Flask-Testing Documentation

Release 0.3

Dan Jacob

August 02, 2012

CONTENTS

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

CHAPTER

ONE

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

TWO

WRITING UNIT TESTS

Simply subclass the TestCase class:

```
from flask.ext.testing import TestCase

class MyTest(TestCase):

    pass
```

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

```
from flask.ext.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.

CHAPTER

THREE

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))
```

CHAPTER

FOUR

USING WITH TWILL

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

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.

CHAPTER

FIVE

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.ext.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 threadsafe 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

SIX

CHANGELOG

- **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.
  - Sounds crazy but improved python2.5 support.
  - Use Flask's preferred JSON module.

CHAPTER

SEVEN

API

PYTHON MODULE INDEX