Djamon (Django Monitoring) Documentation

Release 1

Paul Horner

May 13, 2013

CONTENTS

This system is used to monitor the current status of web-based resources.

In simple terms, you need to setup various URLs to test important elements of your site, such as the database, file server, etc etc. Then point this system at each of those URLs, and it will notify you if something goes wrong.

The source code is hosted on Bitbucket at https://bitbucket.org/paulh/djamon/.

Note: It relies on a cron job to look for the URLs. At the moment it checks every minute during work time, and every 5 minutes outside of work time. It only records an 'outage' when any one element is still not available after 3 attempts.

CHAPTER

ONE

CONTENTS:

1.1 Django MVC

1.1.1 Models

class `monitor.models.`**`Group`**(**args*, ***kwargs*)
Group sites by type

class `monitor.models.`**`Outage`**(**args*, ***kwargs*)
Records when an outage has taken place, including all the things that have gone down and the dates when it was resolved etc

class `monitor.models.`**`OutageSites`**(**args*, ***kwargs*)
Records which sites are effected by the outage

class `monitor.models.`**`Site`**(**args*, ***kwargs*)
Core information about a site

class `monitor.models.`**`SiteView`**(**args*, ***kwargs*)
Records when it has been accessed and the code returned

This will get big very quickly, so there will need to be a database purge script written. This shouldn't be done on save because the normal monitoring system should do no extra work.

Using a UUID rather than an auto-increment for the pk

1.1.2 Views

`monitor.views.`**`group_data`**(*request*, **args*, ***kwargs*)
Returns a json object containing details for each site being monitored in this particular group

`monitor.views.`**`home`**(*request*)
Show monitoring results for all systems being monitored.

This doesn't do any parsing, it just gets the latest information from the database. A seperate script does the db update work.

`monitor.views.`**`login_user`**(*request*)
Logs the user in

`monitor.views.`**`outage`**(*request*, **args*, ***kwargs*)
Shows details of an outage, or the current outage if there is one

`monitor.views.`**`outages`**`(`*request*, **args*, ***kwargs*`)`
Lists all outages recorded on the system. You can click on each one to view the details of it.

`monitor.views.`**`site_data`**`(`*request*, **args*, ***kwargs*`)`
Returns details of the last 10 requests to view this particular site as a json object

1.1.3 Templatetags

readable_time

`monitor.templatetags.readable_time.`**`readable_time`**`(`*value*`)`
Pass it a time in seconds and it returns a human readable string showing the hours, minutes and seconds

number_suffix

`monitor.templatetags.number_suffix.`**`ordinal`**`(`*value*`)`
http://code.activestate.com/recipes/576888-format-a-number-as-an-ordinal/

Converts zero or a *postive* integer (or their string representations) to an ordinal value.

```
>>> for i in range(1,13):
...     ordinal(i)
...
u'1st'
u'2nd'
u'3rd'
u'4th'
u'5th'
u'6th'
u'7th'
u'8th'
u'9th'
u'10th'
u'11th'
u'12th'

>>> for i in (100, '111', '112',1011):
...     ordinal(i)
...
u'100th'
u'111th'
u'112th'
u'1011th'
```

paginator

`monitor.templatetags.paginator.`**`paginate`**`(`*obj*`)`
Takes an object `obj` and puts the pagination links on screen.

This is an inclusion tag rather than just calling the fragment because it is totally generic - it uses `{{obj}}` rather than the name of the variable.

1.2 Python Scripts

1.2.1 Cron Job

`crontab`

The cron job needs to have a record in your cron tab to work.

To open your crontab (in Ubuntu), run:

```
crontab -e
```

That opens an editor allowing you to edit your crontab. At present, the following has been set:

```
*/1 9-16 * * 1-5 python ~/private_html/django/djamon/scripts/cronjob.py
*/5 0-8 * * 1-5 python ~/private_html/django/djamon/scripts/cronjob.py
*/5 17-23 * * 1-5 python ~/private_html/django/djamon/scripts/cronjob.py
*/5 * * * 0,6 python ~/private_html/django/djamon/scripts/cronjob.py
```

This ensures that `cronjob.py` is called by the server every minute Monday-Friday between 9.00a.m. and 4.59p.m., and every 5 minutes at all other times.

`cronjob.py`

This script is called by the cron job.

The cron job checks that the internet connection is working before checking your site(s). This means that it won't record an outage if the problem might not be localised to your web pages. You will need to add some information to settings.py to set this up:

```
# Add a list of URLs that you know are usually working. If they are all down, then we assume the inte
SANDBOX_URLS = ('http://www.bbc.co.uk/', 'http://www.google.co.uk/')

GOOD_CODES = (200,301,304)
```

The good codes are a list of HTTP responses that it deems as being 'up'. The sandbox urls are a list of URLs that usually work.

1.2.2 `my_shortcuts`

`scripts.my_shortcuts.`**`HttpResponseJson`**(*biggus_dictus*)

My own function this one. But it's based on the above render_to_json.

Effectively you pass it a dictionary/list thing and it renders a JSON object over HTTP.

"I have a gweat fwiend in Wome called Biggus Dickus"

And this takes a big dict... or a small dict. Or a list. It doesn't discriminate as long as the object is serializable as JSON.

I'd like to write something that loops through the queryset and renders as a JSON object, regardless of type, so splitting datetimes into datetimes and all subqueryset objects into dicts/lists too. But that might be one for another time.

`scripts.my_shortcuts.`**`paginate`**(*the_list*, *num*, *page*)

Simple function to paginate a list

Pass the queryset, the number of objects to return and details of the current page and it returns the paginated list of objects, e.g.:

```
groups = paginate(group_list, 25, request.GET.get('page'))
```

This means you don't need to do this for all paginated pages

`scripts.my_shortcuts.`**`render_to_hxr_response`**(**args*, ***kwargs*)

Use this instead of render_to_response if you want to access using an XMLHttpRequest

This is because ipads return a response status of 0 (error) when it's cached content, meaning that it just doesn't work.

`scripts.my_shortcuts.`**`render_to_json`**(**args*, ***kwargs*)

Use this instead of render_to_response if you want to use JSON information.

This will stop IE from caching the results and make sure that browsers treat the data as JSON rather than HTML.

From http://djangosnippets.org/snippets/468/

1.2.3 `status_code`

`scripts.status_code.`**`get_status_code`**(*url*)

This function gets the status code of a website (host) by requesting HEAD data from the host. This means that it only requests the headers. If the host cannot be reached or something else goes wrong, it returns None instead.

Thanks to Evan Fosmark - http://stackoverflow.com/questions/1140661/python-get-http-response-code-from-a-url

Works as following:

```
>>> url = 'http://www.bbc.co.uk/'
>>> get_status_code(url)
200
>>> url = 'http://www.bbc.co.uk/404-page/'
>>> get_status_code(url)
404
>>> url = 'http://www.bbc.co.uk/500-error/'
>>> get_status_code(url)
500
```

If it takes more than 5 seconds to get the header we assume that there is a problem with the server and returns a 500 code. The server might be OK, but if it's taking that long to respond, we should assume that it's down - our users won't tolerate it taking that long.

1.2.4 `unique_slugify`

See http://djangosnippets.org/snippets/690/

`scripts.unique_slugify.`**`unique_slugify`**(*instance*, *value*, *slug_field_name='slug'*, *queryset=None*, *slug_separator='-'*, *case='lower'*)

Calculates and stores a unique slug of `value` for an instance.

`slug_field_name` should be a string matching the name of the field to store the slug in (and the field to check against for uniqueness).

`queryset` usually doesn't need to be explicitly provided - it'll default to using the `.all()` queryset from the model's default manager.

1.3 Alerts

The Monitoring System is setup to alert you when your site is unavailable. To do this, it sends out a message whenever the site has been down for more than three attempts.

The reason it waits for three attempts is simple. 9 times out of 10, it will right itself after one or two attempts, in which case you needn't have bothered looking for a solution. If it's still down after 3 attempts it's likely that it's a more serious problem, and it's fairly likely that it will be alerting you to the problem quicker than any end-user could.

The system can send two types of alert - an email and a Tweet.

1.3.1 Email Alerts

Some settings need to be added to your settings.py file for this system to work.

You will then need to add the following information:

```
FROM_EMAIL = ''
TO_EMAIL = []
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = '@gmail.com'
EMAIL_HOST_PASSWORD = ''
```

This sets up gmail as the mail host. The `TO_EMAIL` list is a list of all of the email addresses that should get copies of alerts.

1.3.2 Twitter

There are two big benefits of using twitter.

The first is that it will notify your customers of outages in real time.

The second is that you can set it up to text you whenever that account tweets. This means that you get an SMS text message notifying you that the site is down and it doesn't cost you anything. This is awesome.

To send messages to twitter, you will need to set up Django to work with Twitter. First install the python twitter library:

```
pip install twitter
```

Then set up an OAUTH consumer key/token thingy. This links your twitter account to this app. The following needs to be added to settings.py:

```
TWITTER = True #False if you don't want to tweet
CONSUMER_KEY = ''
CONSUMER_SECRET = ''
OAUTH_TOKEN = ''
OAUTH_SECRET = ''
```

If you setup a twitter account, then go to the developer tools section of twitter, it talks you through setting this up. It's not that hard.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX