Django Auto Healthchecks Documentation

Release 0.1.5

Cronitor.io

Feb 15, 2017

Contents

CHAPTER 1

Installation

1.1 Stable release

To install Django Auto Healthchecks, run this command in your terminal:

```
$ pip install django_auto_healthchecks
```

This is the preferred method to install Django Auto Healthchecks, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.2 From sources

The sources for Django Auto Healthchecks can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/cronitorio/django_auto_healthchecks
```

Or download the tarball:

```
$ curl  -OL https://github.com/cronitorio/django_auto_healthchecks/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 2

Usage

To use Django Auto Healthchecks in a project:

```
import django_auto_healthchecks
```

CHAPTER 3

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/cronitorio/django_auto_healthchecks/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

3.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

3.1.4 Write Documentation

Django Auto Healthchecks could always use more documentation, whether as part of the official Django Auto Healthchecks docs, in docstrings, or even on the web in blog posts, articles, and such.

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/cronitorio/django_auto_healthchecks/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.2 Get Started!

Ready to contribute? Here's how to set up *django_auto_healthchecks* for local development.

1. Fork the *django_auto_healthchecks* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django_auto_healthchecks.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django_auto_healthchecks
$ cd django_auto_healthchecks/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 django_auto_healthchecks tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/cronitorio/django_auto_healthchecks/pull_requests and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

```
$ py.test tests.test_django_auto_healthchecks
```

CHAPTER 4

Module reference

4.1 django_auto_healthchecks

4.1.1 Module contents

class `django_auto_healthchecks.`**`Healthcheck`**(*route=None*, *args=None*, *kwargs=None*, *current_app=None*, *name=None*, *key=None*, *method=u'GET'*, *querystring=None*, *body=None*, *headers=None*, *cookies=None*, *assertions=None*, *tags=None*, *note=None*, *interval_seconds=None*, *timeout_seconds=None*)

Bases: `object`

:type healthchecks.Healthcheck

`display_name`()
Retrieve the effective name of this healthcheck.

`resolve`()
Because the route cannot be reversed into a URL at the same time its defined, we delay route resolution until we are ready to submit the healthchecks to the API.

`serialize`()
Serialize current instance details into valid API payload :return: dict

exception `django_auto_healthchecks.`**`HealthcheckError`**
Bases: `exceptions.RuntimeError`

:type healthchecks.HealthcheckError Healthcheck definition exception

`django_auto_healthchecks.`**`put`**(*healthchecks=()*)
Optionally, use the put method to batch create/update healthcheck definitions

`django_auto_healthchecks.`**`url`**(*regex*, *view*, *healthcheck=None*, ***kwargs*)
url is a drop-in replacement for django URL that adds a new healthcheck kwarg

4.1.2 django_auto_healthchecks.healthchecks module

class django_auto_healthchecks.healthchecks.**Healthcheck**(*route=None*, *args=None*, *kwargs=None*, *current_app=None*, *name=None*, *key=None*, *method=u'GET'*, *querystring=None*, *body=None*, *headers=None*, *cookies=None*, *assertions=None*, *tags=None*, *note=None*, *interval_seconds=None*, *timeout_seconds=None*)

Bases: object

Define a healthcheck using a django url route that will be put to the Cronitor API

route (str): Name of a url route. If omitted, must be set before resolve().

args (list|tuple): Optional args to be passed to *reverse()* this route. Cannot be used with *kwargs*.

kwargs (dict): Optional kwargs to be passed to *reverse()* this route. Cannot be used with *args*.

current_app (str): If the app is namespaced or route name is not unique the *current_app* argument is needed for *reverse()*.

name (str): Optional name for this monitor. If none is provided, a name will be generated.

key (str): Optional monitor key. Use this to tie to an existing healthcheck.

method (str): Request method used when performing this healthcheck.

querystring (dict): Optional querystring parameters that will be appended to the URL

body (str): Request body sent when performing this healthcheck if method is PUT, POST or PATCH.

headers (dict): Optional request headers that will be sent when performing this healthcheck.

cookies (dict): Optional cookies that will be sent when performing this healthcheck.

assertions (dict): Optional assertions for this monitor. See API docs for details.

tags (list|tuple): Optional tags for this monitor.

note (string): Optional note that will be attached to this monitor.

interval_seconds (int): Optional interval between healthcheck tests. If omitted, a default will be used.

timeout_seconds (int): Optional timeout for this request, maximum of 10 seconds.

display_name()
Retrieve the effective name of this healthcheck.

resolve()
Because the route cannot be reversed into a URL at the same time its defined, we delay route resolution until we are ready to submit the healthchecks to the API.

serialize()
Serialize current instance details into valid API payload :return: dict

exception django_auto_healthchecks.healthchecks.**HealthcheckError**
Bases: exceptions.RuntimeError

class django_auto_healthchecks.healthchecks.**HealthcheckUrl**(*path*, *querystring*)
Bases: object

display = None
:type unicode Shorter, prettier version of URL for display

url = None
:type unicode Fully qualified URL that will be used in healthcheck

class django_auto_healthchecks.healthchecks.**IdempotentHealthcheckClient**
Bases: object

Put enqueued healthchecks to the Cronitor API.

drain()
Drain enqueued healthchecks and return a list of distinct Healthcheck objects :return: List[Healthcheck]

enqueue(*healthcheck*)
Add a healthcheck instance to a queue for later processing. healthcheck (Healthcheck): Healthcheck instance to enqueue

put(*additional_healthchecks=None*)

django_auto_healthchecks.healthchecks.**put**(*healthchecks=()*)
Batch create-or-update health checks with supplied list of Healthcheck instances. Invoke from your deploy script, or add healthchecks for third-party apps without having to hack their code. :param healthchecks: list[Healthcheck] of Healthcheck objects These healthchecks will be merged with any defined in

urls.py file(s). See https://cronitor.io/docs/django-health-checks for details.

django_auto_healthchecks.healthchecks.**url**(*regex*, *view*, *healthcheck=None*, ***kwargs*)
Drop-in replacement for django.conf.urls.url to create and update a Cronitor healthcheck when your app restarts. See https://cronitor.io/docs/django-health-checks for details. regex (str): Route regex, passed to *django.conf.urls.url()* view (mixed): Attached view for this route, passed to *django.conf.urls.url()* healthcheck (Healthcheck): Define your healthcheck with a *Healthcheck()* instance. :return RegexURLPattern

4.1.3 django_auto_healthchecks.apps module

class django_auto_healthchecks.apps.**HealthchecksAppConfig**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

Attach into the Django app startup. At this point the URL resolver cache has not been warmed yet.

name = 'django_auto_healthchecks'

ready()

CHAPTER 5

Index

- genindex
- modindex
- search

d

D

E

H

I

N

P

R

S

U