Flask-Testing Documentation

Release 0.3

Dan Jacob

Dec 31, 2017

Contents

The **Flask-Testing** extension provides unit testing utilities for Flask.

CHAPTER 1

Installing Flask-Testing

Install with **pip** and **easy_install**:

```
pip install Flask-Testing
```

or download the latest version from version control:

```
git clone https://github.com/jarus/flask-testing.git
cd flask-testing
python setup.py develop
```

If you are using **virtualenv**, it is assumed that you are installing **Flask-Testing** in the same virtualenv as your Flask application(s).

CHAPTER 2

Writing tests

Simply subclass the `TestCase` class:

```
from flask_testing import TestCase

class MyTest(TestCase):

    pass
```

You must specify the `create_app` method, which should return a Flask instance:

```
from flask import Flask
from flask_testing import TestCase

class MyTest(TestCase):

    def create_app(self):

        app = Flask(__name__)
        app.config['TESTING'] = True
        return app
```

If you don't define `create_app` a `NotImplementedError` will be raised.

2.1 Testing with LiveServer

If you want your tests done via Selenium or other headless browser like PhantomJS you can use the LiveServerTestCase:

```
import urllib2
from flask import Flask
from flask_testing import LiveServerTestCase

class MyTest(LiveServerTestCase):
```

```
    def create_app(self):
        app = Flask(__name__)
        app.config['TESTING'] = True
        # Default port is 5000
        app.config['LIVESERVER_PORT'] = 8943
        # Default timeout is 5 seconds
        app.config['LIVESERVER_TIMEOUT'] = 10
        return app

    def test_server_is_up_and_running(self):
        response = urllib2.urlopen(self.get_server_url())
        self.assertEqual(response.code, 200)
```

The method `get_server_url` will return http://localhost:8943 in this case.

2.2 Dynamic LiveServerTestCase port

By default, `LiveServerTestCase` will use the pre-defined port for running the live server. If multiple tests need to run in parallel, the `LIVESERVER_PORT` can be set to `0` to have the underlying operating system pick an open port for the server. The full address of the running server can be accessed via the `get_server_url` call on the test case:

```
import urllib2
from flask import Flask
from flask_testing import LiveServerTestCase

class MyTest(LiveServerTestCase):

    def create_app(self):
        app = Flask(__name__)
        app.config['TESTING'] = True

        # Set to 0 to have the OS pick the port.
        app.config['LIVESERVER_PORT'] = 0

        return app

    def test_server_is_up_and_running(self):
        response = urllib2.urlopen(self.get_server_url())
        self.assertEqual(response.code, 200)
```

2.3 Testing JSON responses

If you are testing a view that returns a JSON response, you can test the output using a special `json` attribute appended to the `Response` object:

```
@app.route("/ajax/")
def some_json():
    return jsonify(success=True)

class TestViews(TestCase):
    def test_some_json(self):
```

```
        response = self.client.get("/ajax/")
        self.assertEquals(response.json, dict(success=True))
```

2.4 Opt to not render the templates

When testing with mocks the template rendering can be a problem. If you don't want to render the templates in the tests you can use the `render_templates` attribute:

```
class TestNotRenderTemplates(TestCase):

    render_templates = False

    def test_assert_not_process_the_template(self):
        response = self.client.get("/template/")

        assert "" == response.data
```

The signal will be sent anyway so that you can check if the template was rendered using the `assert_template_used` method:

```
class TestNotRenderTemplates(TestCase):

    render_templates = False

    def test_assert_mytemplate_used(self):
        response = self.client.get("/template/")

        self.assert_template_used('mytemplate.html')
```

When the template rendering is turned off the tests will also run faster and the view logic can be tested in isolation.

2.5 Using with Twill

Twill is a simple language for browsing the Web through a command line interface.

Note: Please note that Twill only supports Python 2.x and therefore cannot be used with Python 3 or above.

`Flask-Testing` comes with a helper class for creating functional tests using Twill:

```
def test_something_with_twill(self):

    with Twill(self.app, port=3000) as t:
        t.browser.go(t.url("/"))
```

The older `TwillTestCase` has been deprecated.

2.6 Testing with SQLAlchemy

This covers a couple of points if you are using **Flask-Testing** with SQLAlchemy. It is assumed that you are using the Flask-SQLAlchemy extension, but if not the examples should not be too difficult to adapt to your own particular setup.

First, ensure you set the database URI to something other than your production database ! Second, it's usually a good idea to create and drop your tables with each test run, to ensure clean tests:

```
from flask_testing import TestCase

from myapp import create_app, db

class MyTest(TestCase):

    SQLALCHEMY_DATABASE_URI = "sqlite://"
    TESTING = True

    def create_app(self):

        # pass in test configuration
        return create_app(self)

    def setUp(self):

        db.create_all()

    def tearDown(self):

        db.session.remove()
        db.drop_all()
```

Notice also that `db.session.remove()` is called at the end of each test, to ensure the SQLAlchemy session is properly removed and that a new session is started with each test run - this is a common "gotcha".

Another gotcha is that Flask-SQLAlchemy **also** removes the session instance at the end of every request (as should any thread safe application using SQLAlchemy with **scoped_session**). Therefore the session is cleared along with any objects added to it every time you call `client.get()` or another client method.

For example:

```
class SomeTest(MyTest):

    def test_something(self):

        user = User()
        db.session.add(user)
        db.session.commit()

        # this works
        assert user in db.session

        response = self.client.get("/")

        # this raises an AssertionError
        assert user in db.session
```

You now have to re-add the "user" instance back to the session with `db.session.add(user)`, if you are going to make any further database operations on it.

Also notice that for this example the SQLite in-memory database is used : while it is faster for tests, if you have database-specific code (e.g. for MySQL or PostgreSQL) it may not be applicable.

You may also want to add a set of instances for your database inside of a `setUp()` once your database tables have been created. If you want to work with larger sets of data, look at Fixture which includes support for SQLAlchemy.

CHAPTER 3

Running tests

3.1 with unittest

I recommend you to put all your tests into one file so that you can use the `unittest.main()` function. This function will discover all your test methods in your *`TestCase`* classes. Remember, the names of the test methods and classes must start with `test` (case-insensitive) so that they can be discovered.

An example test file could look like this:

```
import unittest
import flask_testing

# your test cases

if __name__ == '__main__':
    unittest.main()
```

Now you can run your tests with `python tests.py`.

3.2 with nose

The nose collector and test runner works also fine with Flask-Testing.

CHAPTER 4

Changes

4.1 0.7.1 (19.12.2017)

- Reverts the request context changes from `0.7.0`. This change broke backwards compatibility so it will be moved to a major version release instead.

4.2 0.7.0 (18.12.2017)

- Changes the way request contexts are managed. Let's Flask be responsible for the context, which fixes some subtle bugs.

4.3 0.6.2 (26.02.2017)

- Add support for OS chosen port in `LiveServerTestCase`
- Better error messages when missing required modules
- `assertRedirects` now supports all valid redirect codes as specified in the HTTP protocol
- Fixed bug that caused `TypeError` instead of `AssertionError` when testing against used templates
- Fixed bug in `assertRedirects` where the location was not being checked properly

4.4 0.6.1 (03.09.2016)

- Fix issues that prevented tests from running when blinker was not installed

4.5 0.6.0 (02.09.2016)

- `LiveServerTestCase` will now start running as soon as the server is up
- `assertRedirects` now respects the `SERVER_NAME` config value and can compare against absolute URLs
- Compatibility with Flask 0.11.1

4.6 0.5.0 (12.06.2016)

- Improvements to `LiveServerTestCase`
  - The test case will now block until the server is available
  - Fixed an issue where no request context was available
  - Fixed an issue where tests would be run twice when `DEBUG` was set to True
- Add missing message arguments for assertRedirects and assertContext
- Better default failure message for assertRedirects
- Better default failure message for assertTemplateUsed
- Fix an issue that caused the `render_templates` option to not clean up after itself if set to False
- Update docs to use new Flask extension import specification

4.7 0.4.2 (24.07.2014)

- Improved teardown to be more graceful.
- Add `message` argument to `assertStatus` respectively all assertion methods with fixed status like `assert404`.

4.8 0.4.1 (27.02.2014)

This release is dedicated to every contributer who made this release possible. Thank you very much.

- Python 3 compatibility (without twill)
- Add `LiveServerTestCase`
- Use unittest2 backports if available in python 2.6
- Install multiprocessing for python versions earlier than 2.6

4.9 0.4 (06.07.2012)

- Use of the new introduced import way for flask extensions. Use `import flask.ext.testing` instead of `import flaskext.testing`.
- Replace all `assert` with `self.assert*` methods for better output with unittest.
- Improved Python 2.5 support.

- Use Flask's preferred JSON module.

CHAPTER 5

API

class flask_testing.**TestCase**(*methodName='runTest'*)

assert200(*response*, *message=None*)
Checks if response status code is 200

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert400(*response*, *message=None*)
Checks if response status code is 400

Versionadded 0.2.5

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert401(*response*, *message=None*)
Checks if response status code is 401

Versionadded 0.2.1

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert403(*response*, *message=None*)
Checks if response status code is 403

Versionadded 0.2

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert404(*response*, *message=None*)
Checks if response status code is 404

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert405(*response*, *message=None*)
Checks if response status code is 405

Versionadded 0.2

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert500(*response*, *message=None*)
Checks if response status code is 500

Versionadded 0.4.1

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assertContext(*name*, *value*, *message=None*)
Checks if given name exists in the template context and equals the given value.

Versionadded 0.2

Parameters

- **name** – name of context variable
- **value** – value to check against

assertMessageFlashed(*message*, *category='message'*)
Checks if a given message was flashed. Only works if your version of Flask has message_flashed signal support (0.10+) and blinker is installed.

Parameters

- **message** – expected message
- **category** – expected message category

assertRedirects(*response*, *location*, *message=None*)
Checks if response is an HTTP redirect to the given location.

Parameters

- **response** – Flask response
- **location** – relative URL path to SERVER_NAME or an absolute URL

assertStatus(*response*, *status_code*, *message=None*)
Helper method to check matching response status.

Parameters

- **response** – Flask response
- **status_code** – response status code (e.g. 200)
- **message** – Message to display on test failure

assertTemplateUsed(*name*, *tmpl_name_attribute='name'*)
Checks if a given template is used in the request. Only works if your version of Flask has signals support (0.6+) and blinker is installed. If the template engine used is not Jinja2, provide `tmpl_name_attribute` with a value of its *Template* class attribute name which contains the provided `name` value.

Versionadded 0.2

Parameters

- **name** – template name
- **tmpl_name_attribute** – template engine specific attribute name

assert_200(*response*, *message=None*)
Checks if response status code is 200

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert_400(*response*, *message=None*)
Checks if response status code is 400

Versionadded 0.2.5

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert_401(*response*, *message=None*)
Checks if response status code is 401

Versionadded 0.2.1

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert_403(*response*, *message=None*)
Checks if response status code is 403

Versionadded 0.2

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert_404(*response*, *message=None*)
Checks if response status code is 404

Parameters

- **response** – Flask response

- **message** – Message to display on test failure

assert_405(*response*, *message=None*)
Checks if response status code is 405

Versionadded 0.2

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert_500(*response*, *message=None*)
Checks if response status code is 500

Versionadded 0.4.1

Parameters

- **response** – Flask response
- **message** – Message to display on test failure

assert_context(*name*, *value*, *message=None*)
Checks if given name exists in the template context and equals the given value.

Versionadded 0.2

Parameters

- **name** – name of context variable
- **value** – value to check against

assert_message_flashed(*message*, *category='message'*)
Checks if a given message was flashed. Only works if your version of Flask has message_flashed signal support (0.10+) and blinker is installed.

Parameters

- **message** – expected message
- **category** – expected message category

assert_redirects(*response*, *location*, *message=None*)
Checks if response is an HTTP redirect to the given location.

Parameters

- **response** – Flask response
- **location** – relative URL path to SERVER_NAME or an absolute URL

assert_status(*response*, *status_code*, *message=None*)
Helper method to check matching response status.

Parameters

- **response** – Flask response
- **status_code** – response status code (e.g. 200)
- **message** – Message to display on test failure

assert_template_used(*name*, *tmpl_name_attribute='name'*)
Checks if a given template is used in the request. Only works if your version of Flask has signals support (0.6+) and blinker is installed. If the template engine used is not Jinja2, provide

`tmpl_name_attribute` with a value of its *Template* class attribute name which contains the provided `name` value.

Versionadded 0.2

Parameters

- **name** – template name
- **tmpl_name_attribute** – template engine specific attribute name

create_app()
Create your Flask app here, with any configuration you need.

get_context_variable(*name*)
Returns a variable from the context passed to the template. Only works if your version of Flask has signals support (0.6+) and blinker is installed.

Raises a ContextVariableDoesNotExist exception if does not exist in context.

Versionadded 0.2

Parameters **name** – name of variable

flask_testing.**Twill**
alias of `Error`

flask_testing.**TwillTestCase**
alias of `Error`

Python Module Index

f

Index