Testing Framework for ReadTheDocs

TEAM LUCKY 7

Laura Barber | Venessa Johansen-Barrera | Michael Blackburn

Software Engineering
Fall 2014

Table of Contents

Chapter 1 2

Introduction

Chapter 2 4

Test Planning

Chapter 3 6

The Testing Framework

Chapter 4 9

Test Cases

Chapter 5 12

Fault Injections

Chapter 6 15

Reflections

Appendix 16

Requirements

What is ReadTheDocs?

Read the Docs is a document hosting service for the open source community to make project documentation fully searchable and easy to find. Documents can be imported using any of the major version control systems – Git, Subversion, Mercurial, and Bazaar. It also supports webhooks, so that documents can be built when you commit code. There is also support for versioning, so that documents can be built from tags and branches of code in a repository. A full list of features may be found here.

After reviewing several other open source projects, we chose this project because of its approachability, well-documented code and existing testing framework. As part of our team's testing framework we worked with and tested five methods that checked to see whether or not a URL is indeed a functioning URL and if it is served by nginx, django, or perl.

Preliminary Work

Based on the organization of readthedocs.org we began with the "Getting Started" instructions. As we soon discovered, that actually builds the documentation to be utilized by ReadTheDocs (RTD) by using Sphinx to parse restructured text (.rst) into html format. So after a few confused starts we moved to the "Installation" instructions, which were the ones relevant to building and installing RTD.

After installing and creating a virtual environment (*virtualenv*) directory to house the python dependencies, of which there were several. These dependencies were required to build the project and run the tests. Thankfully there was a pip_requirements.txt file which could be run with pip install -r pip_requirements.txt. However we had to install pip as well (easy enough: *sudo apt-get install pip*).

There were several more false starts following this regarding installing dependencies from that text file. Further searching revealed that there were additional (not mentioned in the installation documentation) dependencies on C libraries. There were a few major libraries which people would generally have installed prior to using RTD. Found on the "Build Process" page was a list of packages installed in the build environment, seen below.

- Latex (texlive-full)
- libevent (libevent-dev)
- dvipng
- graphviz
- libxslt1.1
- libxml2-dev

In addition to these C libraries we needed the python development headers which were not included with Ubuntu. This was slightly confusing as Python is included by default, but does not come with the *python-dev* package. One command *sudo apt-get install texlive-full libevent-dev dvipng graphviz libxslt1-dev libxml2-dev python-dev* solved all of our dependency and error issues.

These dependency issues caused initial problems in getting the project up and running on each team member's computer. After several weeks of struggle, it was discovered that these missing dependencies

were the root of the problems. After adding them, the remaining team members were able to run the software appropriately and without issue.

Originally the command had libxslt1.1 being installed but this did not fix the problem with errors. It introduced another one: fatal error: libxml/xmlversion.h: No such file or directory. After a pip search libxml we discovered another version libxslt1-dev and upon installing that were able to proceed with no more errors.

Running Existing Tests

RTD is organized in such a way that most of the commands will be run through a manage.py file. Normally the file would be executed with a simple *./manage.py subcommand [options] [args]* command via terminal. However some errors were encountered when attempting to do this. After some minor investigation we recalled that the dependencies were installed using *sudo pip install _*, and by changing the terminal call to *sudo python manage.py subcommand* we eliminated any errors. When installing the dependencies with *sudo* we created elevated permissions that prevented manage.py from accessing those dependencies if it wasn't also run with *sudo*.

Running *sudo python manage.py syncdb* began the process to sync our database and prompted us to create a superuser account for Django. Next we ran *sudo python manage.py migrate* to migrate the dependencies that we installed. Lastly we ran *sudo python manage.py loaddata test_data* to get test data provided from RTD. After this we could run *sudo python manage.py runserver* and open localhost:8000/ in our browsers to view a sample RTD page.

Everything was now set up for testing. Running the existing (but admittedly not extensive) testing framework was simple compared to the build process. Once dependencies were sorted out everything seemed easier. With *sudo python manage.py test rtd_tests* we could run all of the provided tests. The resulting output was slightly intimidating and unreadable due to the amount of information being pushed into the terminal. Attempting to pipe it into a file took a bit of experimentation, *sudo python manage.py test rtd_tests >> test.txt* did not result in all of the terminal output being pushed into the test file as we hoped for. A cursory google search resulted in a helpful StackOverflow inquiry that directed us to use *&>* instead of >> which would direct both the stdout and stderr outputs to a file.

The test command ran 125 tests in 17.195 seconds with 13 failures and 1 error. The successful tests didn't seem to have much verbosity in their output. Each of the failures and the one error did provide some details.

We later discovered that this was due to the fact that the tests were not run while the test_data server was running. Once we loaded up the test_data again and ran the server with *sudo python manage.py runserver* we needed to build some test documentation. In this instance we used (as helpfully instructed on the website) the latest documentation for 'pip' by running *sudo python manage.py update_repos pip*. This updated our localhost version with the pip documentation.

After doing this we ran the tests again getting some more tangible results this time. The information output into the terminal was similar to the first runtime, but this time the tests for building the project passed and a test project of "Read the Docs" was built and visible on our localhost server.

Tested Methods

For our testing framework, we decided on three methods in the python file *nginx-smoke-test.py* located in *readthedocs.org/deploy* (Shown below). 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. The requirements of each of the three methods were relatively similar. For example, in order to pass the requirements for the **served_by_nginx** method, the URL must be served by nginx and return 200 OK. This return statement holds true for the other methods as well. We established five separate URLs for each of the three methods where they were all individually tested.

1. **def served_by_nginx(**url**):**
   Checks if the URL is served by nginx and returns 200 OK.
2. **def served_by_django(**url**):**
   Checks if the URL is served by django and returns 200 OK.
3. **def served_by_perl(**url**):**
   Checks if the URL is served by perl and returns 200 OK.

Preliminary Tests

We then prepared five preliminary test cases to confirm whether our proposed methods were indeed useable, and to confirm our expected results:

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

Determining Oracles

To ensure that the program worked as it should, we found a website that checks to see what a URL is served in: https://www.whatismyip.com/server-headers-check/. We used this site to independently bench test each URL and compare our output to those given by whatismyip.com. Surprisingly, our initial tests found discrepancies between the results for some of the URLs in our list and their results on whatismyip.com.

Testing Schedule

Initial testing took place over a period of approximately one month, which was split into two phases:

- **Phase one** consisted of identification and selection of specific methods to test. As specified, testing initially focused on three methods. Identifying these methods was 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 consisted of writing actual test cases for the selected methods.
- **Phase two** primarily consisted of refactoring test methods and elements that failed during the initial phase. Any errors in initial code were corrected during phase two.

Building the Framework

Our initial attempts to test the chosen methods were thwarted by unconventional naming. Our test file, nginx-smoke-test.py, uses dashes in its name, which are invalid for importing in python. We were able to create a proxy file that effectively changed the dashes to underscores, allowing the file to be imported for access to its methods. After changing the path from an absolute to a relative path, further construction of the framework and continued testing proceeded with minimal difficulty.

Description of the Framework

We created a script, runAllTests.sh, which handles the entire framework per our given specification. First, the script will check if the temp directory is empty, if it is not empty then the files it contains will be removed. Next, the script reads from a file that contains some static CSS and HTML elements for styling and readability of the output. Then we define the start of a loop that will run 25 times. For each portion of the loop the script will:

- Create an array
- Populate the array with the contents of the appropriate testCase file
- Take the information contained in the array and create a table in output.html
- Run the actual tests by calling test.py which will run the provided method with the provided test url
- Create the output cell in output.html and color them according to True/False
- Check if the test passed or failed by comparing the oracle and output results
- Color entire test row according to result of the test
- Then the loop will end and finally open the results in the default browser.

The driver *test.py* contains only one simple method: *runMethod(method,url).* Which will call the appropriate method to be tested. There is also some code that detects if this file is being executed. If it is being executed then it will call *runMethod* with **sys.argv[1:]* or in other terms, the command line arguments provided by the script.

Framework Structure

The testing framework was structured based on the specifications given to us. The file structure is as follows:

```
/Lucky7
        /Deliverables
                Lucky7_Deliverable1.pdf – Lucky7_Deliverable5.pdf
        /Lucky7_TESTING
                /docs
                        README.txt
                /oracles
                /output
                /project
                        /src
                                nginx-smoke-test.py
```

/reports
    style.css
/scripts
    runAllTests.sh
/temp
    output.html
/testCases
    testCase01.txt – testCase25.txt
/testCasesExecutables
    nginx_smoke_test.py
    test.py

File Details

Deliverable.pdf: Chaptered reports created at each specified milestone of the project.

README.txt: A how-to of getting and running the testing framework. The contents will be in a section below.

nginx-smoke-test.py: The original file from the ReadTheDocs project that we are testing.

style.css: A short file containing css and html elements for constructing the test output.

runAllTests.sh: The script for running the testing framework, described in more detail above.

output.html: The output of the tests run, will be removed each time the script is run.

testCase.txt: The files containing specifications for each test case, will be detailed in a section below.

nginx_smoke_test.py: Proxy file for allowing proper Python imput.

test.py: Framework driver, detailed above.

How-To (README.txt)

Requirements

Python 2.7

(Check using python --version)

Ubuntu 14.04

Pip

Install dependencies

sudo apt-get install python-pip

Note: Given that we are only testing a small portion of the project, we only need to cover a few of its dependencies. These are automatically managed by installing pip (python package manager) using the above command.

Getting the Project and Running the Script

svn co https://svn.cs.cofc.edu/repos/CSCI3622014/Lucky7

cd Lucky7

cd Lucky7_TESTING

./scripts/runAllTests.sh

Results should open in default browser.

Test Case Formatting

- Each testCase file must follow a specific format
- **Naming:**
  - The name of the file must be testCase followed by a number
  - If the number is a single digit [0-9] it must be prepended with a 0
    - Example: 1 would become 01.
  - The extension of the file must be .txt
  - Full file example: testCase01.txt
- **File Content:** (The format must be as follows, with each part on a single line)
  - test number
  - requirement
  - method name
  - url being tested
  - oracle
- **NOTE:** the oracle must be in the format "True" or "False" (including the capital letter). There must also be a character turn after this line (hit enter).
- **Example Test file** (testCase01.txt)
  - 1
  - url is served by nginx and return 200 OK
  - served_by_nginx
  - https://pip.readthedocs.org/en/latest/
  - True
  -

Navigate to the top level TESTING directory of the project (Lucky7_TESTING) in the terminal. In order to run the tests type in *./scripts/runAllTests.sh.* The tests should then run automatically and the results will be opened in the default browser.

Tested Items

The items were tested and the results were output in the format outlined below:

- Test number
- Requirement tested
- Method tested
- Test inputs
- Expected outcome

Completing the Test Cases

Upon completion of our initial work with the framework, both it and our script were functional, but not to the given specification. After receiving in-class feedback, we completely rewrote the script so that it handled all of the testing. We also simplified the driver so that it would only accept the method name and item being tested. Additionally, instead of having all of the tests in one file, we broke them down into one text file for each test, each following the "Tested Items" format outlined previously. After updating the original six test cases, we added nineteen more to complete the required twenty-five test cases. We then ensured that the result of each test was compared to the defined oracle. Otherwise, it was a simple matter of some additional formatting changes being applied to the output results and further organization of the framework itself.

Coming up with the twenty-five test cases noted below was one of our easier decisions of this project. We knew that we would be testing methods served_by_nginx, served_by_django, and served_by_perl, as those were methods that were easily testable. So our first five test cases were based off of our familiarity of a URL, and knowing what that particular URL was served under. To verify that our knowledge and our framework testing were correct outside of the methods, we used whatismyip.com as previously noted. We generated the additional nineteen test cases using the same methods, choosing several URLs for each, all with the same structure of using URLs we were familiar with and bench-checking our results against our expectations at the above site. For further test case creation, we would suggest doing the same: pick several known and/or random URLs to test the methods along with an outside source to bench check the results.

Comprehensive List of Test Cases

Below is a comprehensive list of all twenty-five test cases used during the full testing of the framework. Note that each case will follow the "Tested Items" format outlined at the end of the previous chapter.

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

6. **requirement:** url is served by django and return 200 OK
   method: **served_by_django**(url)
   test input: https://cs.cofc.edu
   expected outcome: False

7. **requirement:** url is served by nginx and return 200 OK
   method: **served_by_nginx**(url)
   test input: http://phpmyadmin.readthedocs.org/ja/latest/
   expected outcome: True

8. **requirement:** url is served by nginx and return 200 OK
   method: **served_by_nginx**(url)
   test input: http://phpmyadmin.readthedocs.org/cs/latest/
   expected outcome: True

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

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

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

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

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

14. **requirement:** url is served by nginx and return 200 OK
    method: **served_by_nginx**(url)
    test input: http://docs.pylonsproject.org/projects/pyramid-amon/en/latest/genindex.html
    expected outcome: True

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

16. **requirement:** url is served by django and return 200 OK
    method: **served_by_django**(url)
    test input: https://readthedocs.org/api/v1/?format=json
    expected outcome: True

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

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

19. **requirement:** url is served by perl and return 200 OK
    method: **served_by_perl**(url)
    test input: http://www.perl.org/
    expected outcome: False

20. **requirement:** url is served by perl and return 200 OK
    method: **served_by_perl**(url)
    test input: https://ericholschercom.readthedocs.org/
    expected outcome: False

21. **requirement:** url is served by perl and return 200 OK
    method: **served_by_perl**(url)
    test input: http://www.microsoft.com
    expected outcome: False

22. **requirement:** url is served by nginx and return 200 OK
    method: **served_by_nginx**(url)
    test input: http://www.microsoft.com
    expected outcome: False

23. **requirement:** url is served by perl and return 200 OK
    method: **served_by_perl**(url)
    test input: http://my.cofc.edu
    expected outcome: False

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

25. **requirement:** url is served by django and return 200 OK
    method: **served_by_django**(url)
    test input: http://my.cofc.edu
    expected outcome: True

Choosing Faults

Given the design of the methods being tested for our project, there was little to change to break the code. When deciding on what to choose as our fault injections we had countless options. Some of those options included changing the HTTP status codes for valid URLs from 200 to some other code, changing the predefined lists of valid served URLs to other, valid or invalid URLs, and simply injecting syntax errors. All of our choices for breaking the code allowed individual cases to be broken, without breaking the entire framework. We made the decision to go with changing HTTP status codes, changing ands/ors and a few syntax errors. After discussing the steps of what could be done for each option, we decided that focusing on these three would be most beneficial to our project and our knowledge. So from here, we started our fault injections by focusing on one method and one group of tests at a time. We settled on inducing test failures in three different ways.

- Change HTTP status codes from 200 OK to anything else.
- Change *and* to *or,* vice versa.
- Inject syntax errors by using '=' instead of '==' (A common mistake.)

We created failures by editing the code found in the file *nginx-smoke-test.py.* We planned to have the following tests return either errors or improper results:

Tests 1-14

This test normally looks at the method **served_by_nginx**(url), and requires that the url is served by nginx and returns 200 OK.

```
11  def served_by_nginx(url):
12      """Return True if url returns 200 and is served by Nginx."""
13      r = requests.get(url, allow_redirects=False)
14      status = (r.status_code == 207)
15      nginx = ('x-served' in r.headers and r.headers['x-served'] == 'Nginx')
16      return all([status, nginx])
17
```

Usually these tests would all return true, but by changing the status code on line 14 from *200* to *207* they will return false. It still properly checks that the server is served by nginx, but it now checks for a 207 status code (Multi-Status: The message body that follows is an XML message and can contain a number of separate response codes, depending on how many sub-requests were made.) This will then give a False return, resulting in a failed test.

Test	Requirement	Method	Input	Oracle	Output	Result
1		served_by_nginx	https://pip.readthedocs.org/en/latest/	True	False	✖
2		served_by_nginx	http://docs.fabfile.org/en/latest/	True	False	✖
3		served_by_nginx	https://pip.readthedocs.org/en/latest/usage.html	True	False	✖
4		served_by_nginx	https://pip.readthedocs.org/en/1.4.1/	True	False	✖
5		served_by_nginx	https://pip.readthedocs.org/en/1.4.1/news.html	True	False	✖
6		served_by_nginx	http://docs.fabfile.org/en/latest/usage/execution.html	True	False	✖
7		served_by_nginx	http://phpmyadmin.readthedocs.org/ja/latest/	True	False	✖
8		served_by_nginx	http://phpmyadmin.readthedocs.org/cs/latest/	True	False	✖
9		served_by_nginx	http://phpmyadmin.readthedocs.org/en/latest/	True	False	✖
10		served_by_nginx	https://ericholschercom.readthedocs.org/en/latest/	True	False	✖
11		served_by_nginx	https://ericholschercom.readthedocs.org/about/	True	False	✖
12		served_by_nginx	https://ericholschercom.readthedocs.org/en/latest/about/	True	False	✖
13		served_by_nginx	http://docs.pylonsproject.org/projects/pyramid/en/latest/	True	False	✖
14		served_by_nginx	http://docs.pylonsproject.org/projects/pyramid-amon/en/latest/genindex.html	True	False	✖

Interestingly, note that Test 22 and 24 which use the **served_by_nginx**(url) method will still pass. As the tests are intended to return False.

22		served_by_nginx	http://www.microsoft.com	False	False	✔
23		served_by_perl	http://my.cofc.edu	False	False	✔
24		served_by_nginx	http://my.cofc.edu	False	False	✔

Tests 15-18 and 25

These tests usually looks at **served_by_django**(url) and requires that the url is served by django and returns 200 OK.

```
19  def served_by_django(url):
20      """Return True if url returns 200 and is served by Django. (NOT Nginx)"""
21      r = requests.get(url, allow_redirects=False)
22      status = (r.status_code == 200)
23      django = ('x-served' not in r.headers and r.headers['x-served'] == 'nginx-via-dja
24      return all([status, django])
25
```

By changing line 23 from the original *r.headers or r.headers[]* to *r.headers and r.headers[]* we get the error below. As the code should normally be checking that x-served, or the server headers, are not in the normal headers or are in the x-served headers. With this error they are now checking that the headers are in both. Resulting in the following error.

```
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    runMethod(*sys.argv[1:])
  File "test.py", line 5, in runMethod
    print(getattr(nginx_smoke_test, method)(url))
  File "../project/src/nginx-smoke-test.py", line 23, in served_by_django
    django = ('x-served' not in r.headers and r.headers['x-served'] == 'nginx-vi
a-django')
  File "/usr/local/lib/python2.7/dist-packages/requests/structures.py", line 77,
 in __getitem__
    return self._store[key.lower()][1]
KeyError: 'x-served'
Performing test 16.
```

The results are also displayed below.

15		served_by_django	https://readthedocs.org/search/	True		✖
16		served_by_django	https://readthedocs.org/api/v1/?format=json	True		✖
17		served_by_django	https://readthedocs.org/accounts/login/	True		✖
18		served_by_django	https://readthedocs.org/accounts/login/	True		✖
19		served_by_perl	http://www.perl.org/	False	False	✔
20		served_by_perl	https://ericholscher.com.readthedocs.org/	False	False	✔
21		served_by_perl	http://www.microsoft.com	False	False	✔
22		served_by_nginx	http://www.microsoft.com	False	False	✔
23		served_by_perl	http://my.cofc.edu	False	False	✔
24		served_by_nginx	http://my.cofc.edu	False	False	✔
25		served_by_django	http://my.cofc.edu	True		✖

Given that there are errors, the tests will not actually output a True/False result and comparing the oracle to null will give a failed test, as shown.

All Tests

By injecting a syntax error into line 30 (‘=’ instead of the proper ‘==’) it causes all tests to fail. All tests are still attempted by the framework, but they each return the error of SyntaxError: invalid syntax.

```
26  def served_by_perl(url):
27      """Return True if url returns 200 and is served by Perl."""
28      r = requests.get(url, allow_redirects=False)
29      status = (r.status_code == 302)
30      perl = ('x-perl-redirect' in r.headers and r.headers['x-perl-redirect'] = 'True')
31      return all([status, perl])
32
```

Similar to the previous error injection it will result in no outputs for all tests, causing all tests to fail, as seen below.

Test	Requirement	Method	Input	Oracle	Output	Result
1		served_by_nginx	https://pip.readthedocs.org/en/latest/	True		✖
2		served_by_nginx	http://docs.fabfile.org/en/latest/	True		✖
3		served_by_nginx	https://pip.readthedocs.org/en/latest/usage.html	True		✖
4		served_by_nginx	https://pip.readthedocs.org/en/1.4.1/	True		✖
5		served_by_nginx	https://pip.readthedocs.org/en/1.4.1/news.html	True		✖
6		served_by_nginx	http://docs.fabfile.org/en/latest/usage/execution.html	True		✖
7		served_by_nginx	http://phpmyadmin.readthedocs.org/ja/latest/	True		✖
8		served_by_nginx	http://phpmyadmin.readthedocs.org/cs/latest/	True		✖
9		served_by_nginx	http://phpmyadmin.readthedocs.org/en/latest/	True		✖
10		served_by_nginx	https://ericholschercom.readthedocs.org/en/latest/	True		✖
11		served_by_nginx	https://ericholschercom.readthedocs.org/about/	True		✖
12		served_by_nginx	https://ericholschercom.readthedocs.org/en/latest/about/	True		✖
13		served_by_nginx	http://docs.pylonsproject.org/projects/pyramid/en/latest/	True		✖
14		served_by_nginx	http://docs.pylonsproject.org/projects/pyramid-arnon/en/latest/genindex.html	True		✖
15		served_by_django	https://readthedocs.org/search/	True		✖
16		served_by_django	https://readthedocs.org/api/v1/?format=json	True		✖
17		served_by_django	https://readthedocs.org/accounts/login/	True		✖
18		served_by_django	https://readthedocs.org/accounts/login/	True		✖
19		served_by_perl	http://www.perl.org/	False		✖
20		served_by_perl	https://ericholschercom.readthedocs.org/	False		✖
21		served_by_perl	http://www.microsoft.com	False		✖
22		served_by_nginx	http://www.microsoft.com	False		✖
23		served_by_perl	http://my.cofc.edu	False		✖
24		served_by_nginx	http://my.cofc.edu	False		✖
25		served_by_django	http://my.cofc.edu	True		✖

Michael

The greatest issue we faced during this project was one echoed by many teams throughout this process, and that was meeting as a group. Given our widely varying schedules, meeting in person to accomplish tasks was harder than any other aspect of this project. Because of this, we were rarely able to meet in person, and though we couldn't always get together, we met enough, and communicated well enough, to accomplish what we set out to do. This, as with any other project where more than one person is involved, is all about communicating well and executing tasks through communication. Given that we were rarely able to meet, this project would have not been successful without communication.

I also learned a good bit about Linux, working from the command line, and dependencies. Very early on, I was unable to get the project code to compile, despite having followed the instructions outlined in the source documentation. After some research, it ultimately turned out to be a dependency problem caused by missing libraries on my clean, virtual Linux machine. Were this a well-used machine, these libraries would have probably already been installed at some point, but being a clean image, they were not, causing problems along the way. Once these dependencies were resolved, everything compiled without issue.

Finally, we learned the importance of naming conventions. After deciding on our test methods, our initial attempts to import the file into our project failed, and it took some time to figure out why. As it turns out, Python names that contain dashes are not eligible for import, and the file containing our test methods was named with dashes. We ultimately solved this problem, but were the file named using proper conventions, this would not have been an issue.

Venessa

This project was beneficial to me because unlike most classes, this class gave us a taste of what we will actually be doing in the real world. More than likely, as software engineers we will be consistently working in groups with deadlines, or in our case, deliverables. There were many struggles that our team had to overcome, scheduling being the biggest problem. However, we worked around the difficulties and completed everything on time. This project also helped up work on our communication skills that during some steps were not the greatest. Before this class, I had never worked with virtual machines and now I have a decent understanding of VMware Player and Ubuntu enough to put on my resume. In fact, I found myself smiling whenever companies at the career fair mentioned that they used Ubuntu. Overall, I believe this project has proven to be helpful for real life experience and has given us great preparation for 462.

Laura

This project was incredibly helpful to me. It was quite the challenge to work with a group and figure out the times at which we could meet. Although I think we all gained some valuable project management experience. As I had just started using Linux Mint (Ubuntu) as my main operating system this project also helped me learn a lot about Linux and become much more comfortable with it. I also became much more proficient with the terminal and now feel completely at ease. This process was helped along because of our choice to write our test framework script in Bash. I also learned a few new things about Python. Lastly, although I maintain that Git is much easier to work with, it was valuable to work with SVN. And I gained a new appreciation for using version control in group projects.

The following are requirements for the installation and use of Read the Docs in a Linux environment:

- 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

For use of our testing framework only the following are required:

- Ubuntu 14.04
- Python 2.7
- Pip
- Git or SVN