of the Tier (e.g. `Default API Users`)

simplekeys has the concept of different keys having different usage limits, this is done with Tiers.

For example: you could define a silver tier and a gold tier, where gold tier keys have higher rate limits due to being trusted users.

Each API key is associated with a single tier.

If you only want to have one tier, it is recommended you call it `default`.

class simplekeys.models.**Zone**

slug
internal name of the Zone (e.g. `default`)

name
human-readable name of the Zone (e.g. `Default API Methods`)

simplekeys allows you to define different API zones. This is intended to allow you to specify different rate limits for different API methods.

Each tier can have different limitations defined on different zones. If a tier doesn't have any association with a zone it will not be allowed to access it, giving you fine grained control over which keys can access which endpoints.

If all of your API methods will have the same rate limits, you can just use a single zone. Default settings assume you call this `default`.

class simplekeys.models.**Limit**

tier
ForeignKey relationship to a *Tier*

zone
ForeignKey relationship to a *Zone*

quota_requests
How many requests to allow overall each day/month (according to `quota_period`).

This is independent of the `requests_per_second` and `burst_size`, and sets a hard quota that the user cannot exceed in the given period.

quota_period
Specifies if `quota_requests` is a daily or a monthly limitation.

This relies upon whatever `SIMPLEKEYS_RATE_LIMIT_BACKEND` you're using storing data long enough. If you're using the default cache-based backend you may want to configure `SIMPLEKEYS_CACHE_TIMEOUT` to be longer than a month if you're using a monthly quota.

requests_per_second
Limits how quickly a user can access the API, regardless of their quota.

It is possible for the user to briefly exceed this rate up to their `burst_size` after which they'll be throttled back to this rate until they back off for a sufficient period of time.

For specifics on this behavior you can read about token bucket rate-limiting.

burst_size
The maximum number of requests allowed in a burst situation. This should be configured to be somewhat higher than `requests_per_second`.

class simplekeys.models.**Key**
Keys are the tokens given to users to access the API.

key
The actual key used for access, by default these are randomly generated UUIDs.

status

- 'u' - Unactivated, requests will not be allowed, but validation will be. (If you're using the default views keys are created in this state and updated once the user confirms their email address.)
- 'a' - Activated, requests will be allowed
- 's' - Suspended, requests will not be allowed, neither will activation.

tier
ForeignKey relationship to *Tier* indicating which Tier this key has access to.

email
Email address associated with the key.

name
Name of individual associated with the key.

organization
(Optional) organization associated with the key.

website
(Optional) website associated with the key.

usage
(Optional) Description of intended usage of the API key.

3.2.3 Class-based Views

class simplekeys.views.**RegistrationView**
Presents user with a simple form they can fill out to obtain a key.

Upon successful submission of the form a non-active key is created for the user, and an email is sent with a link that the user must click to verify their email address.

Optional Arguments:

template_name Name of template to use for registration form.

This template should render the `form` context variable and provide a `<form method="POST" action=".">` to send the form contents back to the view for processing.

Default: `simplekeys/register.html`

email_subject Subject of email sent to user.

Default: `API Key Registration`

email_message_template Name of template to use for plain text email.

This template is provided the newly-created `Key` instance as well as the fully-qualified `confirmation_url` based on the optional parameter described below.

Default: `simplekeys/confirmation_email.txt`

from_email Email address from which to send.

Default: `DEFAULT_FROM_EMAIL`

tier tier that will be used for permissions/rate limiting for this view.

Default: `default`

redirect URL, view name, or model to redirect to after registration is complete.

See django's redirect shortcut for options.

confirmation_url URL to include in email, should match URL of *`ConfirmationView`*

If URL is a relative URL, will be appended to the current *Site* <https://docs.djangoproject.com/en/1.11/ref/contrib/sites/>`_

Default: `/confirm/`

class simplekeys.views.**ConfirmationView**

After filling out the registration form the user is emailed a link to confirm their email address. The user must visit this link to finish the process and activate their API key.

This view is quite simple, when accessed via GET it will render `confirmation_template_name` and then when accessed via a successful POST will show `confirmed_template_name`.

If an attempt is made to access this view with invalid activation data this view returns an `HttpResponseBadRequest` 400 error.

Optional Arguments:

confirmation_template_name This template should render the `form` context variable and provide a `<form method="POST" action=".">` to send the form contents back to the view for processing.

All fields on the form render as hidden, you can simply ask the user to press submit to proceed.

Default: `simplekeys/confirmation.html`

confirmed_template Default: `simplekeys/confirmed.html`

This template is shown after the key is sucessfully activated.

It is passed the newly activated `Key` instance, be sure to let the user know what their API key is!

3.2.4 Advanced Settings

SIMPLEKEYS_DEFAULT_ZONE If you use the `key_required()` without a `zone` parameter, simplekeys will consider your view part of this zone.

Default: `default`

SIMPLEKEYS_ZONE_PATHS Used in conjunction with `SimpleKeysMiddleware` to associate request paths with zones.

Default: `[('.*', 'default')]`

SIMPLEKEYS_HEADER HTTP header that `SimpleKeysMiddleware` and `key_required()` will check for presence of API key.

Default: `HTTP_X_API_KEY`

SIMPLEKEYS_QUERY_PARAM HTTP query parameter that `SimpleKeysMiddleware` and `key_required()` will check for presence of API key. (This check occurs after `SIMPLEKEYS_HEADER` check.)

Default: `apikey`

SIMPLEKEYS_RATE_LIMIT_BACKEND String representing full import path to a rate limit backend.

Default: `simplekeys.backends.CacheBackend`

SIMPLEKEYS_CACHE `settings.CACHE` entry to use for `simplekeys.backends.CacheBackend`

Default: `default`

SIMPLEKEYS_CACHE_TIMEOUT Timeout for entries created by `simplekeys.backends.CacheBackend`

Default: `25*60*60` (25 hours)

SIMPLEKEYS_ERROR_NOTE Will be included in error messages, a useful place to direct users to an email address to address their rate quota/etc.

3.2.5 Custom Rate Limiting Backends

Keeping track of how many times a key is used requires some semi-permanent storage that is relatively cheap to access.

Since Django's existing cache framework provides easy access to such data stores, that is the default backend.

There is also a memory backend, which stores the rate-limiting data locally, this is not intended for production use and should only be used if you know what you're doing.

In both of these cases, the rate-limiting data is somewhat ephemeral, a process restarting or a cache getting cleared will allow users to make more calls than you might otherwise have expected. If this does not meet your needs it may be necessary to explore other options, or you may be able to simply write a custom backend that writes to your storage of choice.

If you write a rate limiting backend that you think others might find useful, please consider contributing back to the project.

3.3 Changelog

3.3.1 0.5.3

2018-12-19

- reactivating an active key doesn't show unhelpful error message anymore

3.3.2 0.5.2

2018-10-29

- add missing migration

3.3.3 0.5.1

2018-05-18

- added abillity to search keys to admin

3.3.4 0.5.0

2017-12-12

- exportkeys management command
- usagereport management command

3.3.5 0.4.2

2017-05-22

- error message tweak
- addition of SIMPLEKEYS_ERROR_NOTE

3.3.6 0.4.0

2017-05-22

- refactored decorator and middleware to be independent
- added SIMPLEKEYS_ZONE_PATHS for middleware

3.3.7 0.3.0

2017-04-21

- made organization optional & added optional website & usage fields to Key (requires migration!)

3.3.8 0.2.0

2017-04-18

- documented & cleaned up API and made more consistent with Django

3.3.9 0.1.0

- initial prototype with MVP functionality for Open States.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

simplekeys.models, 9
simplekeys.views, 10

Index

B

burst_size (simplekeys.models.Limit attribute), 10

C

ConfirmationView (class in simplekeys.views), 11

E

email (simplekeys.models.Key attribute), 10

K

Key (class in simplekeys.models), 10
key (simplekeys.models.Key attribute), 10

L

Limit (class in simplekeys.models), 9

N

name (simplekeys.models.Key attribute), 10
name (simplekeys.models.Tier attribute), 9
name (simplekeys.models.Zone attribute), 9

O

organization (simplekeys.models.Key attribute), 10

Q

quota_period (simplekeys.models.Limit attribute), 10
quota_requests (simplekeys.models.Limit attribute), 10

R

RegistrationView (class in simplekeys.views), 10
requests_per_second (simplekeys.models.Limit attribute), 10

S

simplekeys.decorators.key_required() (built-in function), 8
simplekeys.models (module), 9
simplekeys.views (module), 10
slug (simplekeys.models.Tier attribute), 9
slug (simplekeys.models.Zone attribute), 9
status (simplekeys.models.Key attribute), 10

T

Tier (class in simplekeys.models), 9
tier (simplekeys.models.Key attribute), 10
tier (simplekeys.models.Limit attribute), 9

U

usage (simplekeys.models.Key attribute), 10

W

website (simplekeys.models.Key attribute), 10

Z

Zone (class in simplekeys.models), 9
zone (simplekeys.models.Limit attribute), 9