django-with-asserts Documentation

Release 0.0.2dev

John Paulett

November 03, 2014

Contents

django-with-asserts offers an easier way to test HTML content than Django's standard `assertContains(response, ..., html=True)` test.

Using lxml and the `with` statement, django-with-asserts exposes several new assertions, which make your tests more explicit and more concise by focusing on the attributes, values, and content that is relevant to your testing.

Instead of boilerplate that includes unimportant checks of `maxlength`:

```
self.assertContains(
    resp,
    '<input id="id_email" type="text" name="email" maxlength="75" value="bob@example.com>',
    html=True
)
```

Reduce your assertion to only test the relevant content:

```
with self.assertHTML(resp, 'input[name="email"]') as (elem,):
    self.assertEqual(elem.value, 'bob@example.com')
```

django-with-asserts employs the `with` statement to create a DSL for testing HTML. It uses cssselect to provide Level3 CSS selectors (see supported selectors). It returns the matching `lxml.html.HtmlElement` instances as the with statement's target.

`assertHTML()` and `assertNotHTML()` are similar to Django's `assertContains()` and `assertNotContains()`

CHAPTER 1

Rationale

django-with-asserts technically strays from the original intent of context managers in Python (using the `with` statement as an advanced `try` / `except` / `finally` construct). Instead it uses the with statement as a mini domain specific language. This approach allows for cleanly testing multiple parts of the response (note, the with statement does not introduce a new scope, so this is mainly cosmetic):

```
with self.assertHTML(resp, 'input[name="email"]') as (elem,):
    self.assertEqual(elem.value, 'bob@example.com')
with self.assertHTML(resp, 'input[name="first_name"]') as (elem,):
    self.assertEqual(elem.value, 'bob')
```

Additionally, django-with-asserts does not aim to replace or reduce the need for functional testing tools like Selenium, windmill, webunit, etc. Instead, django-with-asserts simply aims to provide an easier way to test HTML than currently is provided by `assertContains()`

CHAPTER 2

Usage

django-with-asserts provides two approaches for incorporating its assertions into your test classes. The first, a subclass of Django's `django.test.TestCase`, is provided as a drop-in replacement, `with_asserts.case.TestCase`:

```
from with_asserts import TestCase

class MyTest(TestCase):
    def test_view(self):
        resp = self.client.get('/my-view/')

        with self.assertHTML(resp) as html:
            self.assertEqual(html.find('head/title').text, 'My Title')
```

The second approach is a mixin, `with_asserts.mixin.AssertHTMLMixin`, which is added into your existing `django.test.TestCase` test or custom subclass:

```
from django.test import TestCase
from with_asserts import AssertHTMLMixin

class MyTest(TestCase, AssertHTMLMixin):
    def test_view(self):
        ...
```

At it's simplest, with no selector, `assertHTML()` will parse the `HttpResponse.content` using lxml and return the entire document as an `lxml.html.HtmlElement`. You can use any of lxml's HTML Element methods, its xpath method, or the Element Tree methods (e.g find, findall, and findtext):

```
with self.assertHTML(resp) as html:
    self.assertEqual(html.find('head/title').text, 'My Title')
```

Similar to `assertContains()`, `assertHTML()` will ensure the status code of the HttpResponse.

By using CSS Selectors, like `#container`, `li.menu`, `.footer`, and `input[name="email"]` (see supported selectors), you can obtain a list of matching elements. If no matching elements are found, the assertion with fail:

```
with self.assertHTML(resp, 'li.menu') as elems:
    self.assertEqual(5, len(elems))
```

Using Python's list destructuring, we can directly access individual elements, especially useful if only one or a few matches exist:

```
with self.assertHTML(resp, 'li.active') as (li,):
    self.assertEqual(li.attrib['href'], '/about/')
```

```
with self.assertHTML(resp, 'td.cell') as (first, second):
    self.assertEqual(first.text, '10.5')
    self.assertEqual(second.text, '23')
```

While we can pass an ID selector, we can alternatively pass the *element_id*, which will always return a single HtmlElement if it is found:

```
with self.assertHTML(resp, element_id='container') as elem:
    self.assertEqual(elem.attrib['width'], '100%')
```

Just as `assertNotContains()` is the inverse of assertContains, so to `assertNotHTML()` will ensure that no matching element is found:

```
self.assertNotHTML(resp, 'input[name="old_password"]')
```

CHAPTER 3

API

class with_asserts.mixin.**AssertHTMLMixin**

assertHTML(*response*, *selector=None*, *element_id=None*, *expected=None*, *status_code=200*, *msg=None*)

assertNotHTML(**args*, ***kwargs*)

CHAPTER 4

Future

django-with-asserts is an experiment in creating a DSL for improving testing in Django.

While less impactful, one future improvement is making an `assertJSON`, similar to `assertHTML`.

CHAPTER 5

Install

django-with-asserts is available on PyPI and can be installed with pip:

```
pip install -U django-with-asserts
```

It has a dependency on lxml and cssselect (formerly part of lxml).

CHAPTER 6

Contribute

The code is hosted at https://github.com/johnpaulett/django-with-asserts

We use tox to run the test suite:

```
django-with-asserts$ tox
<snip>
Creating test database for alias 'default'...
.............
----------------------------------------------------------------------
Ran 13 tests in 0.044s

<snip>
  py27-1.5: commands succeeded
  py27-1.4: commands succeeded
  docs: commands succeeded
  congratulations :)
```

A