Viceroy Documentation

Release 0.1

Jonas Obrist

June 16, 2016

Contents

Warning: This project is still in an early stage and quite a few things are not supported yet. Feel free to give it a try, but don't expect this to be production ready yet.
I will likely change 100% of the APIs in new releases until I hit 1.0.

CHAPTER 1

What is Viceroy

Viceroy is a Python library that allows you to run Javascript tests in a browser, using Selenium, and reports them just like normal Python unit tests.

The main goal is to streamline continuous integration of Python web projects that would like to test their Javascript.

For now, QUnit and Jasmine are supported, but you can add support for your preferred testing library if you want to.

As for Python frameworks, Flask and Django are supported out of the box, but again you may feel free to add support for the framework of your choice.

Note that how to extend Viceroy is not yet documented, as the API is not finalized yet.

CHAPTER 2

Contents

2.1 Installation

2.1.1 Requirements

Python

Viceroy requires Python 3.3 or higher.

Dependencies

You need to install the selenium Python package.

2.1.2 Installation

Simply `pip install viceroy` in your virtualenv, assuming you have the dependencies of selenium installed and a C compiler available.

2.2 Quickstart

Viceroy expects you to have a view in your app that loads your Javascript testing framework, the Viceroy Javascript library and your testing code. For both Django and Flask it will provide a flag for when your code is in Viceroy testing, so you can load those files conditionally if you wish to re-use a template used in your app.

No matter what you're using on the Javascript or Python side, your main entry point into Viceroy is `viceroy.api.build_test_case()`, which will build a test case class for you (and return it). It takes the following arguments:

- `class_name`: The name of the class you want to build.
- `source_file`: Full path to the source file.
- `scanner_class`: Class that scans your `source_file` for test methods, this will depend on your Javascript framework of choice.
- `base_class`: Base class for the test case, this will depend on your Python framework of choice.
- All other keyword arguments will be set as class attributes on the class being built. A common keyword argument is `viceroy_url` which is the URL at which your test view is served.

2.2.1 Django

Viceroy will set `settings.VICEROY_TESTING` to `True` when running Viceroy tests.

Setup

Include `viceroy.contrib.django.urls` in your urls. This will allow you to serve the Viceroy Javascript libraries from that URL. This view will only work if `settings.VICEROY_TESTING` is `True` and otherwise will always return a 404 response.

You may optionally install the `viceroy.contrib.django.context_processor.viceroy` context processor, which will set the template context variable `VICEROY_TESTING` to `True` if the template is being rendered in a Viceroy test.

QUnit

To use QUnit, build a view and template that load the following Javascript files. Order matters:

- QUnit itself.
- `viceroy.js` which can be served from the urls you included above.
- `qunit-bridge.js` which can also be served from the urls you included above.
- Your test file.

Assuming your test file is located at `/path/to/tests.js` on your filesystem, and the url to the view that loads the tests at `/test-url/` you would build the test class as follows:

```
from viceroy.api import build_test_case
from viceroy.contrib.django import ViceroyDjangoTestCase
from viceroy.contrib.qunit import QUnitScanner

MyTestCase = build_test_case(
    'MyTestCase',
    '/path/to/tests.js',
    QUnitScanner,
    ViceroyDjangoTestCase,
    viceroy_url='/test-url/'
)
```

Jasmine

Jasmine works mostly the same as *QUnit*, however you need to load the following Javascript files, again order matters:

- Jasmine itself.
- `viceroy.js` from the included urls.
- `jasmine-bridge.js` from the included urls.
- Your test file.

Build the test case as described in *QUnit*, but replace the `QUnitScanner` with `JasmineScanner`, which you can import from `viceroy.contrib.jasmine`.

2.2.2 Flask

With Flask, you're responsible to serve the Viceroy Javascript files yourself. For your convenience, you may use `viceroy.constants.VICEROY_STATIC_ROOT` which is the path to the directory holding `viceroy.js`, `qunit-bridge.js` and `jasmine-bridge.js`.

`VICEROY_TESTING` in your `app.config` will be set to `True` during Viceroy testing.

Testing in Flask works pretty much the same as *Django*, but substitute the `ViceroyDjangoTestCase` with `ViceroyFlaskTestCase` and make sure to pass your Flask app as the `viceroy_flask_app` keyword argument to `build_test_case`.

Configuration

Besides the required `viceroy_flask_app` extra keyword argument to `build_test_case` there are the following optional configuration values:

- `viceroy_flask_port`: Port to use, default `5000`.
- `viceroy_flask_silent`: Whether to hide the Werkzeug log, default `True`.

2.3 Indices and tables

- genindex
- modindex
- search