Testing Ruby's CGI

Oct 12, 2018

CGI is a standard for generating HTML pages from scripts executed as child processes by a web server. I explained CGI and how I use it in a previous post, CGI: Ruby's Bare Metal (../../../../2015/01/05/cgi-rubys-bare-metal/). I like to use CGI because it means that I don't have to run any Ruby app server (e.g. puma, passenger, unicorn) 24/7. Less moving parts == more robust!

Those app servers get most of the press these days so sadly CGI doesn't have much support and documentation; I couldn't find anything on how to **test** CGI scripts. These scripts handle my business, they need to work so I decided to put some effort into testing CGI. You, dear reader, are the benefactor!

Two things I learned:

1. Webrick, Ruby's built-in HTTP server, does have support for calling CGI scripts. This will be critical for us, it means we don't have to install Apache or nginx to call the CGI scripts.
2. Because CGI outputs raw HTML, we'll need to do browser-level testing using Capybara with headless chrome or another automated browser.

Great! Our test suite will be a Ruby process with two things:

1. Webrick serving CGI requests
2. Capybara requesting URLs from Webrick

How can a Ruby process do two things at once? With Threads, of course!

Webrick

Webrick is set up out of the box to serve any `.cgi` files as Ruby CGI scripts. You point Webrick to your `DocumentRoot` as normal. Note that since Capybara is going to be driving the tests, we need Webrick to execute in

another thread so both the HTTP client and server can execute concurrently:

```
WEBRICK = Thread.new do
  require 'webrick'

  server = WEBrick::HTTPServer.new(
    :Port => 8999,
    :DocumentRoot => File.expand_path("../..", __FILE__),
  )
  trap('INT') { server.shutdown }

  puts "Starting Webrick on port 8999"
  server.start
end
```

Capybara

Most of the Capybara integration is straight out of the README, there is only one trick necessary: since we are starting our own custom Webrick server in a separate thread, we need to tell Capybara not to start a server on its own and point it to ours instead.

```
Capybara.run_server = false
Capybara.app_host = 'http://localhost:8999 (http://localhost:8999)'
```

After an hour of fiddling, here's the `test/helper.rb` I came up with:

```
Thread.report_on_exception = true

require 'minitest/autorun'
require 'capybara/minitest'
Capybara.run_server = false

# I found poltergeist quick to install and worked first time, YMMV
require 'capybara/poltergeist'
Capybara.javascript_driver = :poltergeist

class CGITest < Minitest::Test
  include Capybara::DSL
  include Capybara::Minitest::Assertions

  def setup
    Capybara.current_driver = Capybara.javascript_driver
  end

  def teardown
    Capybara.reset_sessions!
    Capybara.use_default_driver
  end
end


# Here's the trick:
# We need to run the Webrick server in a separate thread so the
# testcases can make requests and block, waiting for the response.
WEBRICK = Thread.new do
  require 'webrick'

  server = WEBrick::HTTPServer.new(
    :Port => 8999,
    :DocumentRoot => File.expand_path("../..", __FILE__),
  )
  trap('INT') { server.shutdown }

  puts "Starting Webrick on port 8999"
  server.start
end

# Point Capybara to Webrick!
Capybara.app_host = 'http://localhost:8999 (http://localhost:8999)'

# give webrick time to boot before the tests can run
sleep 1
```

Now I create `test/smoke_test.rb`:

```
require_relative './helper'

class SmokeTest < CGITest
  def test_pro_navigation
    visit("/spro/new.cgi")
    assert page.has_content?('Subscribe to')

    visit("/spro/update.cgi")
    assert page.has_content?('Update your')

    visit("/spro/cancel.cgi")
    assert page.has_content?('Cancel your')

    visit("/spro/delete.cgi")
    assert page.has_content?('Confirm your')
  end
end
```

That's a quick test to make sure the billing scripts for Sidekiq Pro (https://billing.contribsys.com/spro/) will render correctly and not crash. Now with a little test coverage I've got more confidence that changes won't break these scripts.