Alfajor Documentation

Release tip

The Alfajor Team

July 29, 2014

Contents

CHAPTER 1

Contents

1.1 Introduction to Alfajor

1.1.1 Installing Alfajor

The Alfajor repository is available for check out from our github page.

Setup:

1) Create and activate a virtualenv (optional) 1) Next we need to install dependencies. From the top of the distribution run:

```
$ python setup.py develop
```

1. Next install nose using either:

   ```
   $ easy_install nose
   ```

OR:

```
$ pip install nose
```

If you don't have Selenium installed, download Selenium Server. All you need is the selenium-server.jar, and no configuration is required.

Run it with:

```
$ java -jar selenium-server.jar
```

See it in action:

After following the steps above, the Alfajor plugin should be available and listing command-line options for nose. You can verify this by typing:

```
$ nosetests --help
```

To run the standard tests that use an in-process web app through a WSGI interface, simply type:

```
$ nosetests
```

To run the same tests but using a real web browser, type:

```
$ nosetests --browser=firefox
```

You can use all sorts of browsers

A list of valid browsers is: *firefox *mock *firefoxproxy *pifirefox *chrome *iexploreproxy *iexplore *firefox3 *safariproxy *googlechrome *konqueror *firefox2 *safari *piiexplore *firefoxchrome *opera *iehta *custom

.ini Files

The main action of Alfajor is directed through an *alfajor.ini* file. At the simplest, this can be anywhere on the filesystem (see the –alfajor-config option in nose) or placed in the same directory as the .py file that configures the WebBrowser. See *tests/webapp/{__init__.py,alfajor.ini}*.

1.1.2 Getting started using Alfajor

In this tutorial we will be showing you how to use Alfajor to greatly enhance the tastiness of your functional testing.

We are assuming you have properly *installed Alfajor and ran the browser based test suite to ensure everything is working*.

The Example Application

To start with, you are going to need a web application to test. Since we are nice, we've included an example application in the project for you. You can find it in ./tests/examples. The application is a simple recommendation system that under the covers is just an extremely simple WSGI application.

Let's start by getting to know our example application. You should be able to start it up using the following command:

```
$ alfajor-invoke docs.examples.webapp:run
```

Now if you point your web browser at http://127.0.0.1:8009, you should see the silly example application. Try filling in the form a few different ways and get familiar with the output.

Alright, now let's press CTRL-c to stop your server, we've got some testing to do.

Testing the Example Application

So now we've seen the application in all it's glory, we'd better write a few tests to make sure it is functioning as expected.

Again, cuzz we are such good guys over here, we've started a little test suite that you can kick off pretty easily. To run the tests simply type:

```
$ nosetests docs.examples.test_simple
```

So let's go on a line-by-line tour through the nosetest test_name_entry in tests/examples/test_simple

```
browser.open('/')
```

It is time to introduce you to the browser object. It is going to be available to you somewhat magically throughout each of your tests, all you need to do is import it from your base testing module.

The `open()` method will attempt to load the **url** that is passed in. Absolute urls, as shown in the example code, will work by appending to your *base_url* or *server_url* setting.

Alright now the browser object has loaded the url, it is ready to be poked at.

The `document` represents the HTMLDocument element. If you are familiar with CSS selectors this type of traversal, should be fairly straightforward. Basically what this is saying is, get the `DOMElement` element on the page with the *id* attribute of *mainTitle*. Once that is found use the `text_content` which returns text in all of the text nodes in between the found tags, to see if our value is inside.

Note: Since the specification of the attribute id states that there can be only one element with this id, in an HTML document, this lookup will only return the first occurrence of the id. If you are testing invalid HTML, consider yourself warned.

This could very easily be rewritten as such:

Okay so the next line we want to enter some data into a form

So we get a handle to the input element that we want to add and simply set the `value` attribute.

1.2 Browsers

1.2.1 WSGI

Capabilities

- cookies
- headers
- in-process
- status

1.2.2 Selenium

Capabilities

- cookies
- javascript
- selenium
- visibility

1.2.3 Zero

Capabilities

- no capabilities

1.3 Alfajor-flavored LXML

- Indexing
- Containment

- printing
- text_content
- innerHTML
- forms
- differences from lxml.html

1.4 alfajor.ini

```
[default-targets]
default+browser=wsgi

[self-tests]
wsgi=wsgi
*=selenium
zero=zero

[self-tests+browser.wsgi]
server-entry-point = tests.browser.webapp:webapp()
base_url = http://localhost

[self-tests+browser.selenium]
cmd = alfajor-invoke tests.browser.webapp:run
server_url = http://localhost:8008
ping-address = localhost:8008
```