Testing Plan

Team Lucky 7

Laura Barber, Venessa Johansen-Barrera, Michael Blackburn

Introduction

ReadTheDocs is a documentation hosting service for the open source community to make project documentation fully searchable and easy to find. It also has versioning control so that documents can be built from tags and branches of code in the repository. The documents can be imported using any major version control system. Docs can also be built automatically when committing code.

We chose this project because of its approachability, well documented code and existing testing framework. As part of our team's testing framework we will be working with and testing five methods in the python file *nginx-smoke-test.py* located in *readthedocs.org/deploy*. These methods check to see whether or not a url is indeed a functioning url (returns http code 200 OK) and if it is served by nginx, django, or perl.

Requirements Traceability

The requirements of each of the five methods seem to be relatively the same. In order to pass the requirements for the served_by_nginx method the url must be served by nginx and we check that by making sure the specified url returns 200, indicating that it is valid. This return statement holds true for the other methods as well. We will be establishing five separate urls for each of the five methods where they will all be individually tested.

Tested Items

The items will be tested with the format outlined below:

test number
requirement tested
method tested
test inputs
expected outcome

Proposed Testable Methods

In *nginx-smoke-test.py* there are five, straightforward, testable methods, they all accept urls and test for different HTTP request responses. Generally it tests for a 200 response (200 OK is the standard response for successful HTTP requests. The actual response will depend on the request method used.)

1. **def** **served_by_nginx**(url):

Checks if the url is served nginx and returns 200.

2. **def served_by_django**(url):
   Checks if the url is served by django and returns 200.
3. **def served_by_perl**(url):
   Checks if the url is served by perl and returns 200.
4. **def served**(url):
   Checks if the url returns 200.
5. **def redirected**(url, location):
   Accepts a url and location, and checks if the url will redirect to the location.

Five Potential Test Cases

1

requirement: url is served by nginx and return 200 OK
method: **served_by_nginx**(url)
test input: https://pip.readthedocs.org/en/latest/
expected outcome: True

2

requirement: url is served by nginx and return 200 OK
method: **served_by_nginx**(url)
test input: http://docs.fabfile.org/en/latest/
expected outcome: True

3

requirement: url is served by nginx and return 200 OK
method: **served_by_nginx**(url)
test input: http://cs.cofc.edu
expected outcome: False

4

requirement: url is served by django and return 200 OK
method: **served_by_django**(url)
test input: https://readthedocs.org/security/
expected outcome: True

5

requirement: url is served by django and return 200 OK
method: **served_by_django**(url)
test input: http://www.microsoft.com

expected outcome: False

Test Recording Procedures

We will redirect the output of each of our tests into separate files and format them for increased readability. The test outputs will also be compared to the expected outcome (oracle) and the success or failure of each test will also be recorded. In order to make sure that the program is working as it should be, we have found a website that checks to see what a url is served in: https://www.whatismyip.com/server-headers-check/

Software Requirements

- Ubuntu 14.04
- Git
- Python 2.7 or newer
- Virtual Python Environment (virtualenv)
- pip
- zlib1g-dev
- python development headers (python-dev)
- The dependencies outlined in pip_requirements.txt (name and version)
    - These should be installed with pip *install -r pip_requirements.txt*

      pip==1.5.6
      virtualenv==1.11.6
      docutils==0.11
      Sphinx==1.2.2
      django==1.6.6
      django-tastypie==0.11.1
      django-haystack==2.1.0
      celery-haystack==0.7.2
      django-guardian==1.2.0
      django-extensions==1.3.8
      South==0.8.4
      djangorestframework==2.3.14
      pytest-django==2.6.2
      requests==2.3.0
      slumber==0.6.0
      lxml==3.3.5
      redis==2.7.1
      hiredis==0.1.2
      celery==3.0.24
      django-celery==3.0.23
      django-allauth==0.16.1

bzr==2.5b4
mercurial==2.6.3
github2==0.5.2
httplib2==0.7.7
elasticsearch==0.4.3
pyquery==1.2.2
django-gravatar2==1.0.6
doc2dash==1.1.0
pytz==2013b
beautifulsoup4==4.1.3
Unipath==0.2.1
django-kombu==0.9.4
django-secure==0.1.2
mimeparse==0.1.3
mock==1.0.1
simplejson==2.3.0
sphinx-http-domain==0.2
Distutils2==1.0a3
distlib==0.1.2
django-cors-headers==0.11

- The following C libraries:
- Latex (texlive-full)
- libevent (libevent-dev)
- dvipng
- graphviz
- libxslt1.1 (libxslt1-dev)
- libxml2-dev

Testing Schedule

Initial testing of our selected methods will take place over a three-week period and in two phases:

- Phase one – September 20, 2014 – October 1, 2014
- Phase one will consist of identification and selection of specific methods to test. As specified, testing will initially focus on five methods. Identifying these methods will be the initial step, followed by the selection of specific test elements and cases within those methods. After identification and selection, the remaining time in phase one will consist of writing actual test cases for the selected methods.
- Phase two – October 1, 2014 – October 14, 2014
- Phase two will primarily consist of refactoring test methods and elements that fail during the initial phase. Any errors in initial code will be corrected. Completion of this phase will result in completion of Deliverable 3.

Constraints

Potential constraints will be mostly related to our schedules. Especially regarding fall break being due right before deliverable #4. However, we all communicate regularly and have decided that weekends are the best to meet up for the most part. We have also discussed that if meeting during the weekend becomes an issue then we will update each other on our weekly schedules to find the most efficient replacement. Other than our conflicting schedules there are no other constraints aside from lack of experience. Nonetheless, we are working on gaining more experience by learning through the processes created from specifications of the deliverables.